



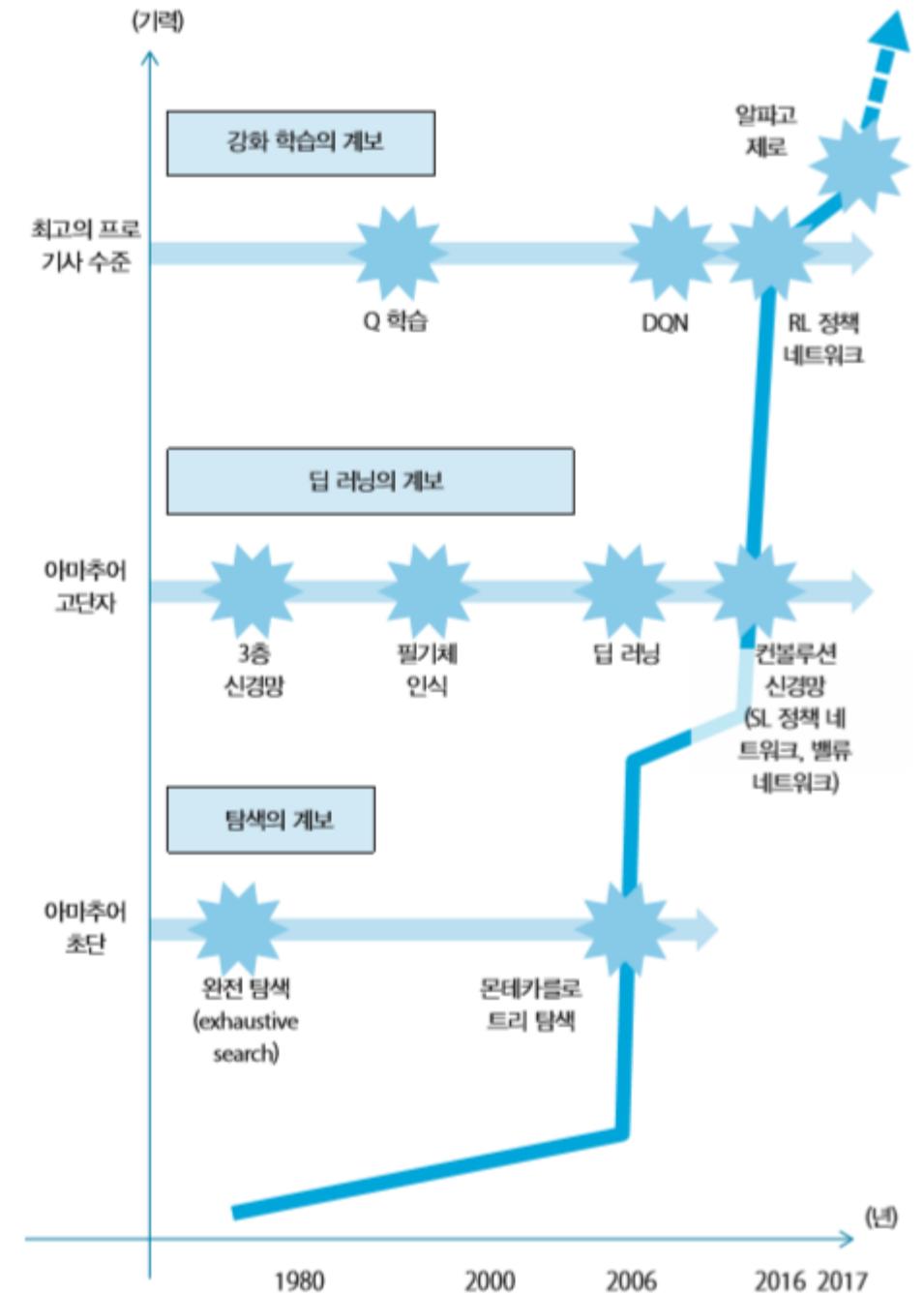
A l p h a g o  
알파고를  
분석하며 배우는  
인공지능  
Artificial Intelligence

오크키 토모시 저음 | 정인식 읊김

# Alpha Go

중앙대학교

박호현



# 참고 자료

- 오츠키토모시/정인식, 알파고를 분석하며 배우는 인공지능, 제이펍, 2019
- 유튜브, 전민영, 노승은, 알파고 논문 리뷰, 팡요랩
- David Silver, ... Demis Hassabis, Mastering the game of Go with deep neural networks and tree search, Nature, 2016. 01.
- David Silver, ... Demis Hassabis, Mastering the game of Go without human knowledge, Nature, 2017. 10.

# 목차

1. 알파고의 등장
  1. 지도학습 – 로지스틱 회귀
2. 딥러닝 – CNN 기반의 SL
3. 강화학습 – Policy Gradient
4. 탐색 – MCTS
5. 알파고의 완성
6. 알파고 제로

# 알파고의 등장

게임 AI의 역사는 인공지능의 아버지라고도 불리는 앤런 튜링의 시대부터 연구되어 왔다. 체스나 장기는 이미 인간의 챔피언 수준에 도달하였지만, 바둑에 관해서는 “앞으로 10년은 걸린다”고들 말하고 있었다. 그런 상황에서 현재 데미스 하사비스가 이끄는 구글 딥 마인드가 알파고를 들고나와 순식간에 세계 최고의 프로 바둑 기사의 실력을 능가하고 말았다.

이 장의 마지막 부분에서는 바둑 AI란 무엇인지, 그리고 AI에게 있어 바둑이 얼마나 힘든지에 대해 설명한다. 또한, 기존의 머신 러닝에 의해 ‘다음의 한 수’를 도출하는 방법에 관해서도 설명하겠다.

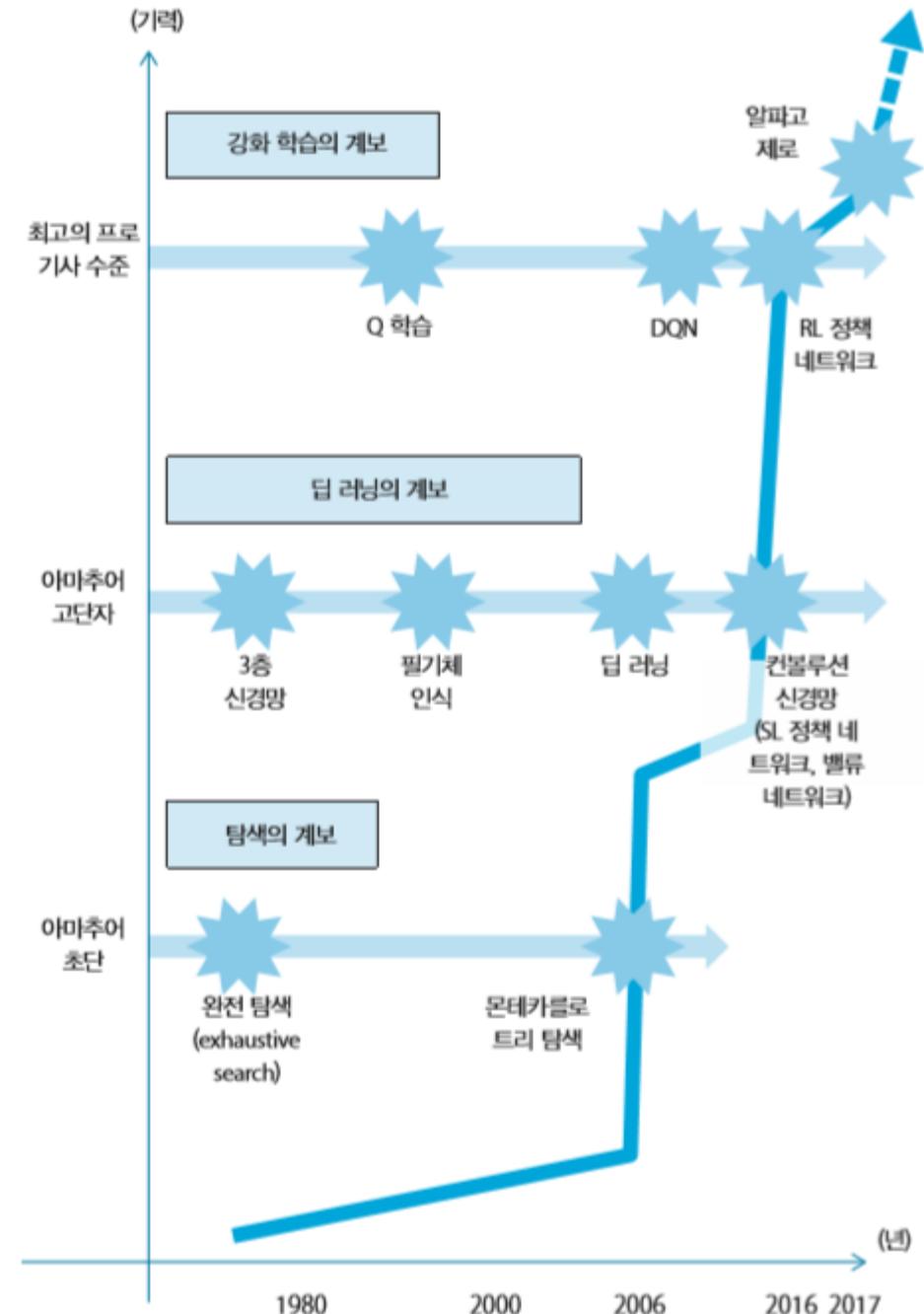
# 1. 알파고의 등장

1.1 게임 AI의 역사와 발전

1.2 천재 데미스 하사비스의 등장

1.3 알파고의 활약

1.4 바둑 AI의 기초

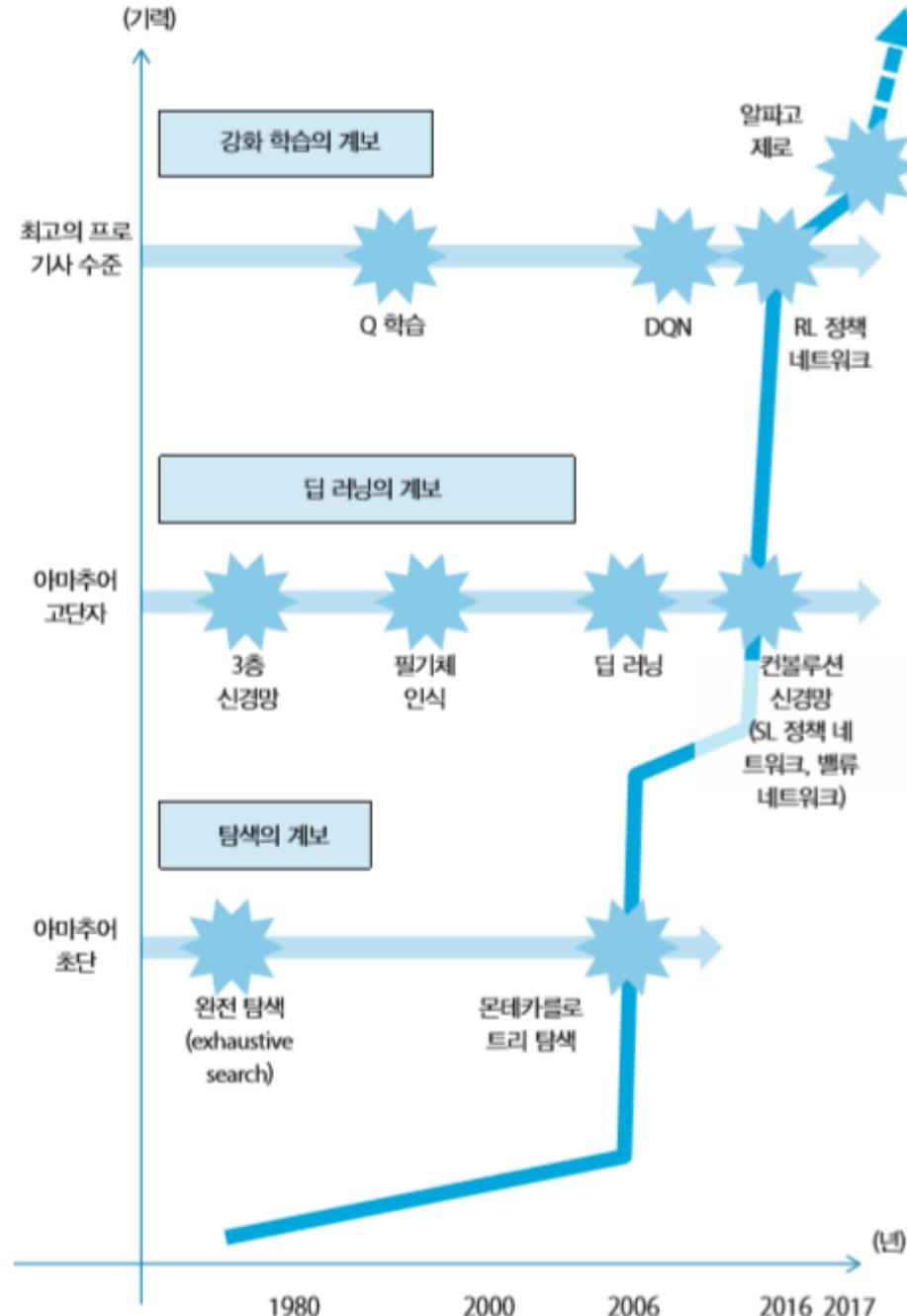


# 알파고의 탄생과 역사

- 딥마인드
- Nature 논문
  - Mastering the game of Go with deep neural networks and tree search, Nature 2016.01
- 알파고와 이세돌
- 알파고 제로
  - Mastering the game of Go without human knowledge, Nature 2017.10

# 바둑 AI의 역사

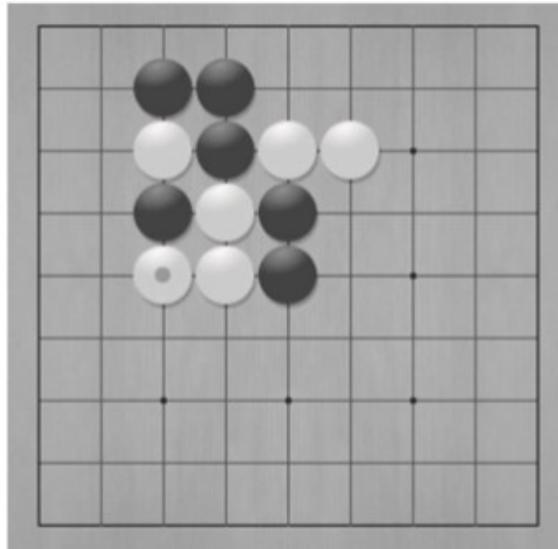
- 몬테카를로 트리 탐색, 딥러닝, 강화 학습 기술의 계보와 바둑 AI의 기력 향상



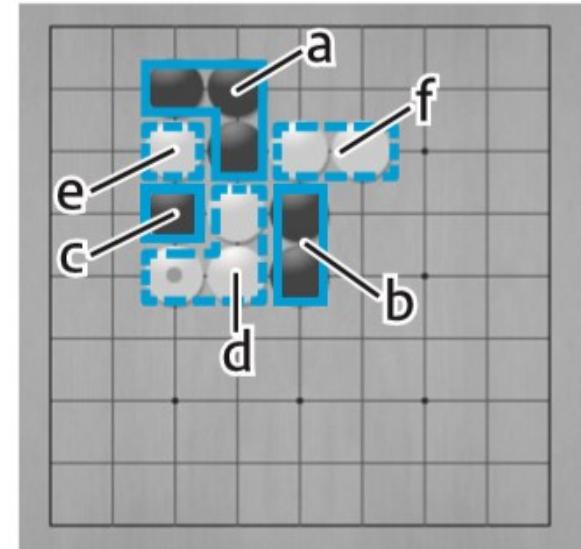
# 바둑 AI의 기초

- 국면 : 바둑판의 상태
- 연: 돌의 묶음
- 호흡점: 연이 앞으로 몇 수에서 잡힐지에 대한 값

(a) 국면의 그림



(b) 연의 분할



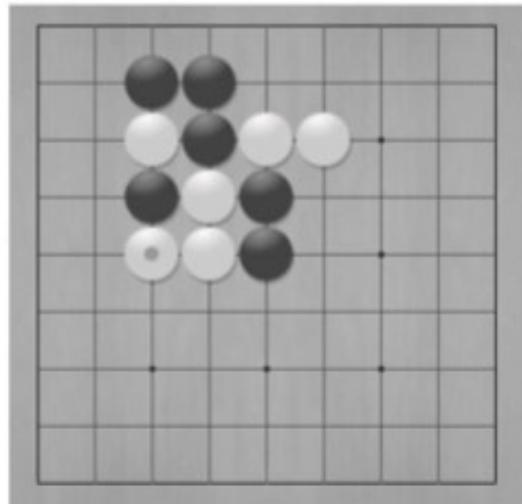
(c) 각 연의 지표

연	a	b	c	d	e	f
소속	흑	흑	흑	백	백	백
돌의 수	3	2	1	3	1	2
호흡점의 수	4	3	1	3	1	4

# 바둑 AI의 기초

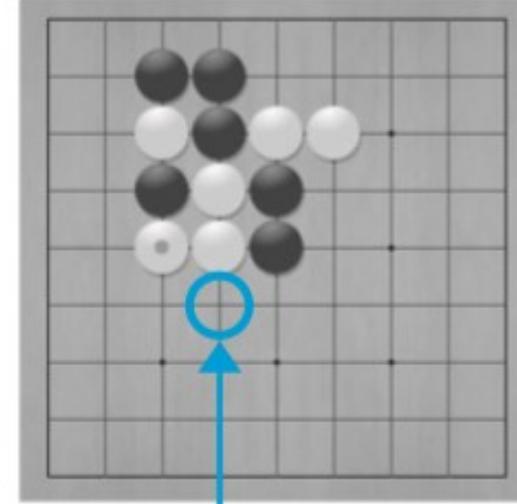
- 바둑의 '다음의 한 수' 태스크란  
➤ 입력 상황에 대해 어디에 둘지를 결정하는 태스크를 말한다.

(a) 입력: 흑의 순서에서 13수째의 국면



69곳의 빈 점 중에서  
어디에 둘지를 선택한다.

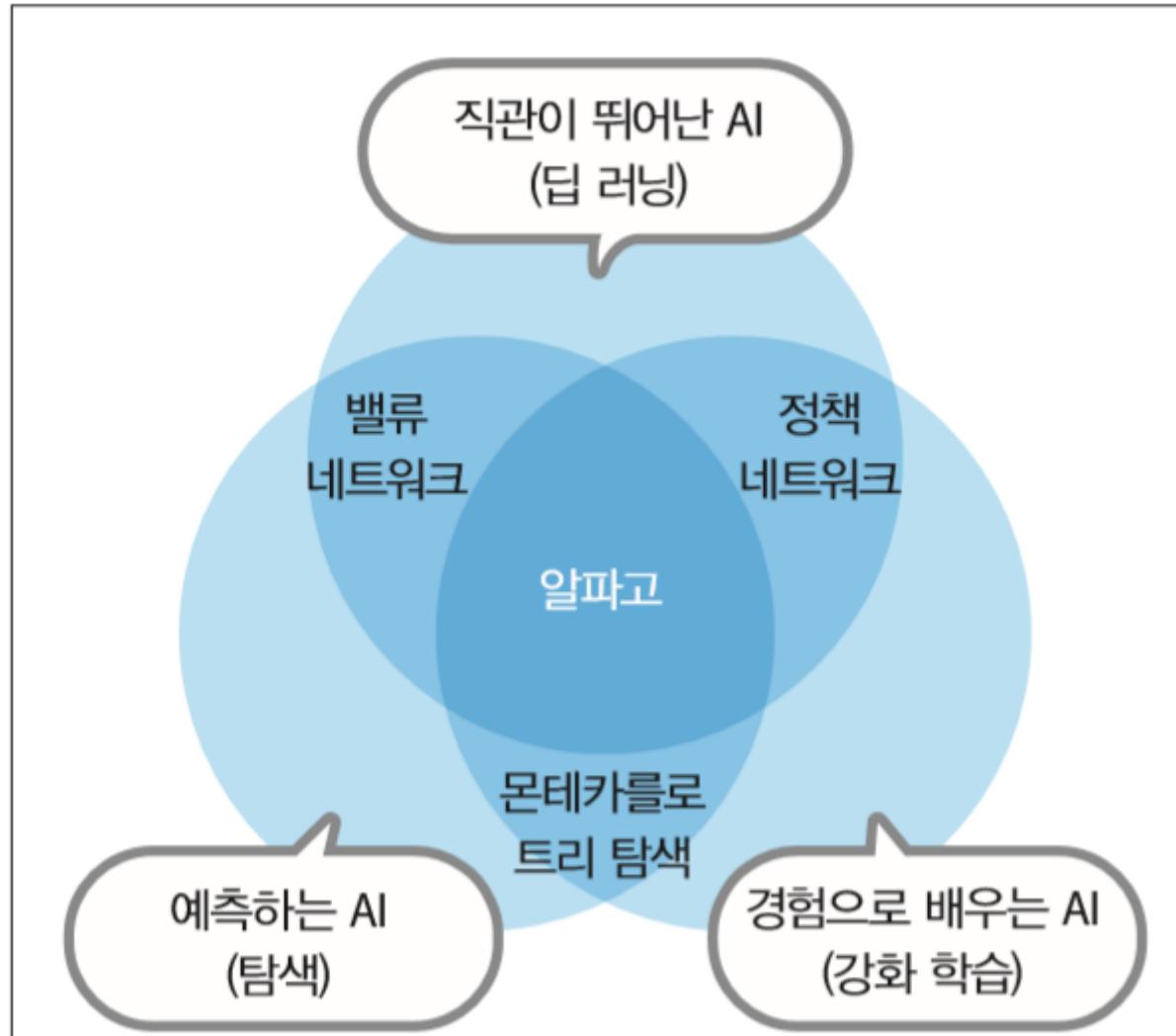
(b) 출력: 어디에 둘지를 결정

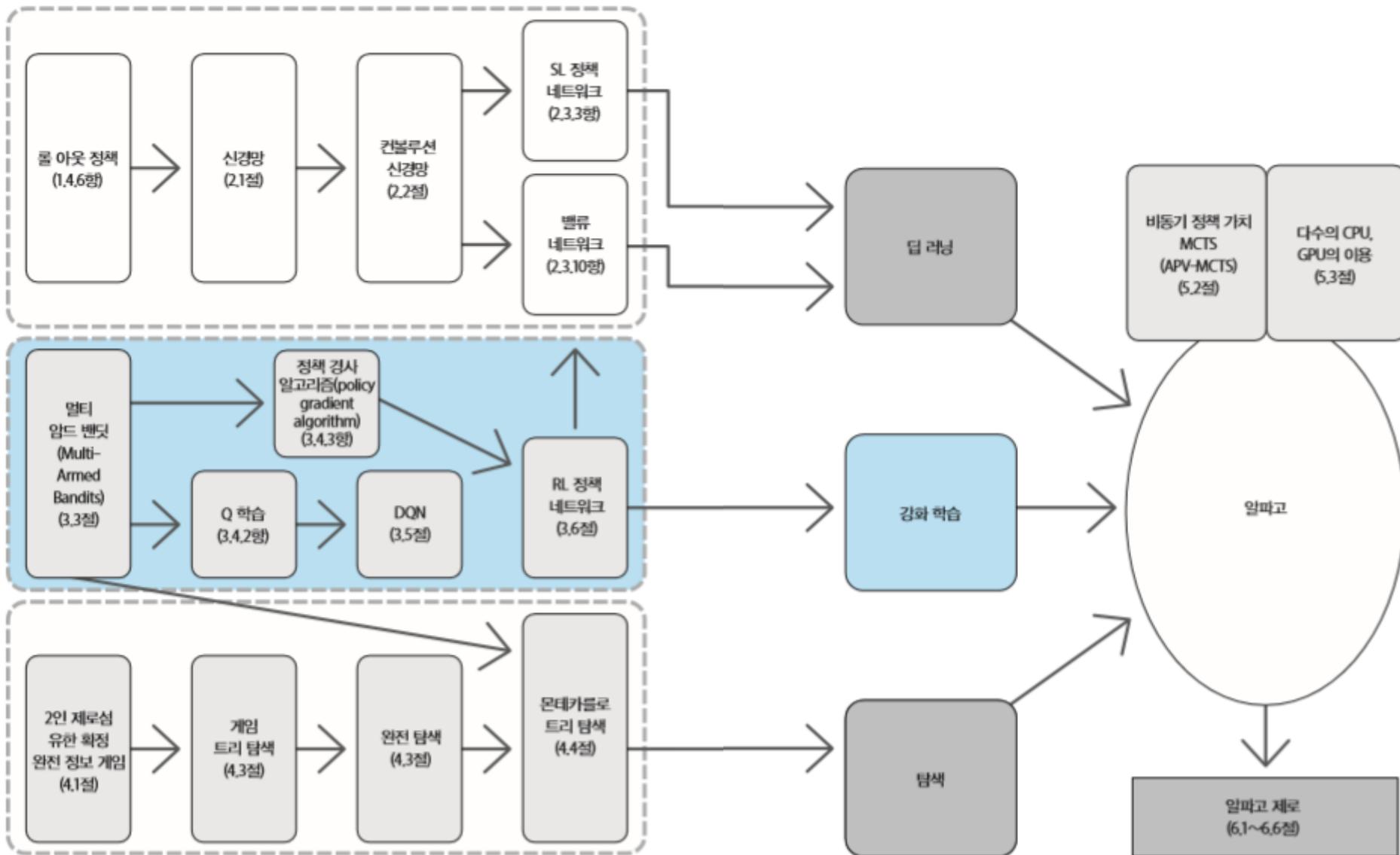


출력

12수째까지 진행된 국면을 입력으로 하고, 13수째의 흑돌을 어디에 둘지를 결정한다.

# 알파고 AI의 3대 요소





# 머신러닝을 이용한 '다음의 한 수' 태스크

## '다음의 한 수' 태스크

- 모든 국면의 수를 기억할려면
  - 탐색공간 :  $250^{150} \approx 10^{360}$
- 검색 기반의 방법
  - 예측을 바탕으로 앞으로 전개 가능한 것을 열거하여 그중에서 가장 좋은 전개를 알아내는 방법
  - Chapter 4
- 머신러닝 기반의 방법
  - 컴퓨터에 '학습'을 시켜서 어떤 태스크에 대한 컴퓨터의 '예측능력'과 '판단능력'을 향상시켜 나가는 방법
  - 지도학습 – Chapter 1.4.5 ~ Chapter 2
  - 강화학습 – Chapter 3

# 기존의 지도 학습 – Logistic Regression

## • 전처리

- 원시 데이터에서 노이즈를 제거하여 (입력, 정답레이블) 쌍을 만든다.

## • 특징 추출

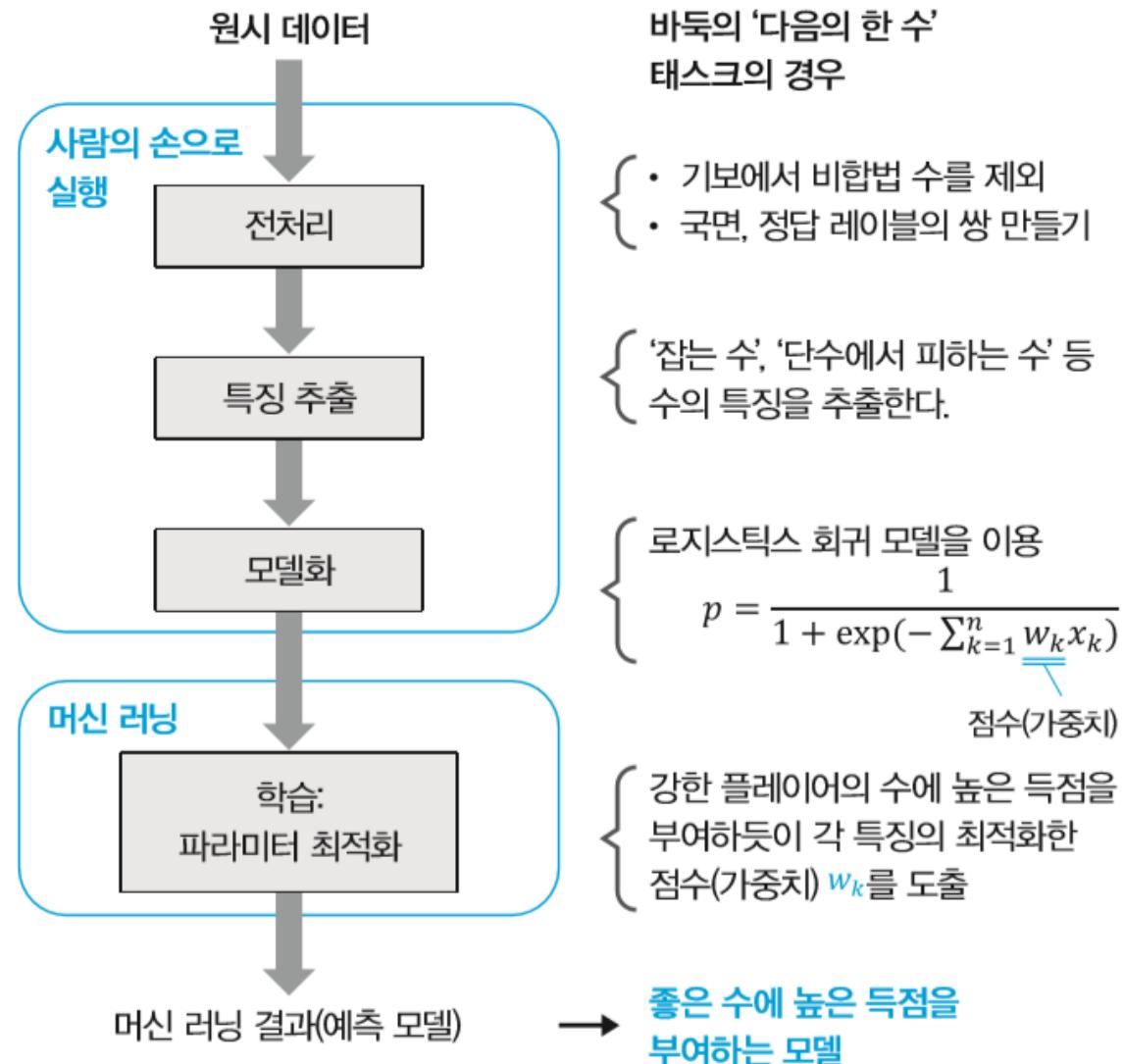
- 분류에 필요한 입력의 특징을 선택한다.

## • 모델화

- 특징에서 정답 레이블을 예측하는데 필요한 수학적 모델을 만든다.

## • 머신 러닝

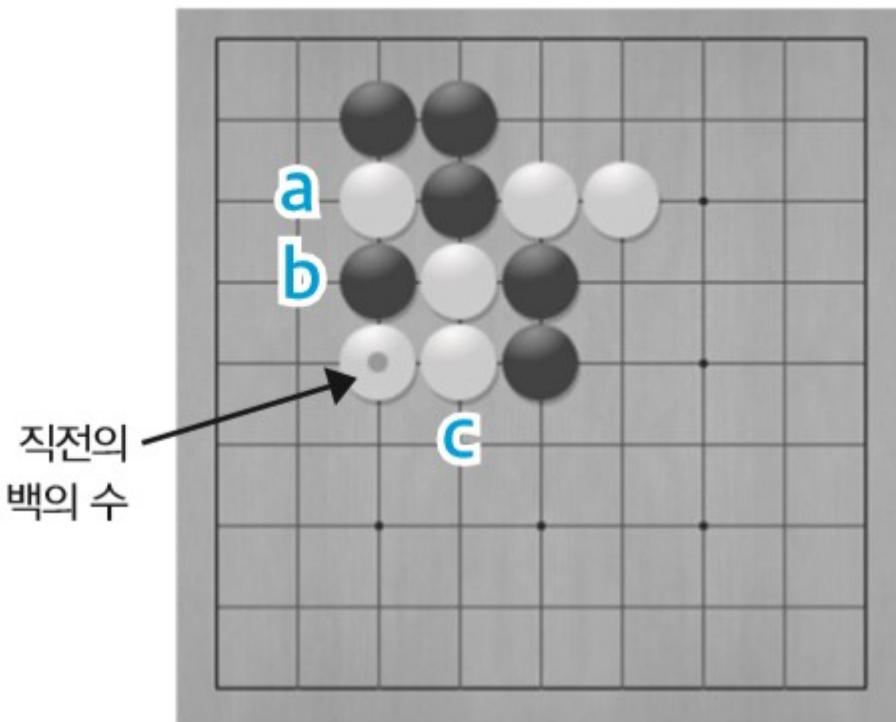
- 위의 수학적 모델의 정답률을 최대한 높이기 위해 튜닝한다.



# 특징의 예

- 각 특징의 점수(가중치)를 이용한 후보 수 a, b, c의 득점

(a) 현 국면(흑)과 후보 수 a, b, c



(b) 각 특징의 가중치 예와 각 후보 수의 득점 계산 예

특징(득점)	잡는 수(3점)	단수에서 피하는 수 (2점)	직전 수의 8 근방의 수 (1점)	합계 득점
수 a	3	2	0	5
수 b	0	2	1	3
수 c	0	0	1	1

# 롤 아웃 정책

- 알파고에서 수의 표면적인 특징을 바탕으로 그 수가 놓일 예측 확률을 출력하는 수학적 모델 중 하나
- 빠르게 평가할 수 있는 특징만을 이용하므로 평가 속도는 빠르지만, 수의 예측 확률은 다소 떨어져 24% 정도에 머문다.

# 룰 아웃 정책에 사용하는 특징과 학습 방법

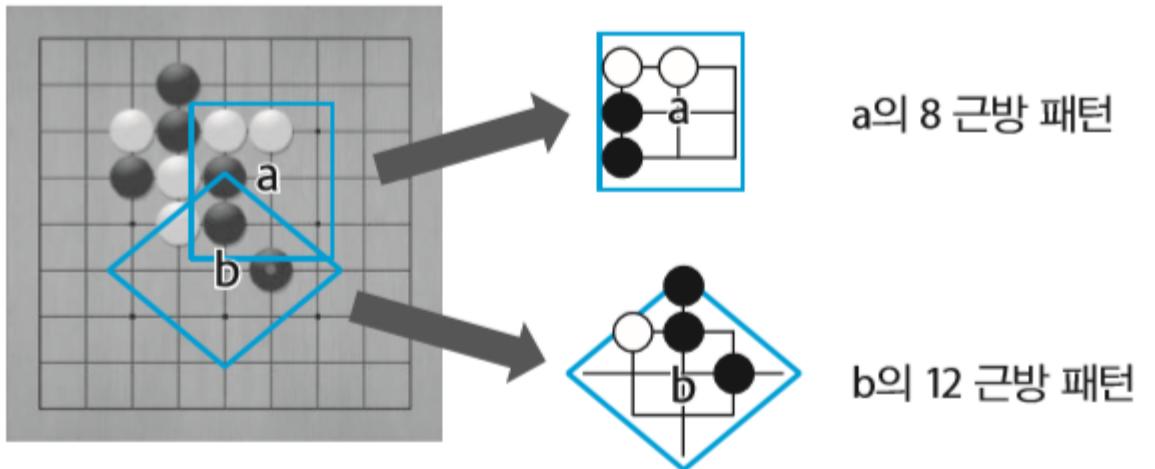
## (a) 바둑의 수의 특징

내용	종류
(1) 직전의 상대 수에 대응하는 수인가?	1
(2) 단수에서 피하는 수인가?	1
(3) 직전의 상대 돌의 8 근방에 두는 수인가?	8
(4) 직전의 상대에게 치중수 <sup>*1</sup> 의 형태로 놓여진 경우에 급소에 반격하는 수인가?	8,192
(5) 직전의 상대 돌의 12 근방에 둘 경우, 12 근방의 패턴이 특정 패턴과 일치하는가?	32,207
(6) 두고 싶은 위치의 8 근방 패턴이 특정 패턴과 일치하는가?	69,338
합계	109,747

\*1 상대의 지역 안에 돌을 둠으로써 상대에게 2집을 만들지 못하게 해 상대의 돌을 죽이는 것을 말한다.

# 롤 아웃 정책에 사용하는 특 징과 학습 방법

(b) 근방 패턴의 특징의 예

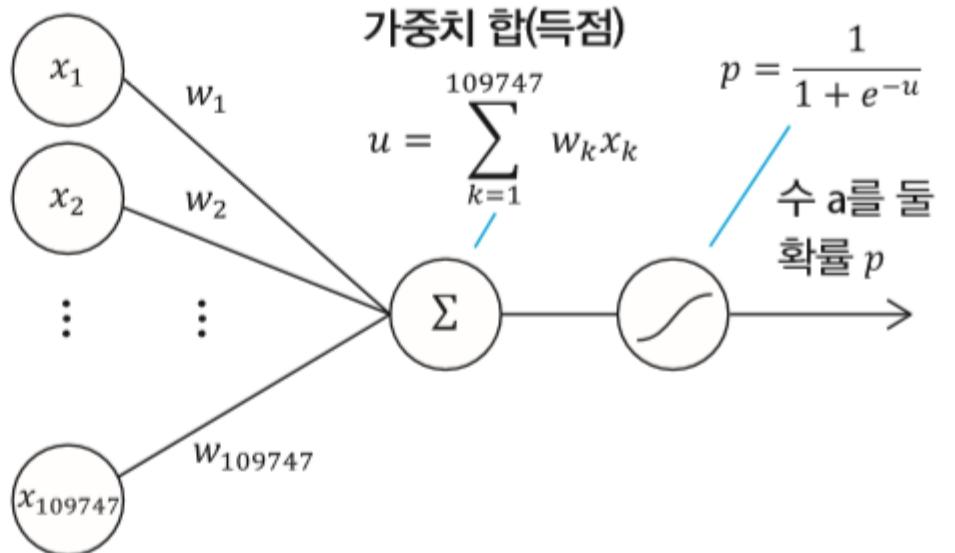


(c) 특정 수 a를 둘 확률을 도출하는 모델  
(로지스틱 회귀)

어떤 수 a가 상대의 수에  
대응하는 수인가?

어떤 수 a가 단수를  
피할 수 있는 수인가?

⋮



## 딥러닝-바둑 AI는 순간적으로 수를 떠올린다

알파고의 직관력을 지탱하는 것이 딥 러닝이다. 딥 러닝이란, 수학적 모델로 4층 이상의 신경망을 이용하는 머신 러닝법을 말한다. 여기에서는 '필기체 숫자 인식'을 사례로 들어 딥 러닝 방법의 하나인 컨볼루션 신경망을 설명한다.

또한, 알파고에서 두 개의 컨볼루션 신경망인 SL 정책 네트워크와 밸류 네트워크의 구조와 학습 방법에 대해 자세히 설명한다.

SL 정책 네트워크는 바둑의 국면을 마치 그림처럼 바라봄으로써 인간의 직관에 필적하는 '다음의 한 수' 태스크를 수행한다.

또한, 밸류 네트워크는 지금까지 불가능하다고 생각되었던 바둑의 평가 함수를 실현했다는 의미에서 알파고의 가장 큰 성과라고 할 수 있다.

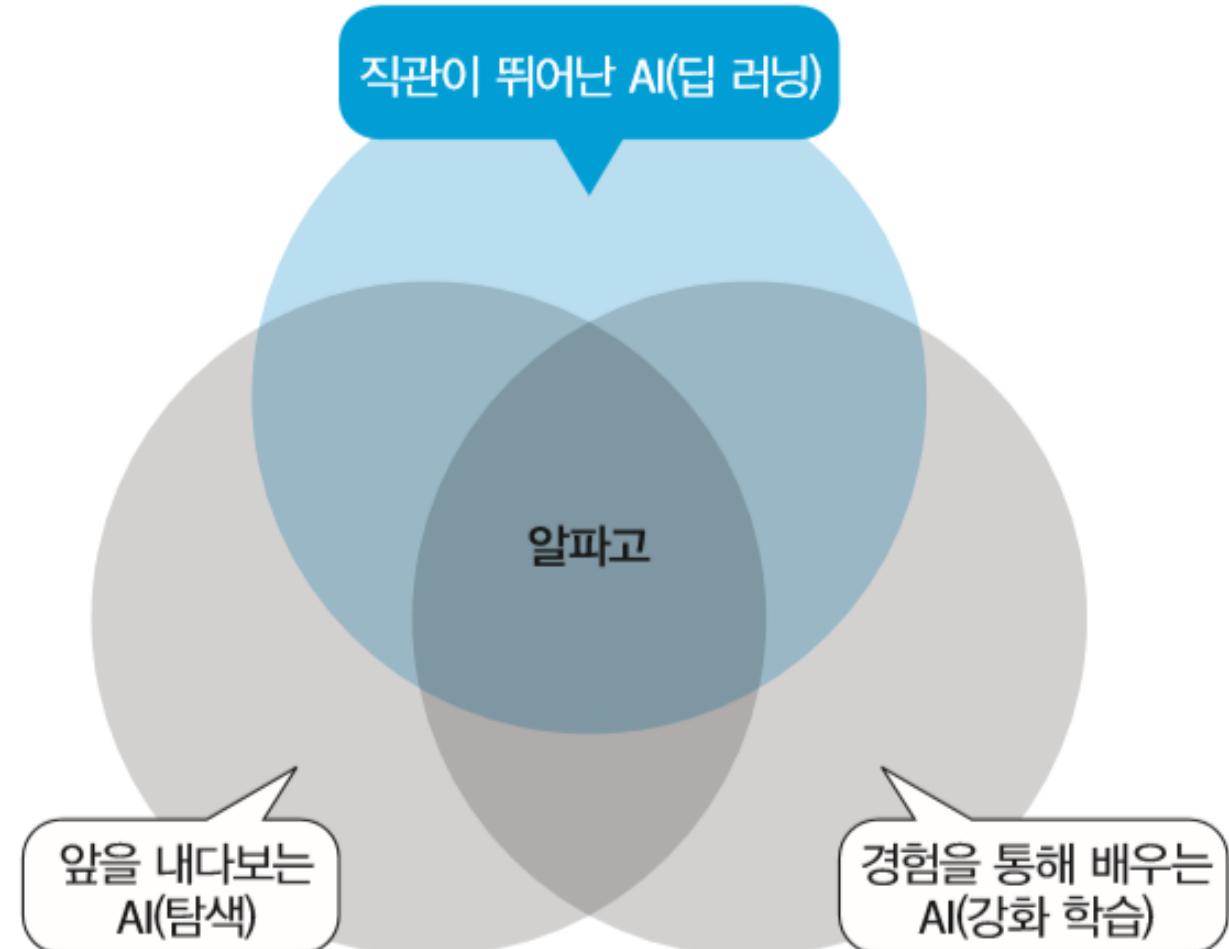
# 2. 직관에 뛰어난 AI: 딥 러닝

2.1 딥러닝이란?

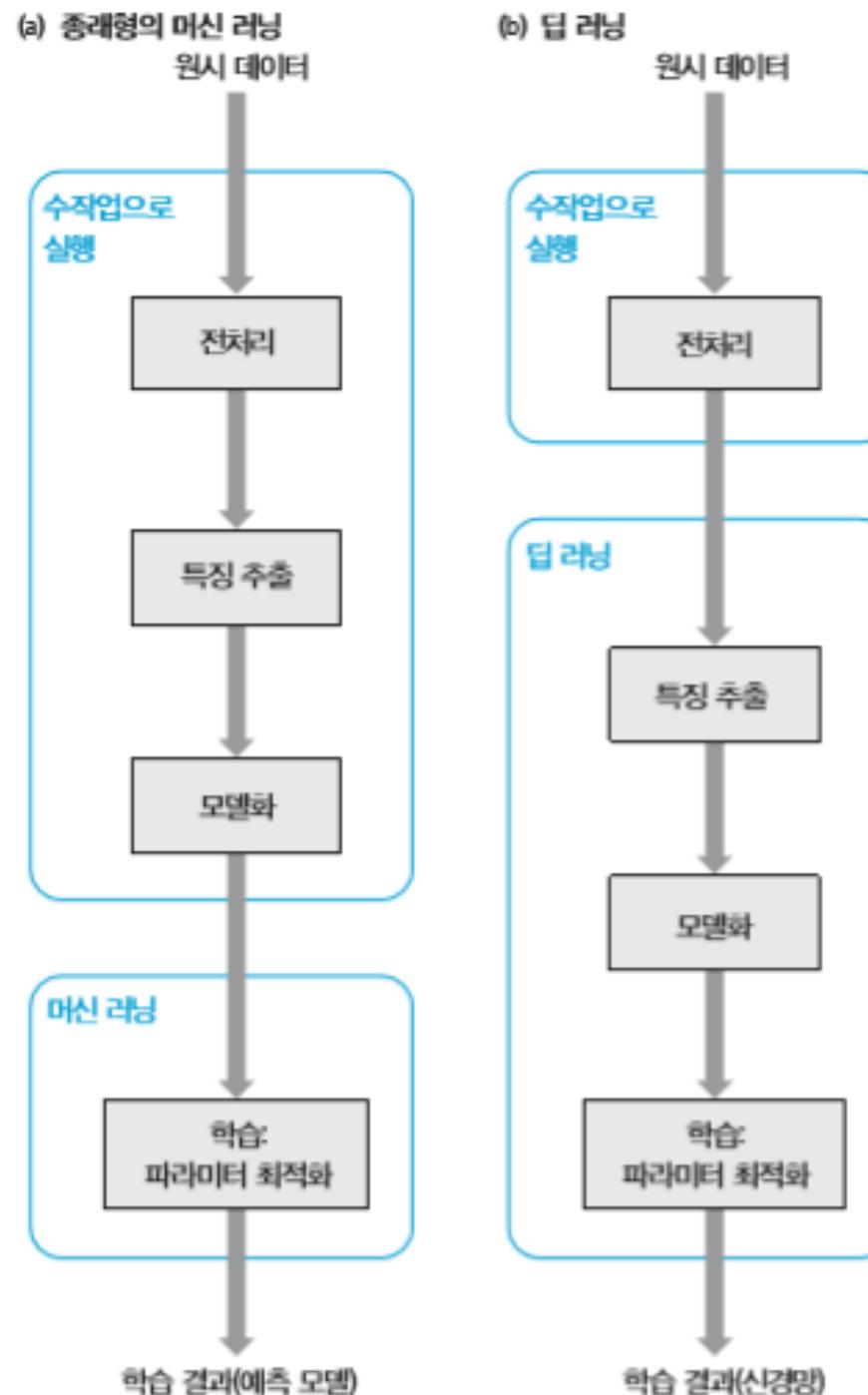
2.2 필기체 숫자 인식의 예

2.3 알파고의 컨볼루션 신경망

2.4 Chainer로 CNN 학습시키기

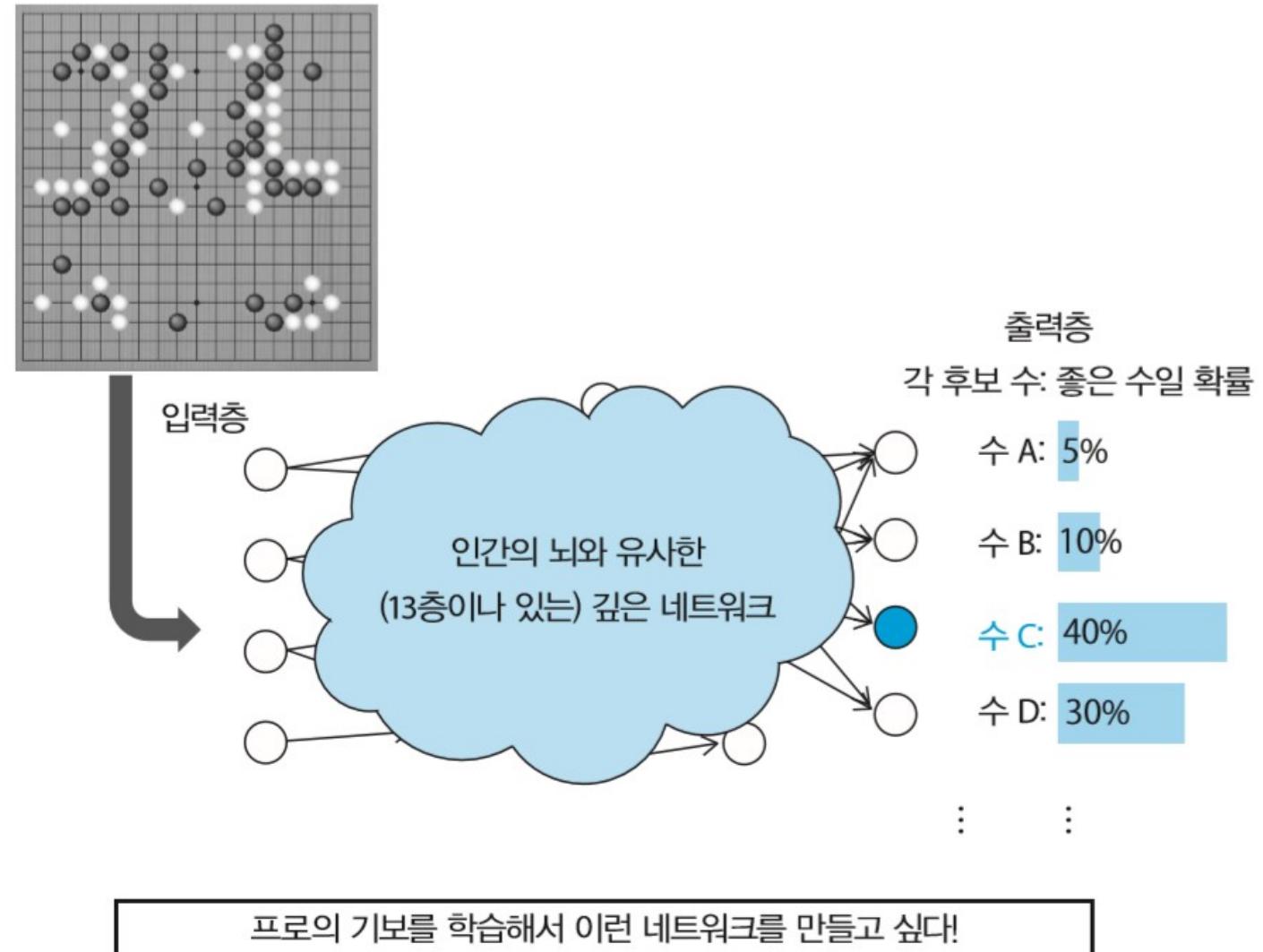


# 기존의 머신러닝 과 딥러닝과의 비교



# 알파고의 컨볼루션 신경망

- 바둑의 국면을 2 차원 이미지로 간주하여 입력하고, 각 후보 수가 좋은 수가 될 확률을 출력한다
- '다음의 한 수' 태스크에서 프로그리사와의 일치율 : 57%

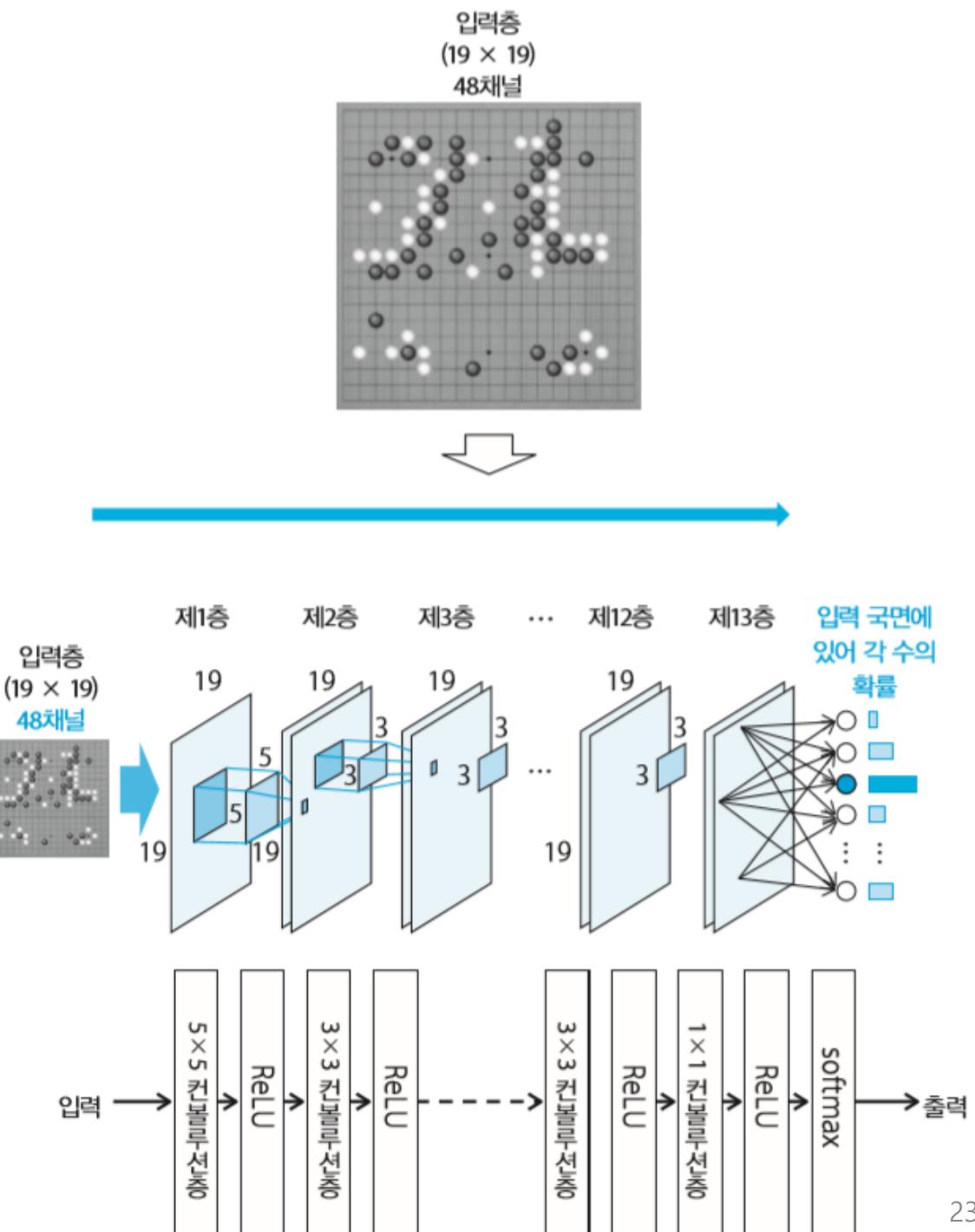


# 필기체 숫자 인식과 바둑 AI의 CNN과의 유사성

	이미지 인식(필기체 숫자 인식의 예)	바둑의 '다음의 한 수' 태스크
대상	필기체의 숫자	바둑판 위의 정보
기법	N층의 딥 러닝 (다양한 기법 존재)	13층의 CNN
입력층	<ul style="list-style-type: none"><li>• <math>28 \times 28</math>픽셀의 이미지</li><li>• 그레이스케일의 1채널</li><li>• 값은 0~255의 정수</li></ul>	<ul style="list-style-type: none"><li>• <math>19 \times 19</math>의 19줄 바둑판</li><li>• 48채널</li><li>• 값은 0~1 중 하나</li></ul>
출력	0~9 중 하나	19줄 바둑판의 361종류의 위치 중 하나
학습 데이터	사람이 필기한 대량의 숫자 (예: 우편번호)	강한 플레이어에 의한 바둑의 대량의 기보

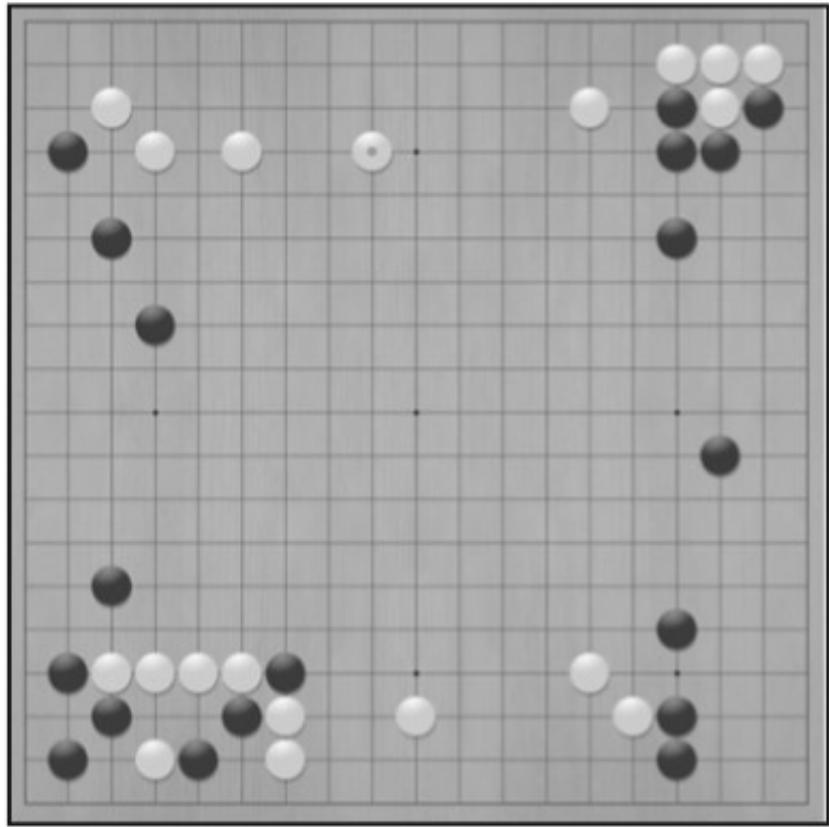
# 알파고의 SL 정책 네트워크

- 입력은 48채널(흑돌/백돌의 위치, 돌을 잡을 수 있는 위치, 축(단수와 피하는 수를 반복해서 돌을 취하는 변화), ...)
- 전체가 13층
- 1~12층의 필터는 각 층에 192 종류씩 있음
- 제1층은  $5 \times 5$ 의 필터 192 종류, 제2~12층은  $3 \times 3$ 의 필터 192종류로 구성된다.
- 제13층은  $1 \times 1$ 의 필터 1종류에 마지막으로 소프트맥스 함수를 통해 확률값을 반환한다

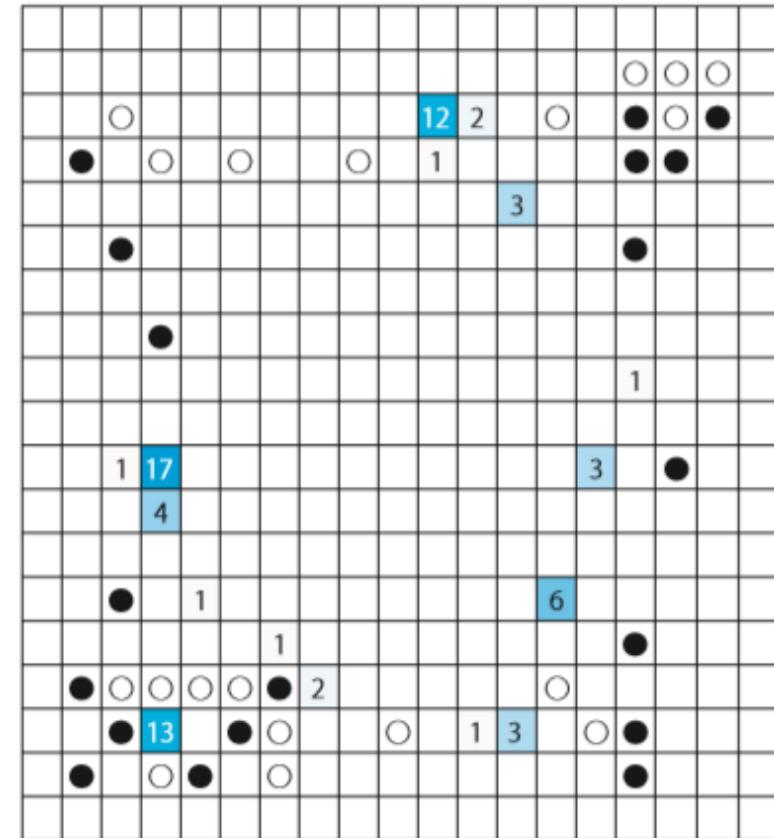


# SL 정책 네트워크의 출력 예

(a) 현재의 국면(39수째 흑의 차례)



## (b) SL 정책 네트워크의 출력



(b)에서 각 칸의 값은 소수점 이하를 반올림한 출력 확률(%)을 나타낸다. 빈칸은 모두 1% 미만이다

# SL 정책 네트워크의 48채널분의 입력

- 모든 채널은  $19 \times 19$ 의 19줄 바둑판에 대응한 0-1 데이터로 표현된다. 또한,  $k = 1 \sim 8$ 로 되어 있는 항목에 대해서는  $k = 1, 2, \dots, 7$  및 8 이상의 각각의 경우를 1채널로 나타내 총 8채널로 표현 한다

연: 돌의 묶음

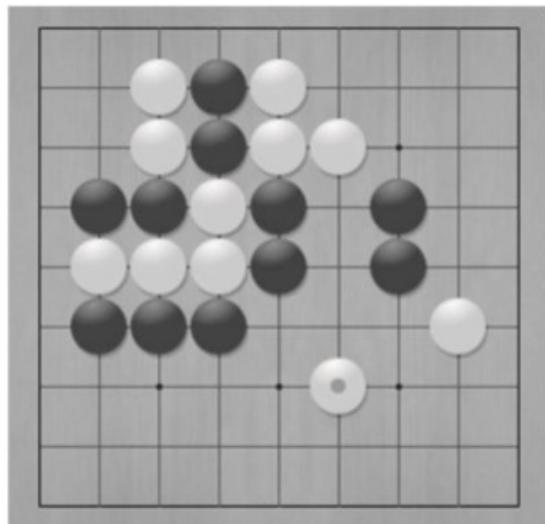
호흡점: 연이 앞으로 몇 수에서 잡힐지에 대한 값

입력 채널의 종류	채널 수
흑의 위치	1
백의 위치	1
빈 곳의 위치	1
$k$ 수 앞에 둔 위치( $k = 1 \sim 8$ )	8
돌이 있는 경우 해당 연의 호흡점 수( $k = 1 \sim 8$ )	8
거기에 둔 후, 돌을 취할 수 있는가?(취하는 수: $k = 1 \sim 8$ )	8
거기에 둔 후에 해당 연을 빼앗길 경우, 몇 개의 돌을 빼앗기는가?(돌의 수: $k = 1 \sim 8$ )	8
거기에 둔 후, 해당 연의 호흡점의 수(호흡점의 수: $k = 1 \sim 8$ )	8
거기에 둔 후, 인접하는 상대의 연을 축으로 취할 수 있는가?	1
거기에 놓인 후, 인접하는 아군의 연을 축으로 빼앗기는가?	1
합법 수인가?	1
모두 1로 채운다.	1
모두 0으로 채운다.	1
합계	48

# SL 정책 네트워크의 입력 특징량의 예

- 여기에서는 (a)의  $9 \times 9$  바둑판 위에서의 6가지 특징을 나타내었다. (b)(c)(d)의 위치 정보는  $9 \times 9$ 의 0-1 데이터로 표현된다. (e)(f)(g)에 대해서는 원래  $9 \times 9$ 의 0-1 데이터로 구성된 8채널로 나타내지만, 지면을 절약하기 위해 각각 1~8의 숫자를 기입한  $9 \times 9$ 의 파란색 음으로 표시하고 있다

(a) 현재의 국면  
(흑의 차례)



(b) 흑의 위치

0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

(c) 백의 위치

0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

(d) 빈 자리의 위치

1	1	1	1	1	1	1	1	1	1
1	1	0	0	0	1	1	1	1	1
1	1	0	0	0	0	1	1	1	1
1	0	0	0	0	1	0	1	1	1
1	0	0	0	0	1	0	1	1	1
1	0	0	0	1	1	1	0	1	1
1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

(e) n수 앞의 위치

0	0	0	0	0	0	0	0	0	0
0	0	0	0	5	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	4	0	0	0
0	7	0	0	0	0	2	0	0	0
0	0	8	6	0	0	0	3	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

(f) 연 호흡점의 수

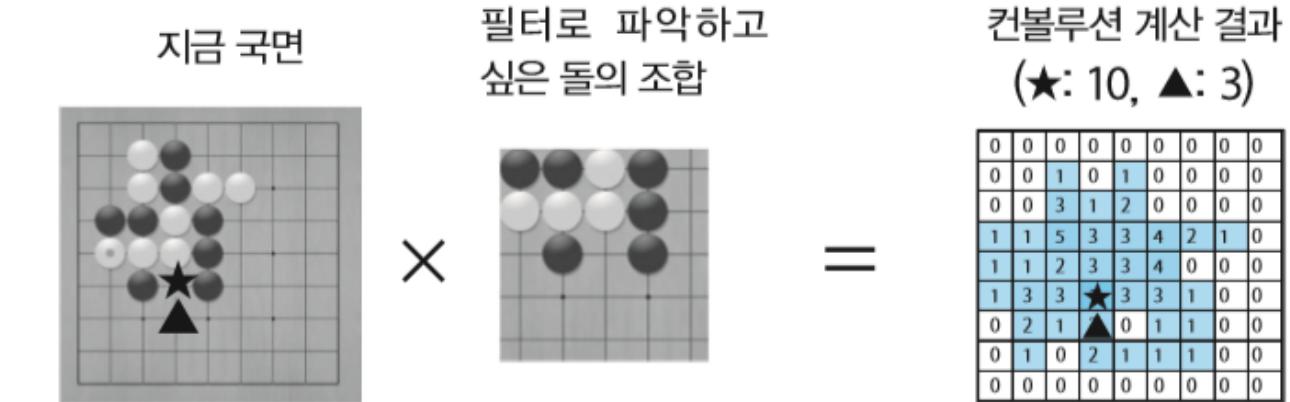
0	0	0	0	0	0	0	0	0	0
0	0	3	1	4	0	0	0	0	0
0	0	3	1	4	4	0	0	0	0
0	2	2	2	3	0	6	0	0	0
0	2	2	2	3	0	6	0	0	0
0	5	5	5	0	0	0	4	0	0
0	0	0	0	0	4	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

(g) 잡을 수 있는 돌의 수

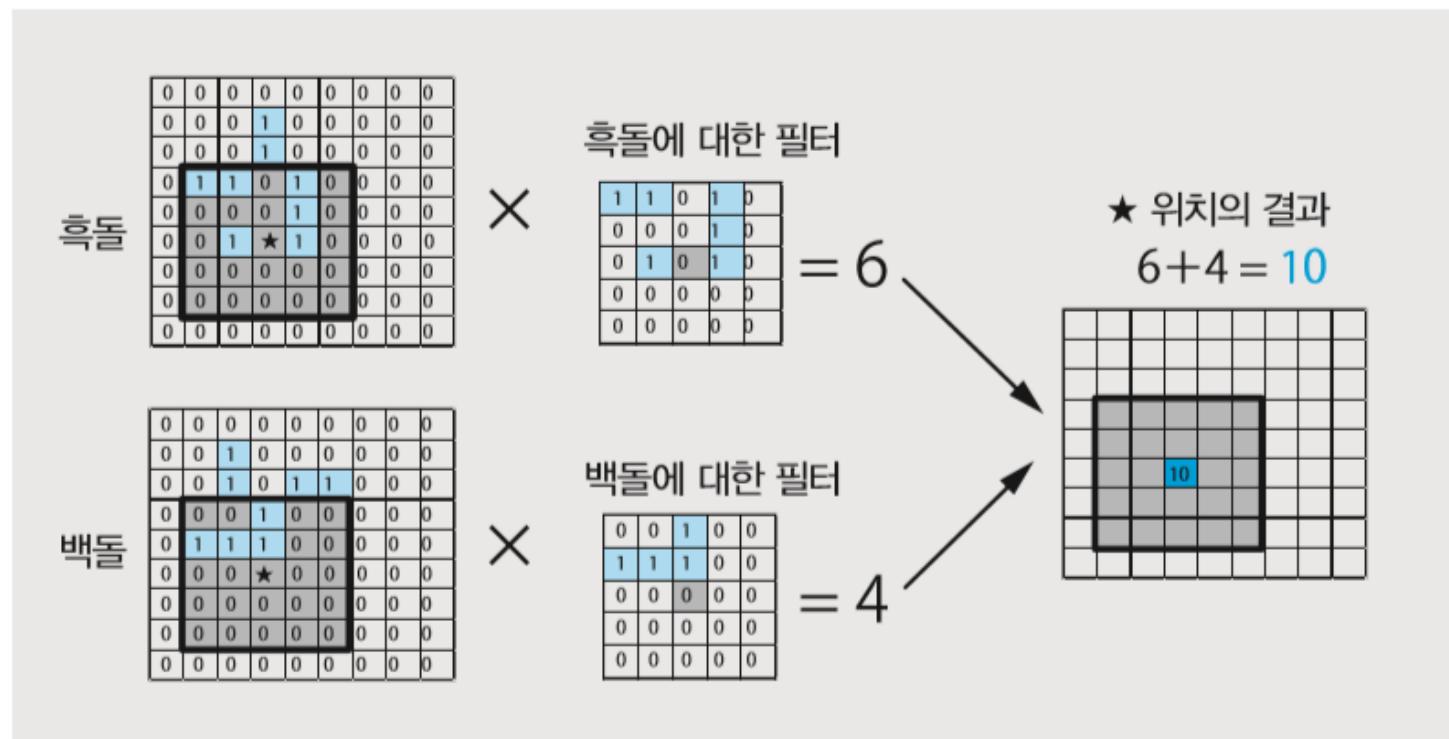
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

# SL 정책 네트워크에 있어서 제1층의 컨볼 루션 계산 예

(a) 흑을 두면 백을 잡을 수 있는 위치

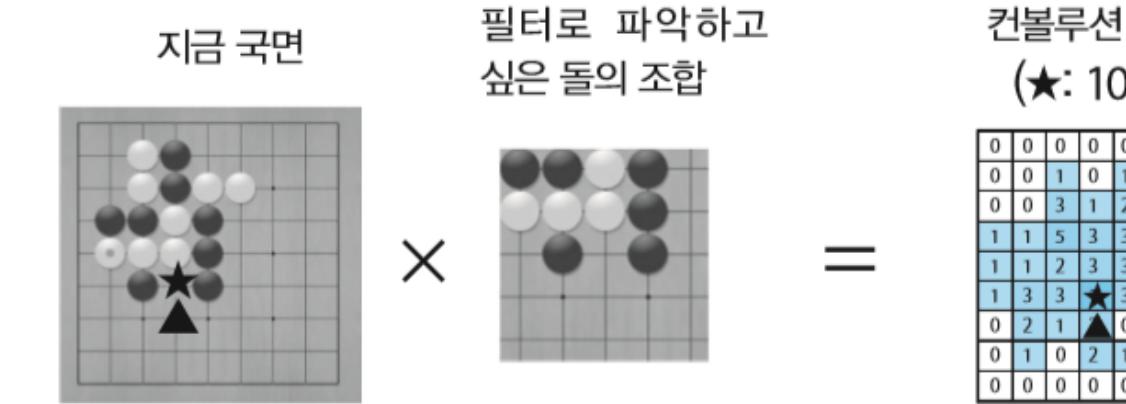


(b) ★ 위치의 컨볼루션 계산의 예



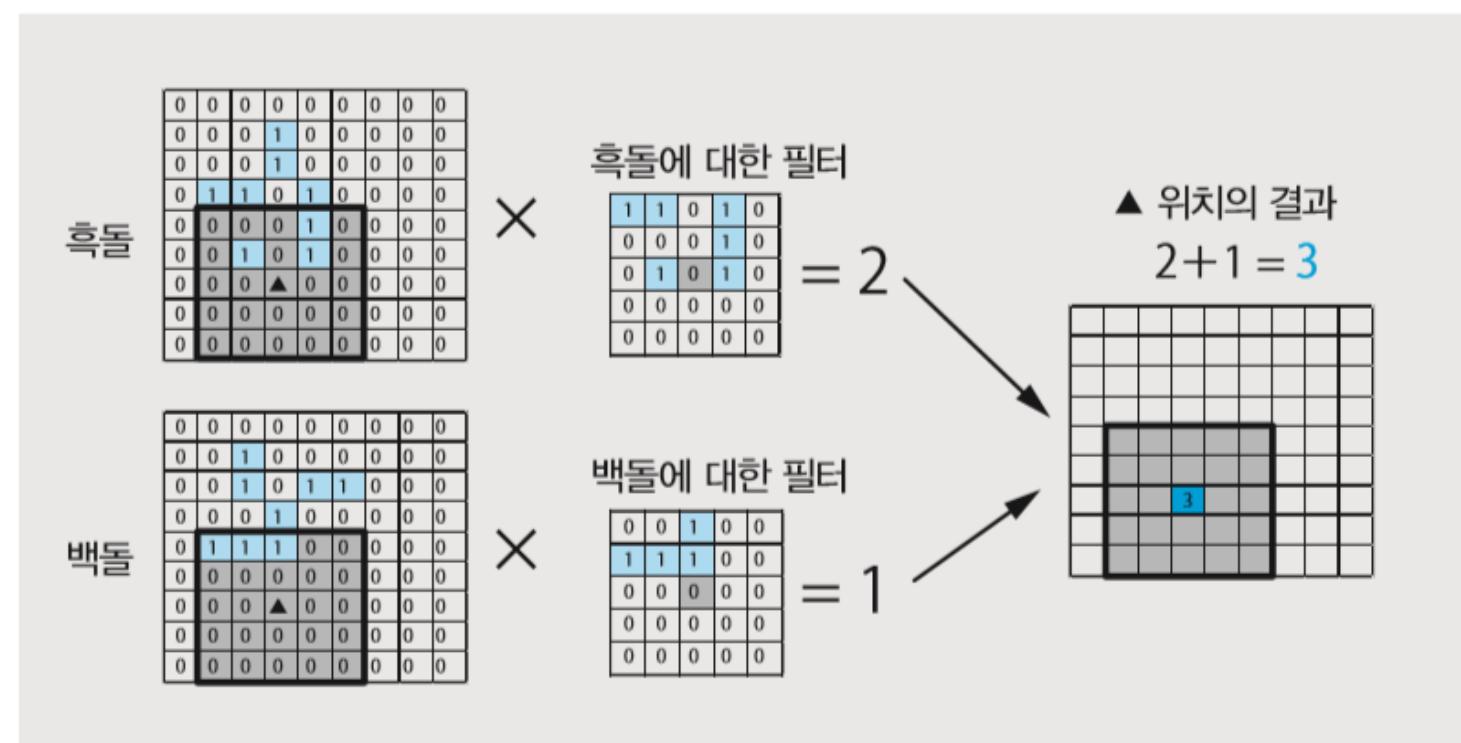
# SL 정책 네트워크에 있어서 제1층의 컨볼 루션 계산 예

(a) 흑을 두면 백을 잡을 수 있는 위치



컨볼루션 계산 결과  
(★: 10, ▲: 3)

(c) ▲ 위치의 컨볼루션 계산의 예



# SL 정책 네트워크의 컨볼루션 계산 및 계산량

특정 층의 입력( $x_{ij}$ )  
( $19 \times 19 \times 192$ 장)

컨볼루션 결과( $u_{ij}$ )  
( $19 \times 19$ )

특정 층의 출력( $y_{ij}$ )  
( $19 \times 19$ )

$$\text{컨볼루션 계산: } u_{ij} = \sum_{k=1}^{192} \sum_{p=1}^3 \sum_{q=1}^3 w_{pqk} \cdot x_{i+p,j+q,k} + b$$

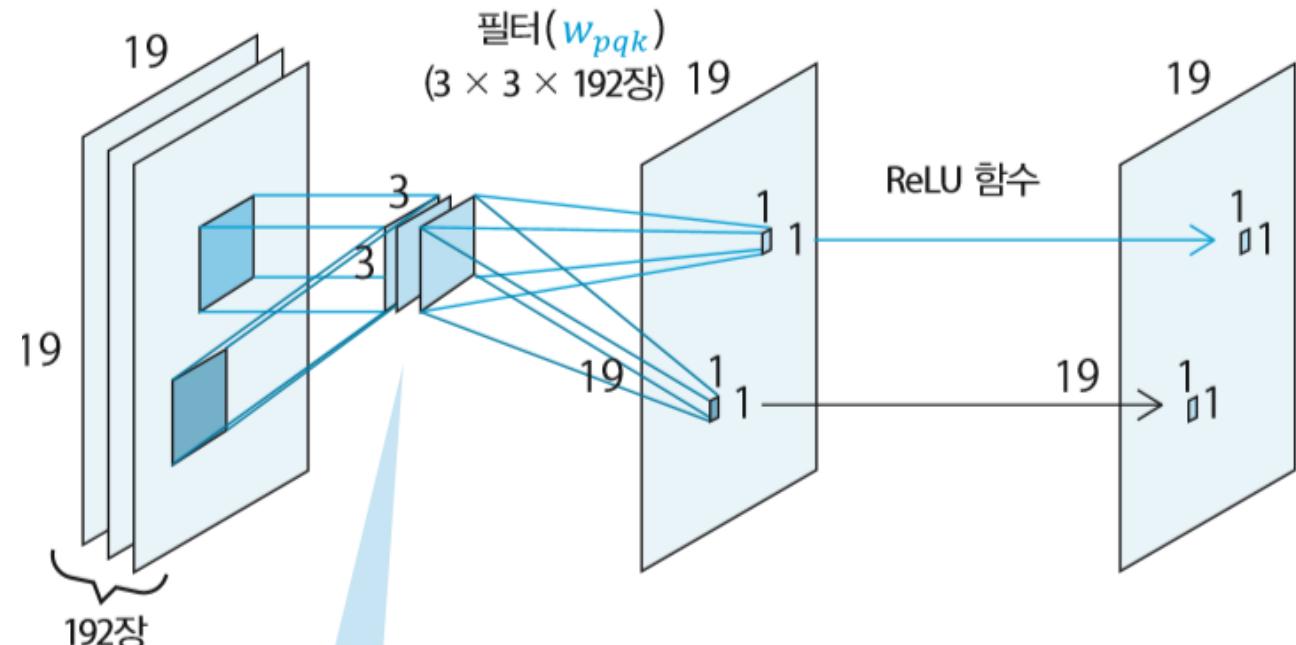
$$\text{ReLU 함수: } y_{ij} = \text{MAX}(0, u_{ij})$$

파라미터  $w_{pqk}$  의 개수:  $3^2 \times 192^2 \times 12$

$$32 \times (\text{필터의 종류: } 192)^2 \times (\text{층의 수: } 12) = \text{약 } 400\text{만 개}$$

컨볼루션 계산의 덧셈 횟수:  $19^2 \times 3^2 \times 192^2 \times 12$

$$192 \times 3^2(\text{필터의 종류: } 192)^2 \times (\text{층의 수: } 12) = \text{약 } 14\text{억 회}$$

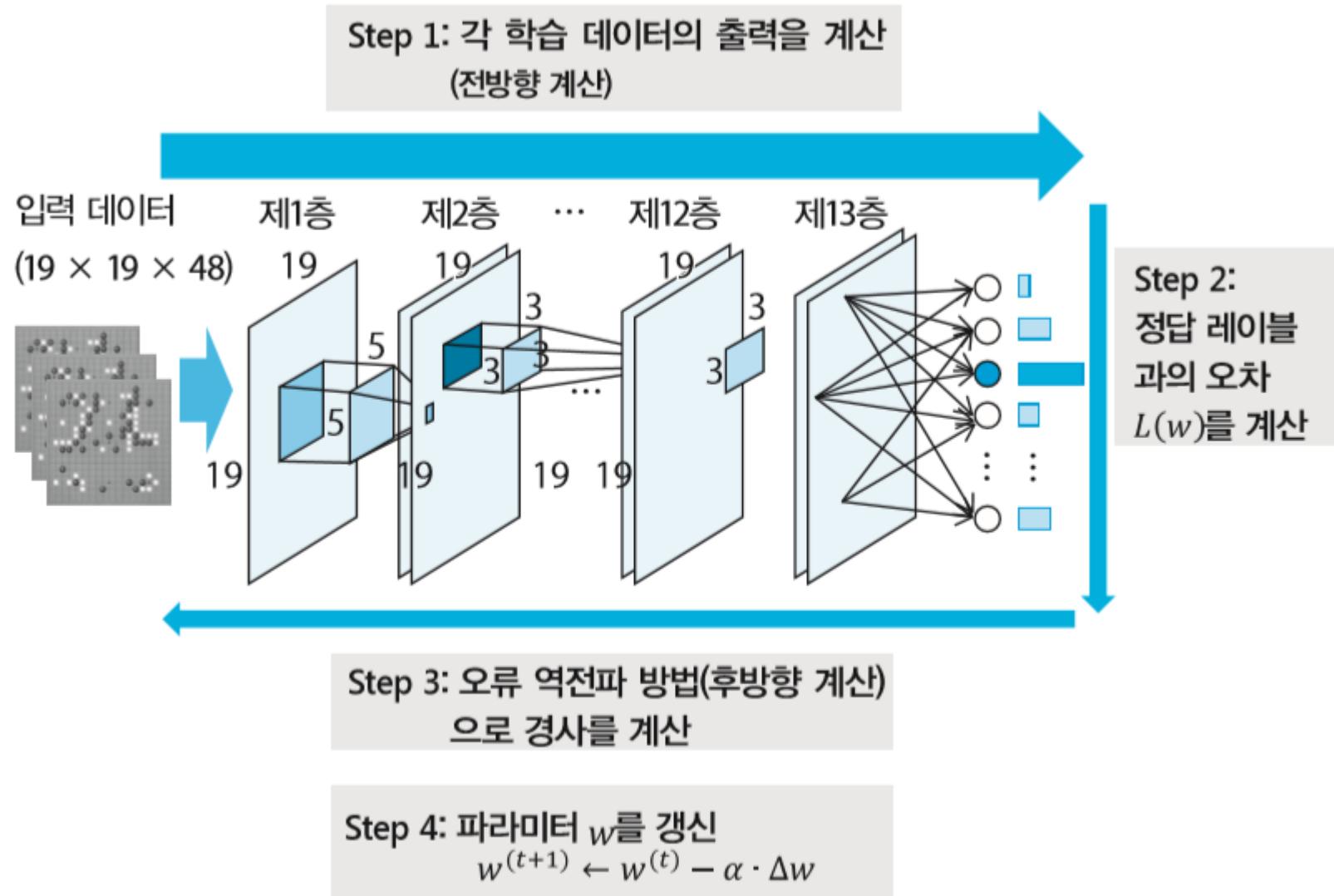


사실은 이 필터 자체가  
192종류다.

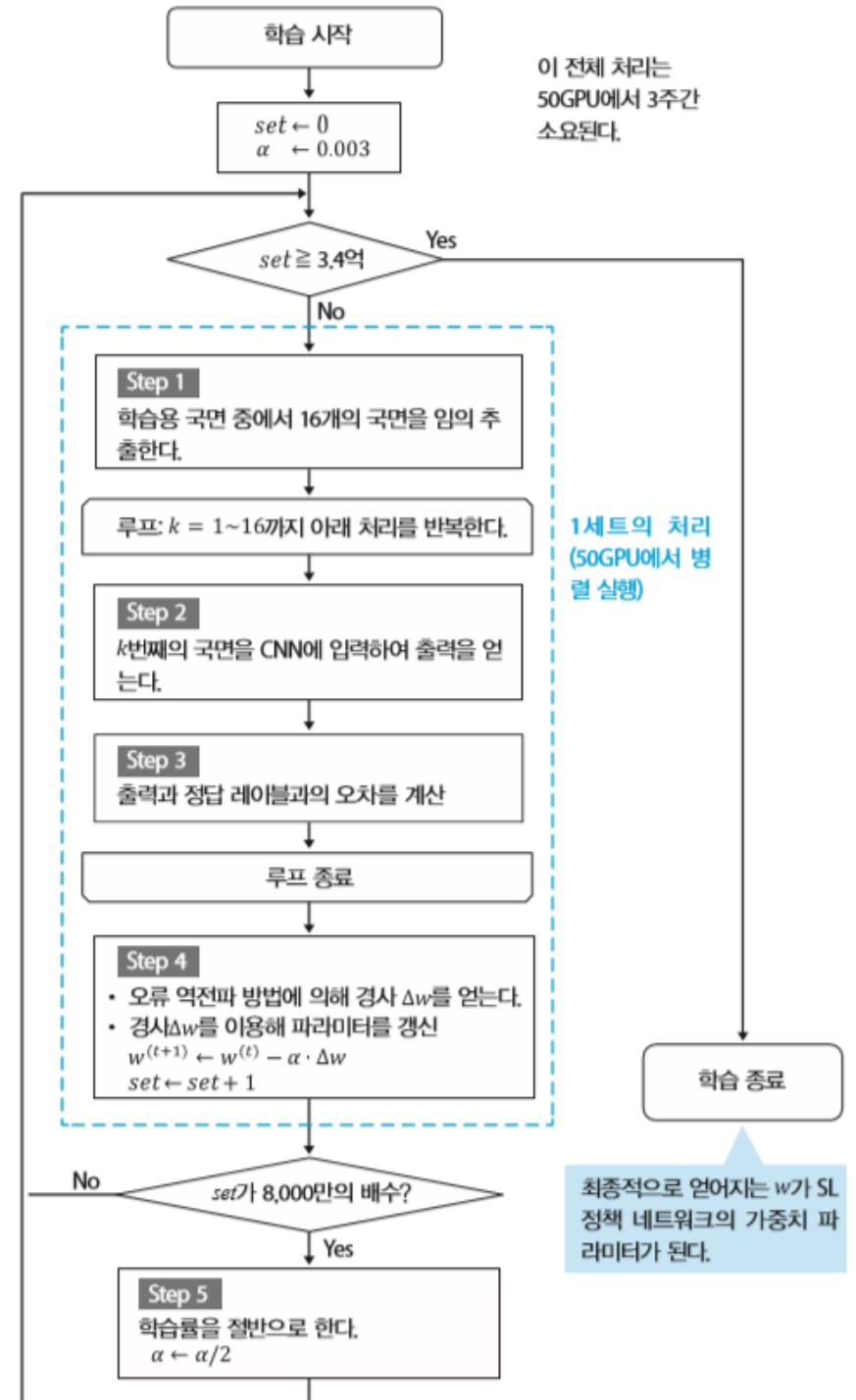
※ 편의상 제1층도 제2~12층과 같은 구조라고 가정하였으며,  
계산량과 파라미터가 적은 제13층은 무시하였다.

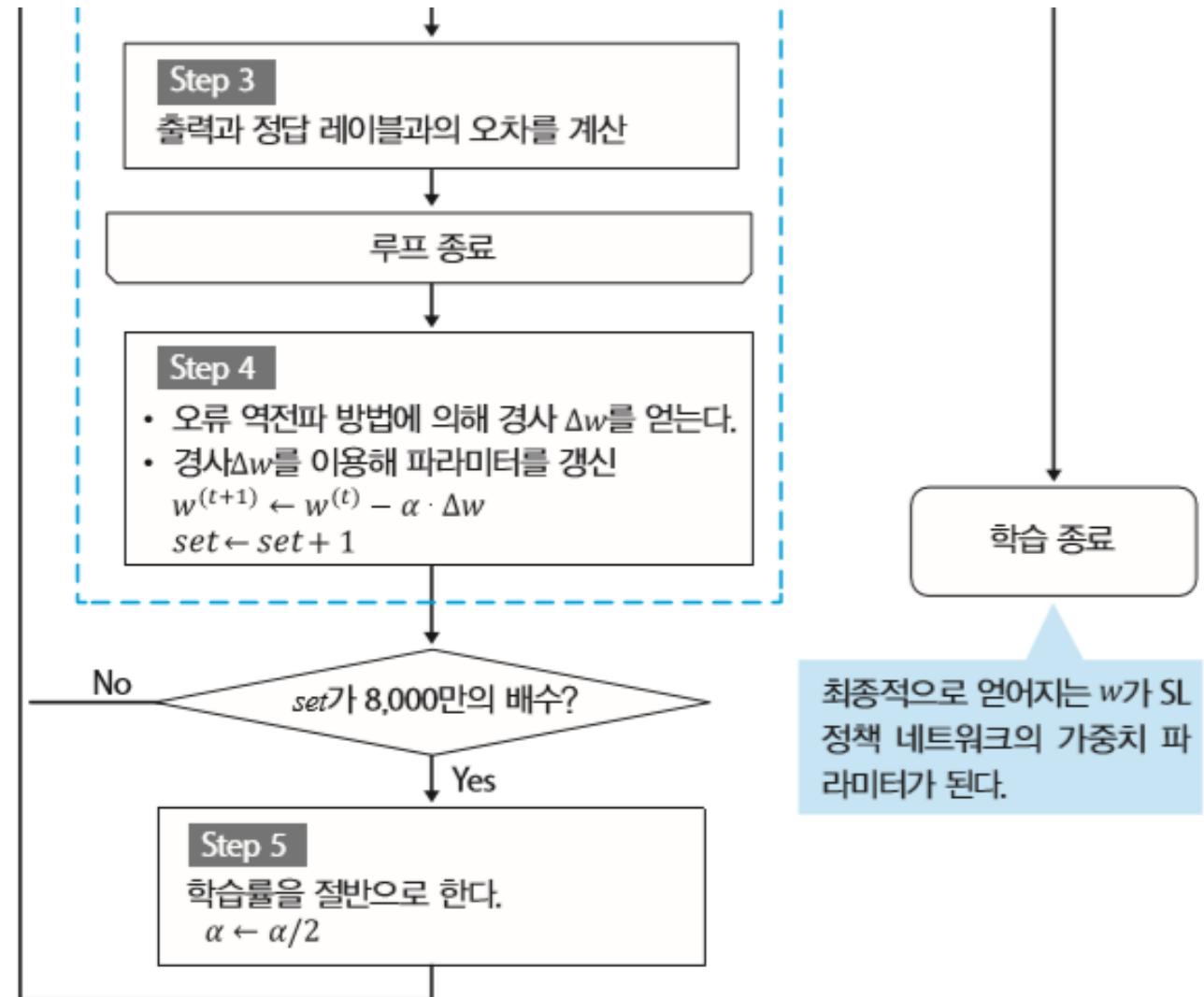
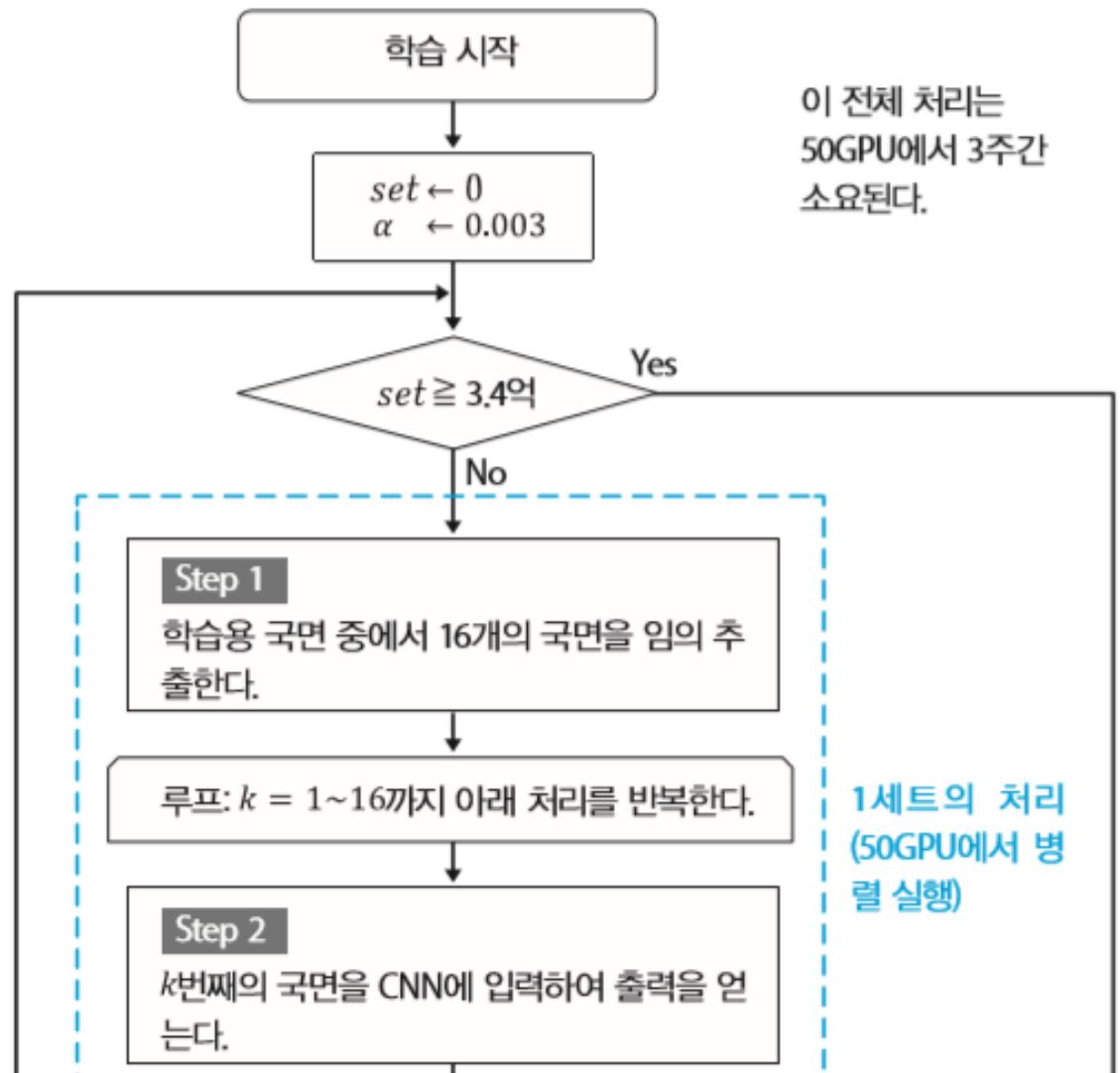
# SL 정책 네트워크의 학습

- 학습 데이터: (국면, 강한 인간 플레이어의 수)의 쌍
  - 인터넷 바둑 도장 'KGS' 6~9 단의 기보 16만 국(약 3,000 만 국면)
- 오류 역전파 방법을 통해 CNN의 출력과 인간의 수가 최대한 일치하는 필터 가중치를 구한다.
- 1회의 컨볼루션 계산은 GPU(이미지 처리용 프로세서)라면 약 5밀리초이지만, 그래도 학습에는 50GPU에서 3주 정도 소요된다.



# SL 정책 네트워크의 학습 플로차트



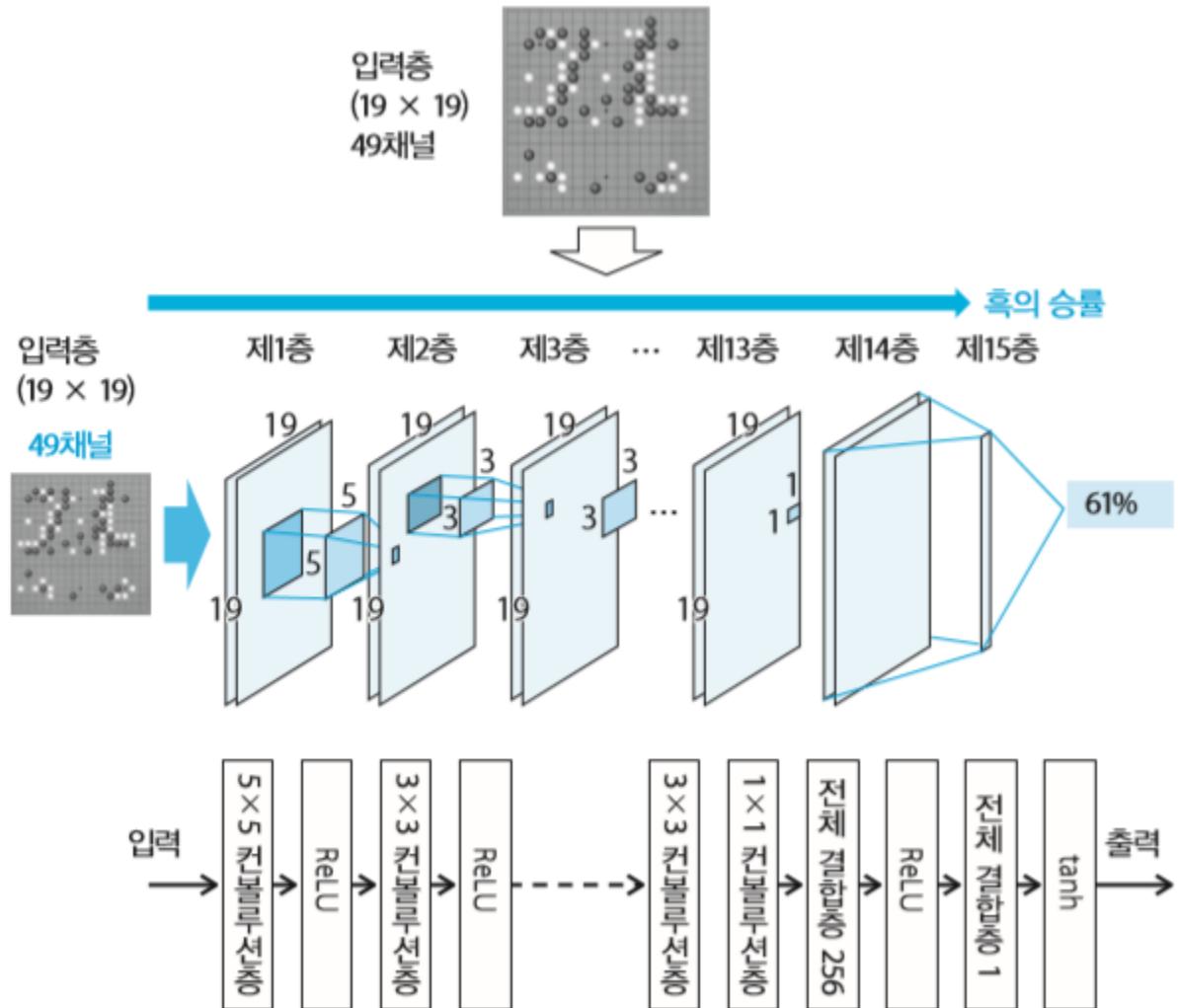


# SL 정책 네트워크의 학습 결과

- 테스트 데이터
  - KGS 기보 약 3,000만 국면 중 100만 개를 테스트 데이터로 사용
- 학습 결과
  - 각 층의 필터를 192장으로 한 경우 (판후이 버전) : 55.7%의 일치율
  - 각 층의 필터를 256장으로 한 경우 (이세돌 버전) : 57%의 일치율

# 승률 예측값을 계산하는 밸류 네트워크

- 밸류 네트워크는 국면을 입력으로 하고, 승률 예측값을 출력하는 CNN
- 입력은 49채널(SL 정책 네트워크와 같은 48채널 + 두는 쪽 정보)
- 전부 15층
- 1~12층의 필터는 각 층에 192종류씩 있고, 1층째만  $5 \times 5$ , 2~12층은  $3 \times 3$
- 정책 네트워크의 구조와 비교하면 입력이 한 채널 늘어난 점과 출력이 승률 1개인 점이 다르다. 제1~13층의 구조는 거의 동일하다
- 최종적으로 tanh 함수를 통과시켜 -1.0~1.0의 승률 예측치를 얻음
  - 1에 가까울 수록 두는 쪽 승률 예측값이 크고, -1에 가까울수록 상대방의 승률 예측값이 크다.



# 강화 학습-바둑 AI는 경험을 배운다

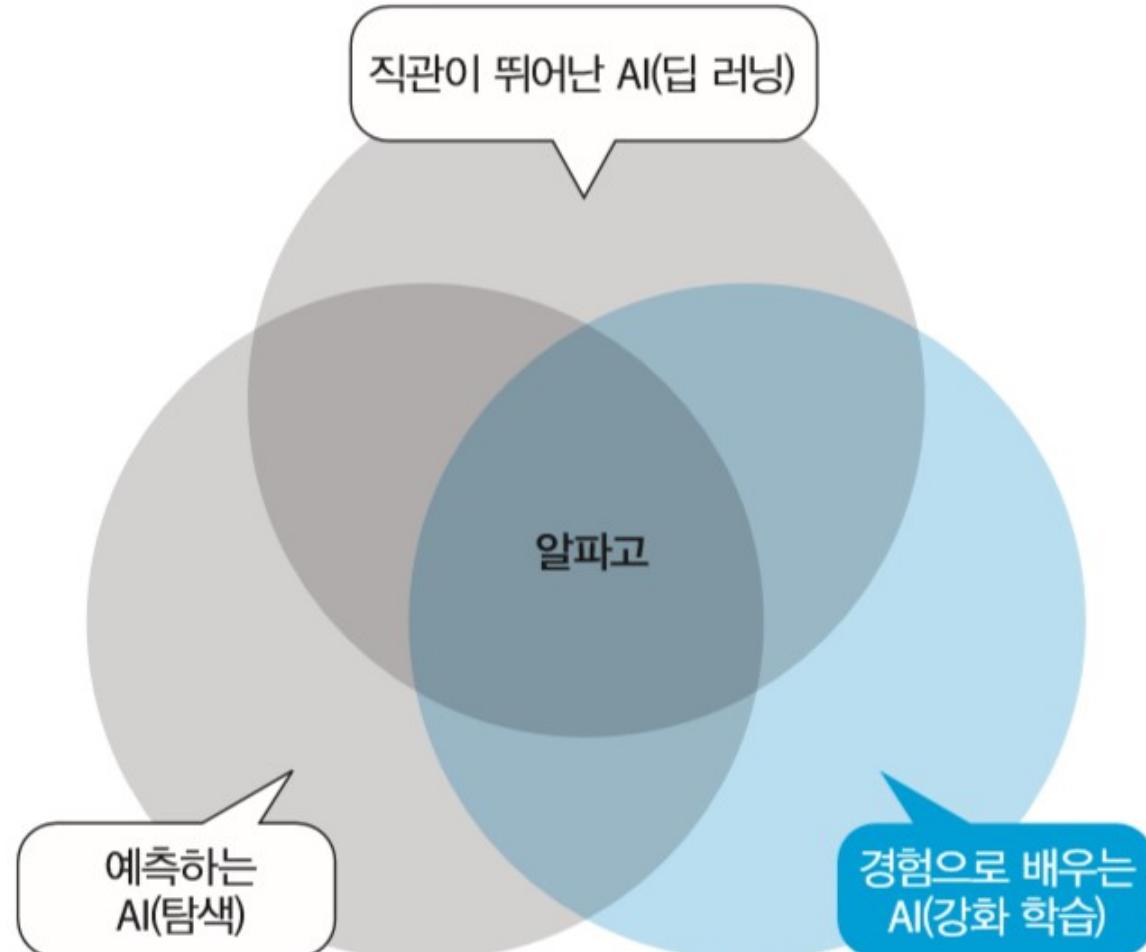
알파고는 경험을 배워 더욱더 강해진다. AI가 성공 경험을 바탕으로 행동을 개선해 나가는 기법을 강화 학습이라고 한다.

먼저, 강화 학습의 기본적인 구조를 설명하기 위해 멀티 앰드 밴딧 문제(multi-armed bandit problem)와 미로의 사례를 소개한다. 특히, 미로의 사례에서는 Q 학습과 정책 경사법(policy gradients method)이라는 최근 많이 사용되는 두 개의 강화 학습 방법을 설명한다. 또한, 비디오 게임에서 처음부터 플레이를 반복함으로써 인간 전문가 수준의 기술을 습득할 수 있는 DQN이라 불리는 기술을 소개한다.

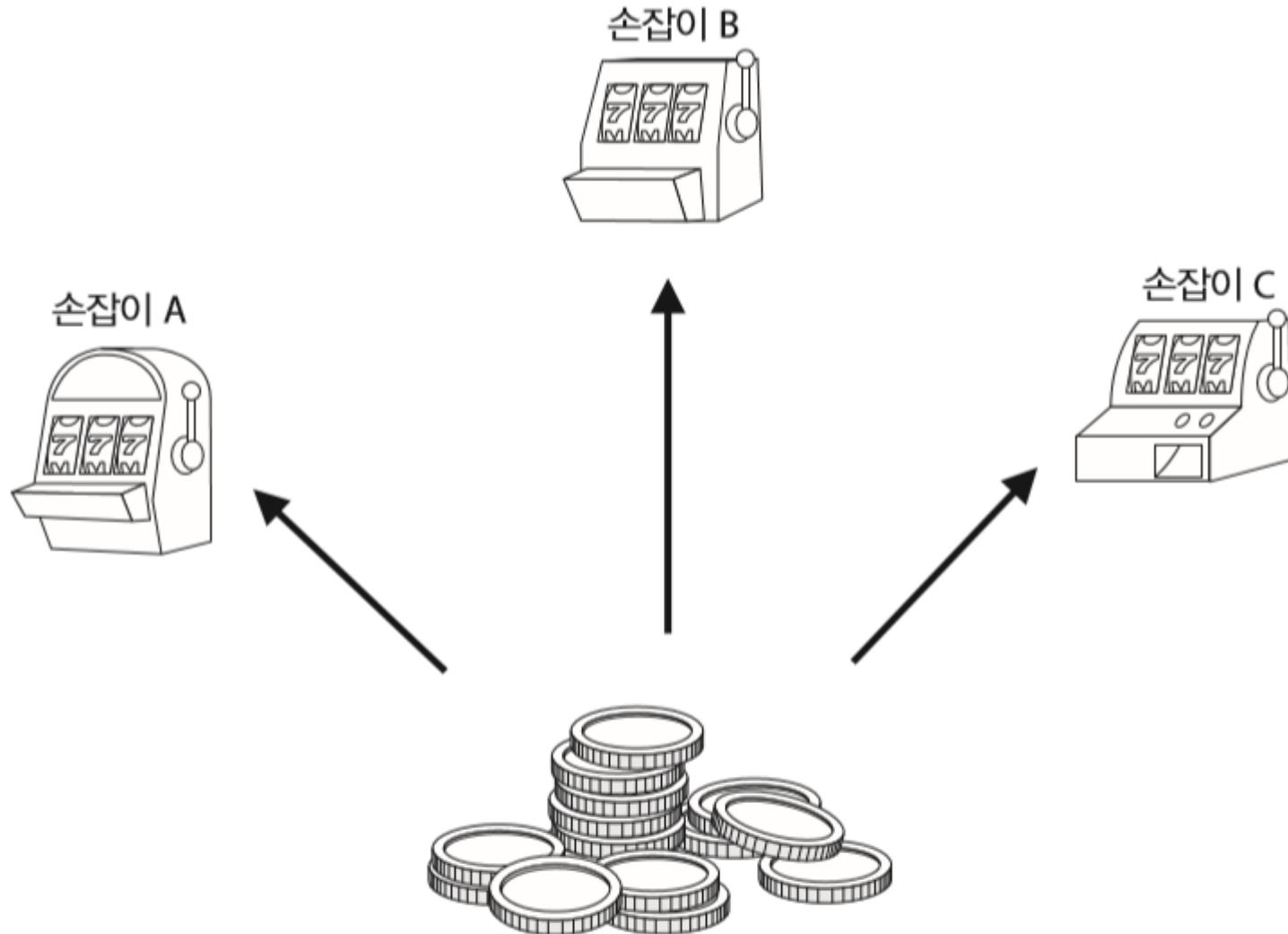
마지막으로, 앞 장에서 설명한 SL 정책 네트워크끼리 자기 대전시켜 더 강한 정책 네트워크를 획득하는 알파고의 강화 학습 방법에 대해 설명한다.

# 3. 경험에서 배우는 AI : 강화 학습

- 3.1 강화 학습이란?
- 3.2 강화 학습의 역사
- 3.3 멀티 암드 밴딧 문제
- 3.4 미로를 풀기 위한 강화 학습
- 3.5 비디오 게임 조작 스킬을 얻기  
위한 강화 학습
- 3.6 알파고의 강화 학습



# Multi-Armed Bandits Problem

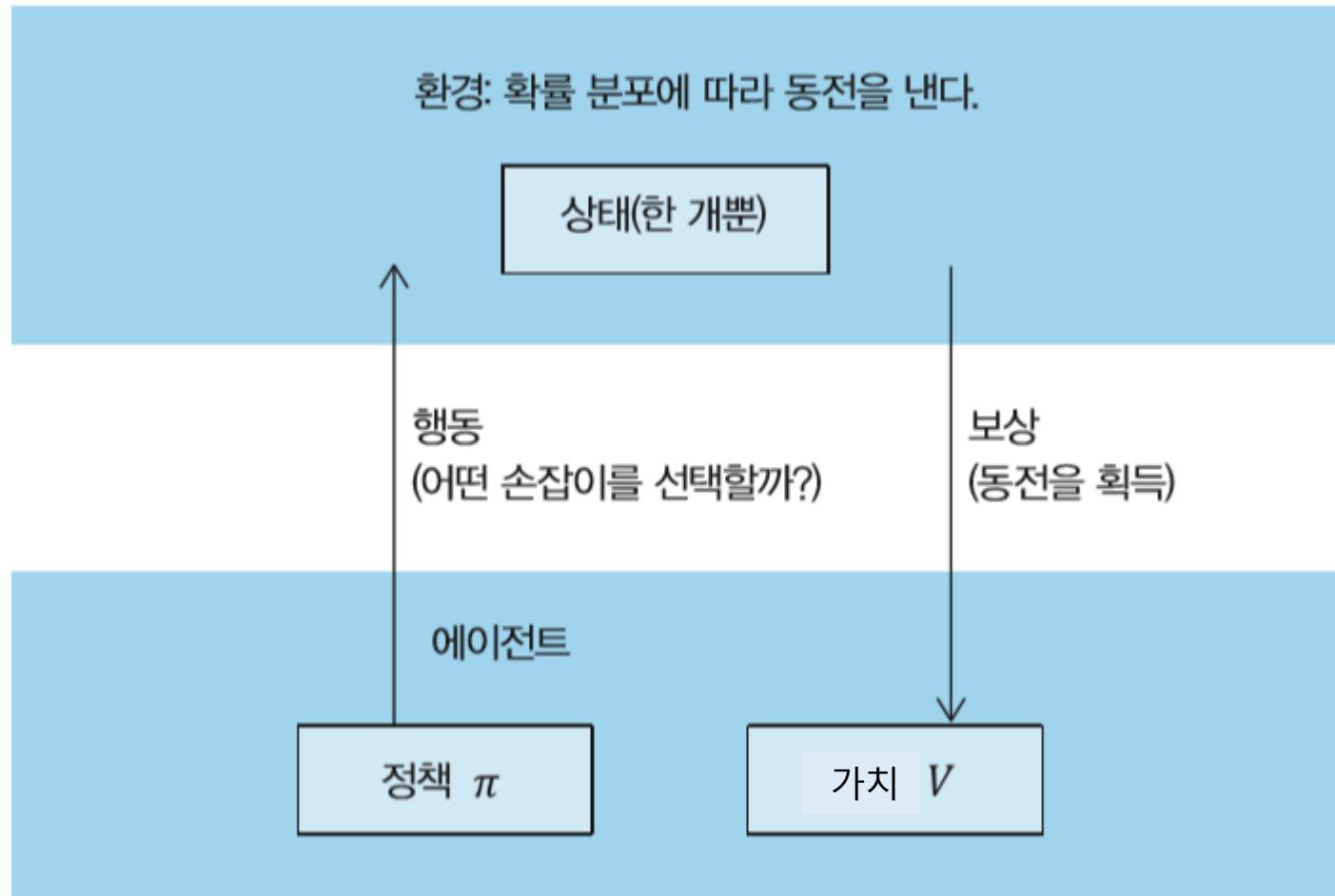


※ 탐사와 활용의 트레이드 오프  
(exploration-exploitation tradeoff)를 다루는 문제

어떤 손잡이(슬롯 머신)에 동전을 넣으면  
가장 많은 코인을 얻을 수 있을까?

# Multi-Armed Bandits Problem

- 얻을 수 있는 코인의 수를 극대화하는 것을 목적으로 어떤 손잡이를 선택할지에 대한 정책을 학습한다





- 멀티 암드 밴딧 문제에 대해 (a) 성공 확률이 높은 손잡이를 계속 선택하는 간단한 정책의 경우와, (b) UCB1이 높은 손잡이를 선택하는 UCB 정책의 시도 결과. 두 표 모두 왼쪽에는 각 회의 선택과 그 결과, 오른쪽에는 손잡이 A•B 각각에 대한 중간 회까지의 통산 성적을 나타낸다. (a)에서는 성공 확률만을 다음 선택의 지침으로 하므로 1차에 우연히 손잡이 A에서 실패 한 다음에는 손잡이 B를 계속 선택한다. 한편, (b)에서는 성공률에 바이어스를 가한 UCB1을 다음 선택의 지침으로 하므로 6번째 이후는 손잡이 A를 올바르게 선택하게 된다

=>

- 탐사와 활용의 트레이드 오프 (exploration-exploitation tradeoff)

(a) 성공 확률이 높은 손잡이를 선택하는 정책의 경우

시도 횟수	선택	성공/실패	손잡이 A 선택의 결과		손잡이 B 선택의 결과		다음 선택
			성공 횟수/시도 횟수	성공 확률	성공 횟수/시도 횟수	성공 확률	
1회째	손잡이 A	X	0/1	0%	0/0	-	→ B
2회째	손잡이 B	O	0/1	0%	1/1	100%	→ B
3회째	손잡이 B	X	0/1	0%	1/2	50%	→ B
4회째	손잡이 B	O	0/1	0%	2/3	67%	→ B
5회째	손잡이 B	X	0/1	0%	2/4	50%	→ B
6회째	손잡이 B	O	0/1	0%	3/5	60%	→ B
7회째	손잡이 B	X	0/1	0%	3/6	50%	→ B
8회째	손잡이 B	O	0/1	0%	4/7	57%	→ B
9회째	손잡이 B	X	0/1	0%	4/8	50%	→ B
10회째	손잡이 B	O					

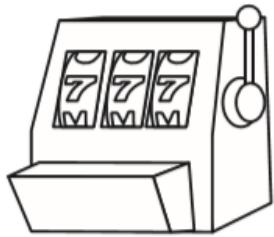
→ 처음에 선택한 손잡이 A의 실패가 발단이 되어, 손잡이 B를 잘못해서 계속 선택한다.

(b) UCB1이 높은 손잡이를 선택하는 정책의 경우

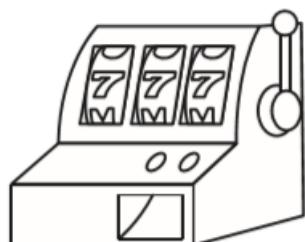
시도 횟수	선택	성공/실패	손잡이 A 선택의 결과			손잡이 B 선택의 결과			다음 선택
			성공 횟수/시도 횟수	성공 확률	ucb1	성공 횟수/시도 횟수	성공 확률	ucb1	
1회째	손잡이 A	X	0/1	0%	0.00	0/0	-	1.41	→ B
2회째	손잡이 B	O	0/1	0%	0.78	1/1	100%	1.78	→ B
3회째	손잡이 B	X	0/1	0%	0.98	1/2	50%	1.09	→ B
4회째	손잡이 B	O	0/1	0%	1.10	2/3	67%	1.17	→ B
5회째	손잡이 B	X	0/1	0%	1.18	2/4	50%	0.94	→ A
6회째	손잡이 A	O	1/2	50%	1.48	2/4	50%	1.09	→ A
7회째	손잡이 A	O	2/3	67%	1.52	2/4	50%	1.20	→ A
8회째	손잡이 A	X	2/4	50%	1.28	2/4	50%	1.28	→ A
9회째	손잡이 A	O	3/5	60%	1.31	2/4	50%	1.34	→ A
10회째	손잡이 A	O							

→ 처음에 선택한 손잡이 A에서 실패해도, 나중에 올바른 손잡이 B를 선택한다.

손잡이 A: 진정한 성공률 80%



손잡이 B: 진정한 성공률 50%



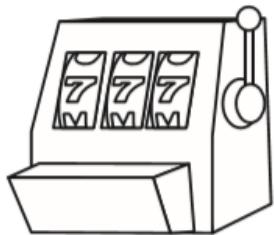
(a) 성공 확률이 높은 손잡이를 선택하는 정책의 경우

시도 횟수	선택	성공/ 실패	손잡이 A 선택의 결과		손잡이 B 선택의 결과		다음 선택
			성공 횟수/ 시도 횟수	성공 확률	성공 횟수/ 시도 횟수	성공 확률	
1회째	손잡이 A	X	0/1	0%	0/0	-	→ B
2회째	손잡이 B	O	0/1	0%	1/1	100%	→ B
3회째	손잡이 B	X	0/1	0%	1/2	50%	→ B
4회째	손잡이 B	O	0/1	0%	2/3	67%	→ B
5회째	손잡이 B	X	0/1	0%	2/4	50%	→ B
6회째	손잡이 B	O	0/1	0%	3/5	60%	→ B
7회째	손잡이 B	X	0/1	0%	3/6	50%	→ B
8회째	손잡이 B	O	0/1	0%	4/7	57%	→ B
9회째	손잡이 B	X	0/1	0%	4/8	50%	→ B
10회째	손잡이 B	O					

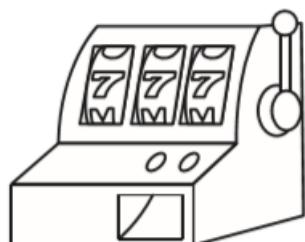
처음에 선택한 손잡이 A의 실패가 발단이 되어, 손잡이 B를 잘못 해서 계속 선택한다.

(b) UCB1이 높은 손잡이를 선택하는 정책의 경우

손잡이 A: 진정한 성공률 80%



손잡이 B: 진정한 성공률 50%



시도 횟수	선택	성공/실패	손잡이 A 선택의 결과			손잡이 B 선택의 결과			다음 선택
			성공 횟수/시도 횟수	성공 확률	ucb1	성공 횟수/시도 횟수	성공 확률	ucb1	
1회째	손잡이 A	X	0/1	0%	0.00	0/0	-	1.41	→ B
2회째	손잡이 B	O	0/1	0%	0.78	1/1	100%	1.78	→ B
3회째	손잡이 B	X	0/1	0%	0.98	1/2	50%	1.09	→ B
4회째	손잡이 B	O	0/1	0%	1.10	2/3	67%	1.17	→ B
5회째	손잡이 B	X	0/1	0%	1.18	2/4	50%	0.94	→ A
6회째	손잡이 A	O	1/2	50%	1.48	2/4	50%	1.09	→ A
7회째	손잡이 A	O	2/3	67%	1.52	2/4	50%	1.20	→ A
8회째	손잡이 A	X	2/4	50%	1.28	2/4	50%	1.28	→ A
9회째	손잡이 A	O	3/5	60%	1.31	2/4	50%	1.34	→ A
10회째	손잡이 A	O							

처음에 선택한 손잡이 A에서 실패해도, 나중에 올바른 손잡이 A를 선택한다.

# UCB 정책의 선택 기준(UCB1)

UCB (Upper Confidence Bound)

- UCB 정책의 선택 기준: UCB1이 최대의 손잡이를 선택

성공률: 이 손잡이의  
성공률

바이어스: 신뢰 구간으로 이  
손잡이의 시도 횟수가 적은 경  
우에 커진다.

$$\text{UCB1} = (w/n) + (2 \log t/n)^{1/2}$$

바이어스란 '우연에 의한  
성공률 편차의 크기'를  
나타낸다

$n$ : 이 손잡이의 시도 횟수

$w$ : 이 손잡이의 성공 횟수

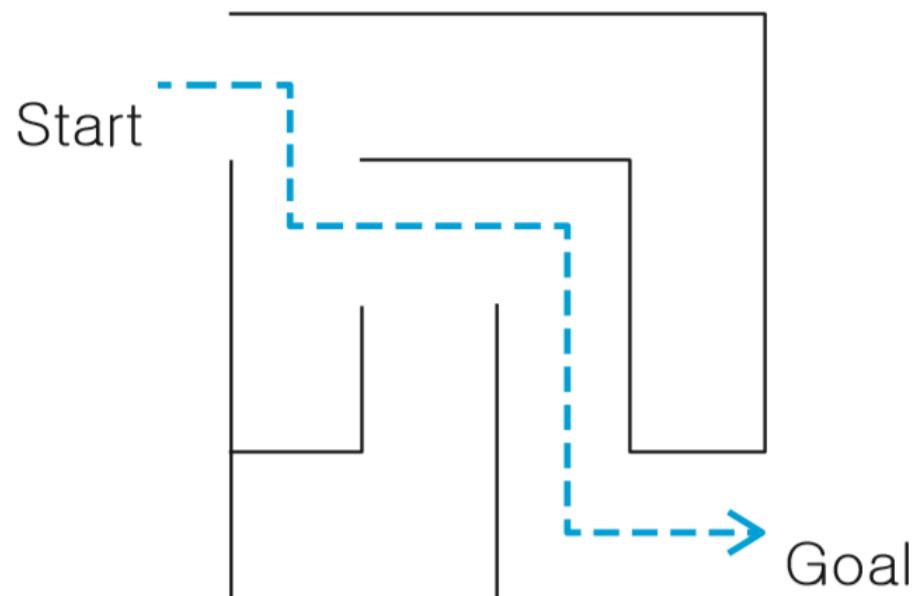
$t$ : 총 시도 횟수

- UCB 정책을 사용하는 경우 리그레트가 최소가 된다.

리그레트 = 신의 선택의 보상 - 어떤 정책의 보상 기댓값

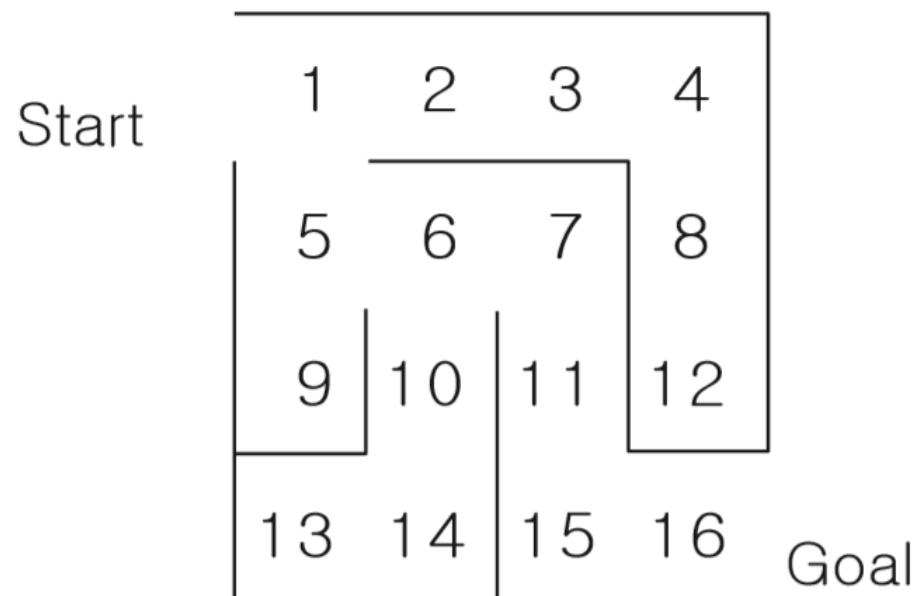
# 미로 풀기

(a) 미로의 예



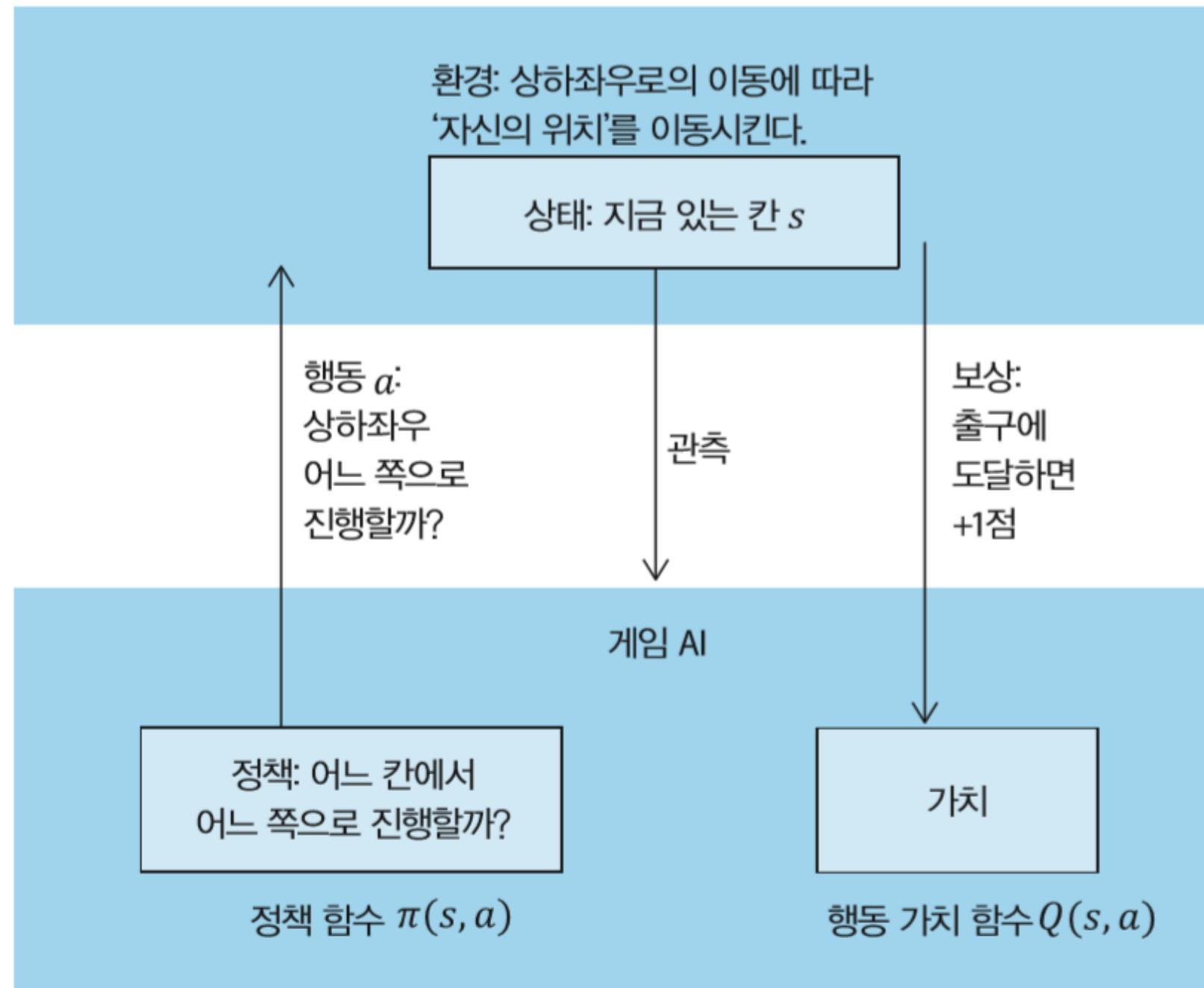
각 칸에서 상하좌우  
어느 쪽으로 갈지를 결정한다.

(b) 칸의 좌표



최단 경로는  
 $1 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 11 \rightarrow 15 \rightarrow 16$ 의  
7단계

- 미로의 강화 학습으로 최대한 빨리 출구에 도달하는 것을 목적으로, 어떤 칸에서 어느 쪽으로 진행할지에 정책을 학습 한다



# 가치기반 방식: Q 학습

- (a) Q 학습에서는 1회 행동 할 때마다 현재 칸의 가치를 옆 칸의 가치에 맞추도록 파라미터 갱신을 실시한다.
- (b)  $\epsilon$ -그리디 알고리즘에 의해 최대 가치의 방향으로  $(1-\epsilon)$ 의 확률로 이동하는 식의 정책을 채택한다

- Q 학습

- 한 번 행동할 때마다 행동 가치 함수  $Q(s, a)$ 를 갱신

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot \Delta Q$$

$$\Delta Q = r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)$$

오차

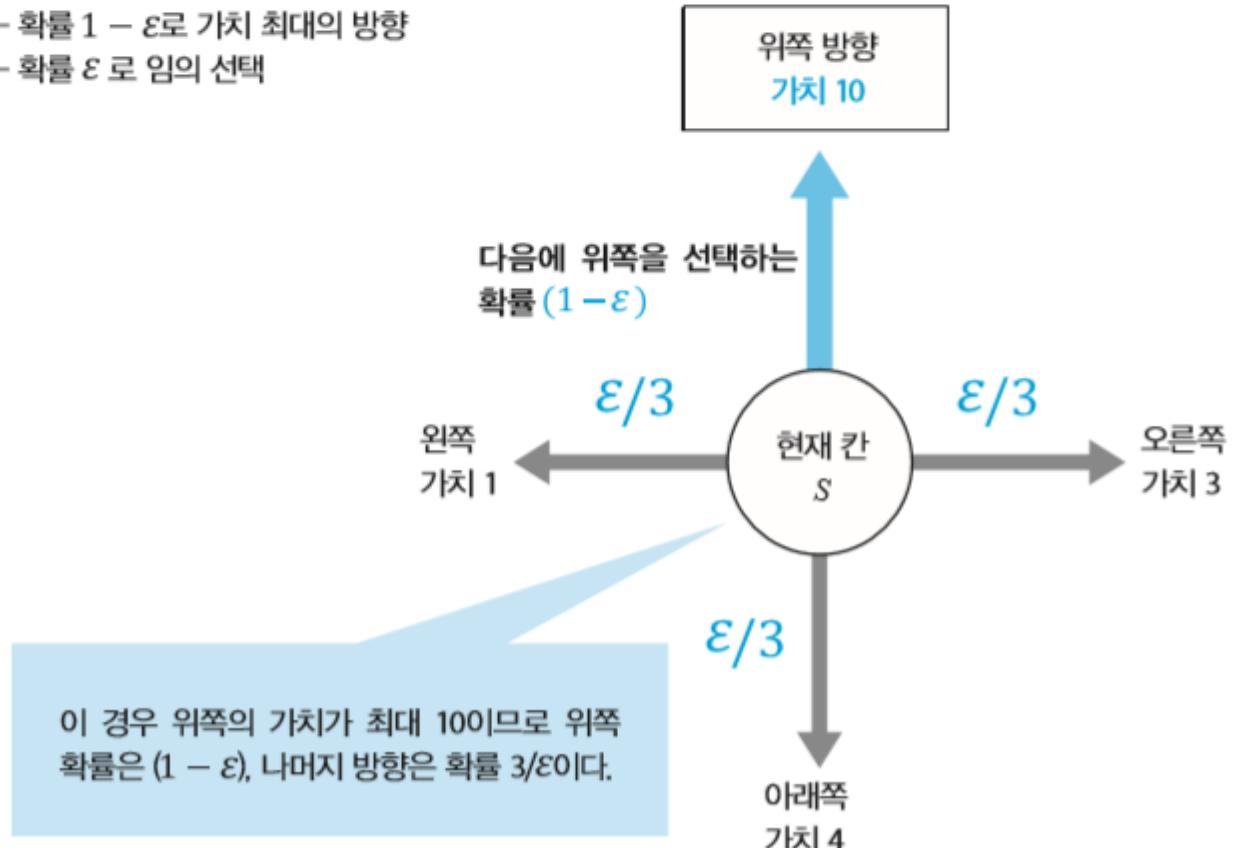
옆 칸 중 가치가 최대가 되는  
것의 가치

현재 칸의 가치

$r$ : 다음에 얻어지는 보상,  $s'$ : 다음의 칸,  $\gamma$ : 할인율

- 정책:  $\epsilon$ -그리디 알고리즘

- 확률  $1 - \epsilon$ 로 가치 최대의 방향
  - 확률  $\epsilon$ 로 임의 선택

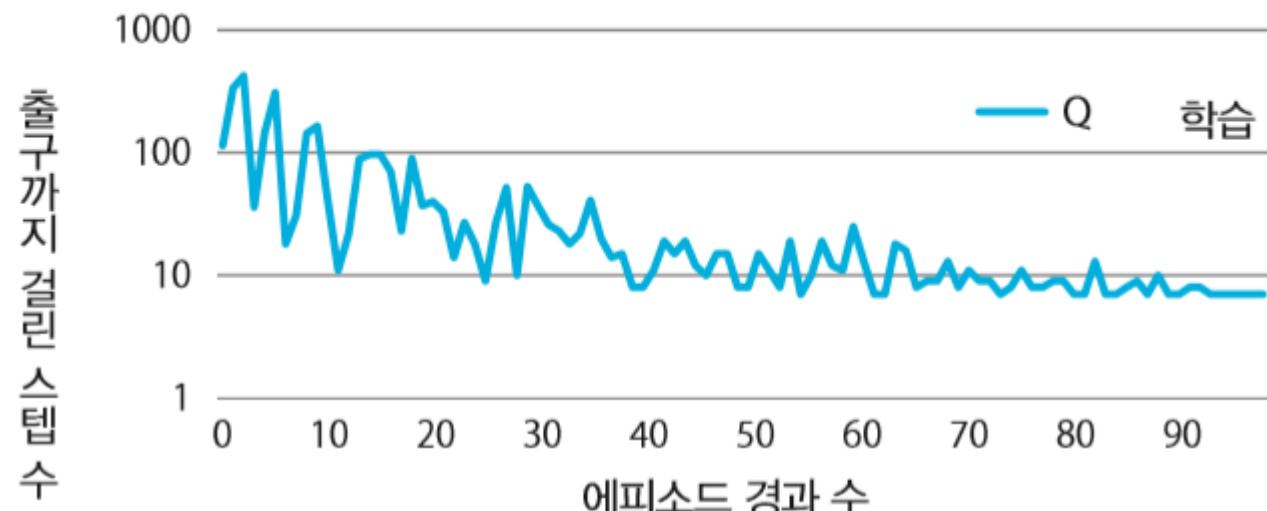


# 미로의 Q 학습 결과

(a) 처음에는 랜덤 이동이므로 출구까지 100 스텝 이상 걸릴 수도 있지만, 학습이 진행되면 최단 스텝인 7 스텝으로 출구에 도달할 수 있게 된다.

(b) 가치 함수의 값은 처음에는 출구 지점에만 값이 부여되지만, 학습이 진행되면 시작으로 향하는 경로상에 큰 값이 부여된다.

(a) 학습의 진행에 따라 출구까지의 스텝 수가 감소



(b) 가치 함수 값의 변화

에피소드 = 1			
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

에피소드 = 20			
1	0	0	0
1	2	5	0
0	1	8	0
0	0	13	20

에피소드 = 100			
33	8	0	0
41	50	61	0
13	11	73	0
0	0	86	100

※ 각 칸에서 (d)의 행동을 선택할 경우의 가치 함수 값을 나타냈다.

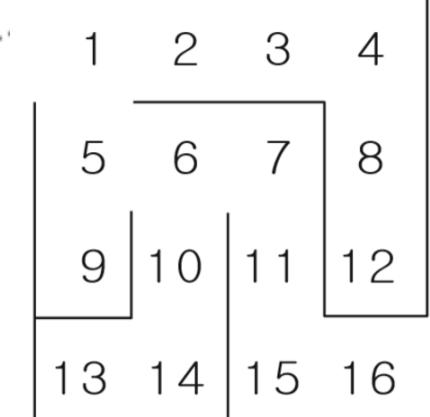
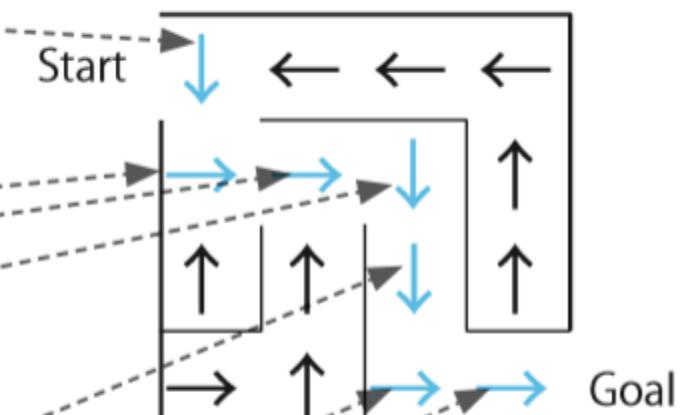
# 미로의 Q 학습 결과

(c)(d) 각 지점에서 가치 최대의 행동을 연결하면 시작부터 출구까지의 최단 경로가 된다

(c) 이동 가치 함수  $Q(s, a)$ 의 학습 결과  
(에피소드 = 100)

상태 $s(\text{칸})$	행동 $a$			
	$\uparrow$	$\rightarrow$	$\downarrow$	$\leftarrow$
1	6.8	2.9	33.3	6.6
2	0.7	0.1	0.6	8.1
3	0.0	0.0	0.1	0.4
4	0.0	0.0	0.0	0.1
5	10.0	41.1	4.0	12.0
6	21.7	50.5	2.7	7.6
7	14.7	16.3	61.1	12.8
8	0.0	0.0	0.0	0.0
9	12.8	0.6	1.0	0.9
10	10.7	0.4	0.1	0.7
11	17.8	26.4	72.7	14.9
12	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0
14	0.3	0.0	0.0	0.0
15	30.1	85.5	27.0	31.5
16	21.8	100.0	30.5	13.1

(d) 이동 가치 함수를  
최대화하는 경로



# 정책기반 방식: 정책 경사법

- (a) 정책 경사법에서  
는 1 에피소드마다  
해당 에피소드에서  
나타난 행동을 다음  
이후에도 선택하기  
쉽도록 파라미터 갱  
신을 실시한다.
- (b) 정책 경사법에서  
는  $\pi(s, a)$ 를 이용한 소  
프트맥스 함수에 의  
해 다음에 상하좌우  
로 이동하는 확률을  
정한다

- 정책 경사법

- 1 에피소드마다 정책 함수의 파라미터  $\pi(s, a)$ 를 갱신

$$\pi(s, a) \leftarrow \pi(s, a) + \Delta\pi(s, a)$$

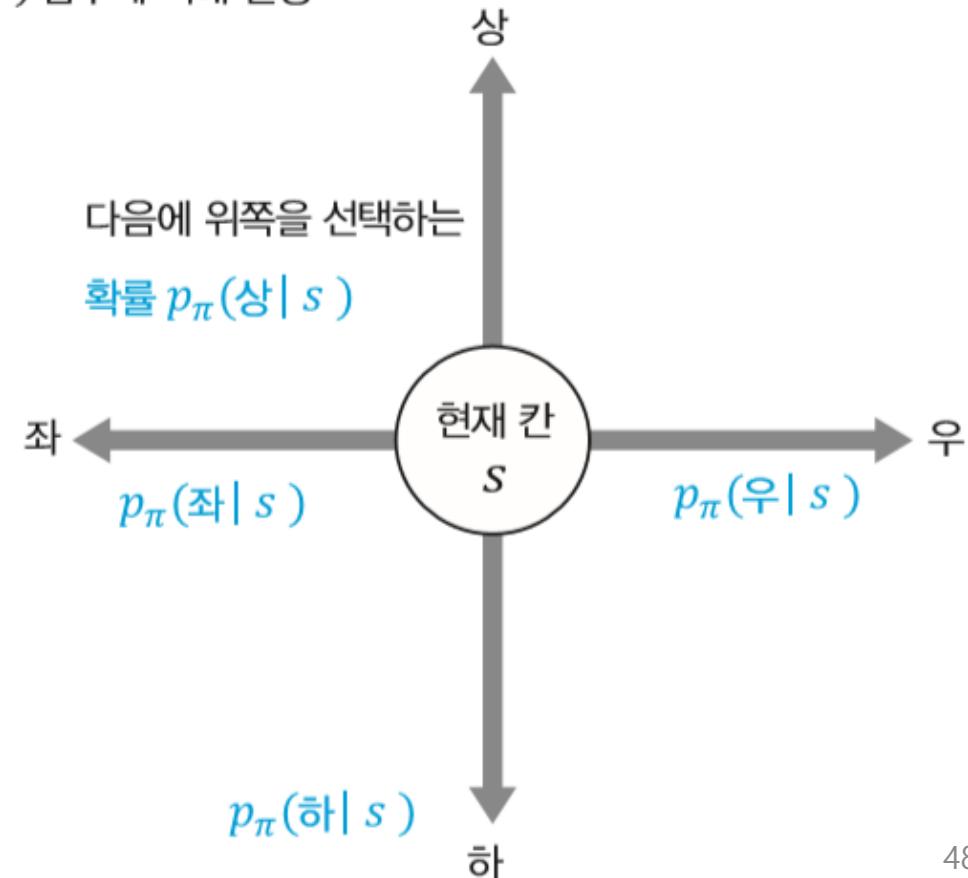
$$\Delta\pi(s, a) \sim (\text{상수}) \cdot N_1 - (\text{정수}) \cdot N_2$$

$N_1$ : 상태  $s$ 에서 행동  $a$ 를 선택한 횟수

$N_2$ : 상태  $s$ 에서 행동  $a$  이외를 선택한 횟수

- 정책

- $\pi(s, a)$ 인 소프트맥스  $p_\pi(a | s)$  함수에 의해 결정

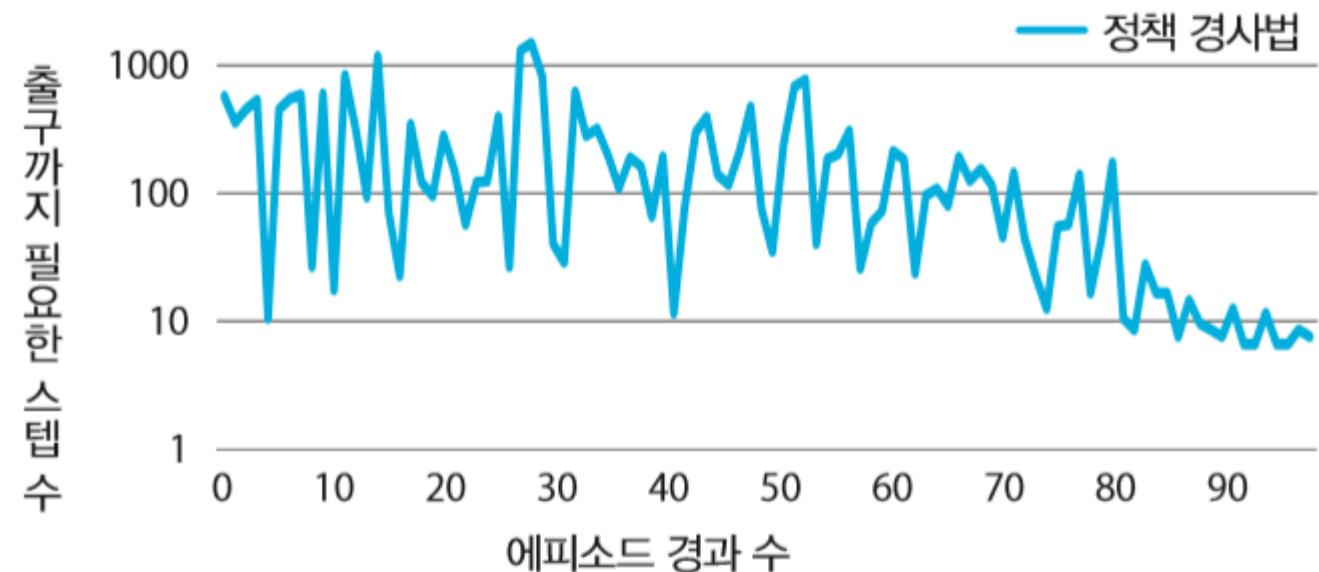


# 미로의 정책 경사법에 의한 학습 결과

(a) 처음에는 랜덤 이동으로 출구까지 1,000 스텝 이상 걸릴 수도 있지만, 학습이 진행되면 최단 스텝인 7 스텝으로 출구에 도달할 수 있게 된다.

(b) 정책 함수의 값은 처음에는 랜덤이므로 25%씩이지만, 학습이 진행되면 최단 경로의 방향으로 큰 확률값이 부여된다.

(a) 학습의 진행에 따라 출구까지의 스텝 수가 감소



(b) 정책 함수 값의 변화

에피소드 = 1

26%	24%	25%	26%
25%	24%	24%	24%
25%	26%	25%	25%
25%	25%	25%	26%

에피소드 = 20

26%	28%	28%	29%
26%	28%	35%	25%
30%	31%	27%	25%
28%	28%	35%	43%

에피소드 = 100

99%	49%	30%	25%
100%	99%	100%	25%
42%	42%	99%	28%
35%	34%	98%	99%

※ 각 칸에서 (d)의 행동을 채택할 확률의 값을 나타냈다.

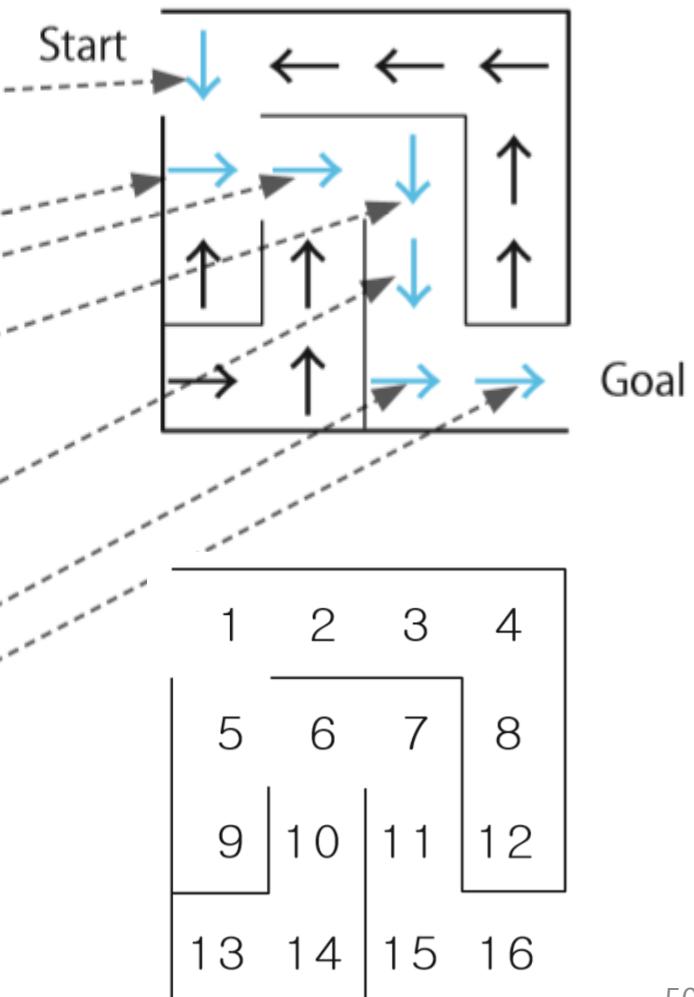
# 미로의 정책 경사법에 의한 학습 결과

(c)(d) 각 지점에서 정책  
함수 최대의 행동을 연결  
하면 시작 부터 출구까지  
의 최단 경로가 된다

(c) 정책 함수의 학습 결과  
(에피소드 = 100)

상태 <i>s</i> (칸)	행동a			
	↑	→	↓	←
1	0%	0%	99%	0%
2	15%	22%	14%	49%
3	28%	15%	27%	30%
4	24%	24%	27%	25%
5	0%	100%	0%	0%
6	0%	99%	0%	1%
7	0%	0%	100%	0%
8	25%	22%	30%	23%
9	42%	16%	22%	21%
10	43%	12%	18%	26%
11	0%	0%	99%	0%
12	28%	30%	19%	23%
13	27%	35%	20%	18%
14	34%	15%	30%	21%
15	1%	98%	1%	0%
16	0%	99%	0%	0%

(d) 정책 함수를  
최대화하는 경로

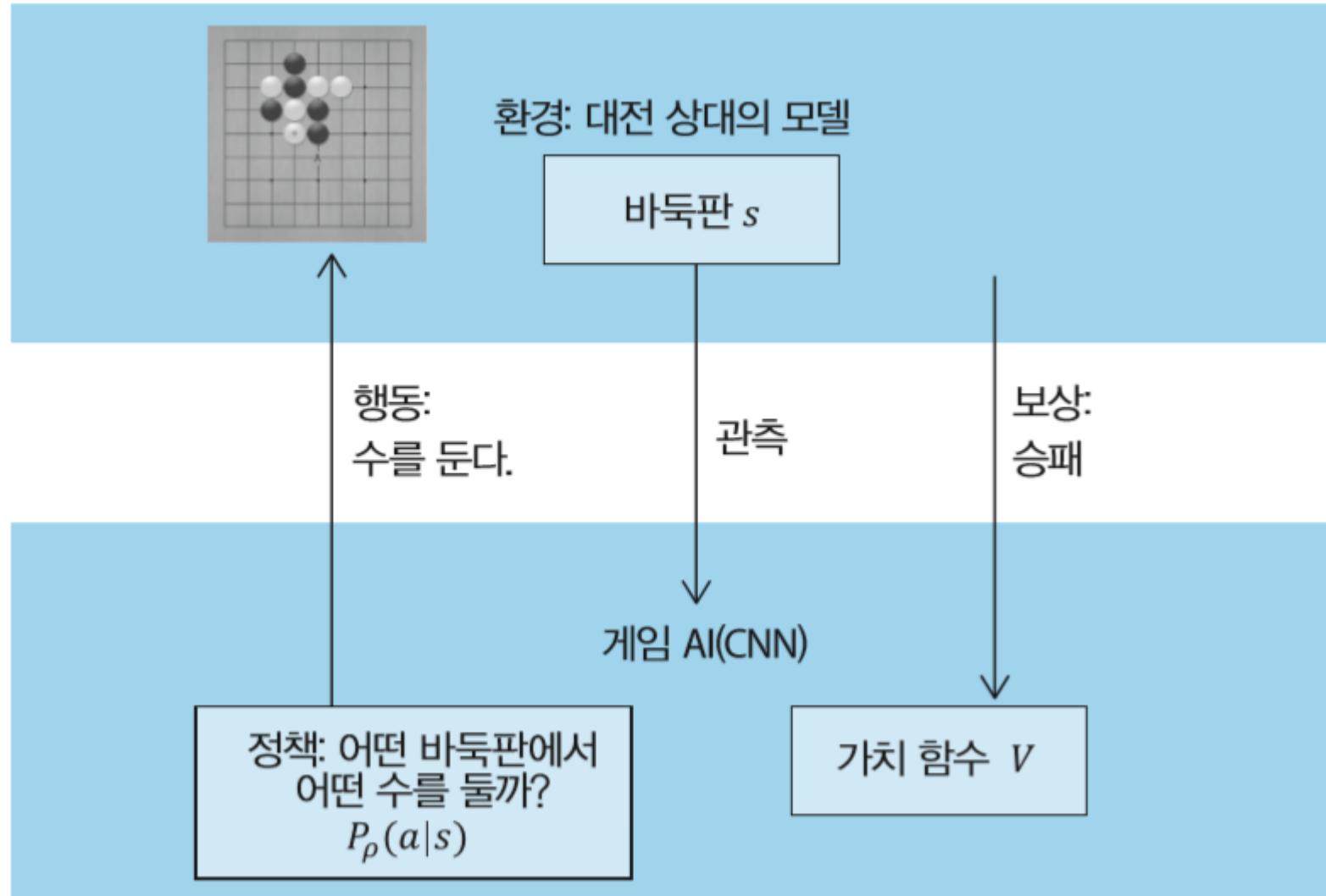


# 알파고의 강화학습

RL 정책 네트워크를 획득하기 위한 강화 학습의 개요

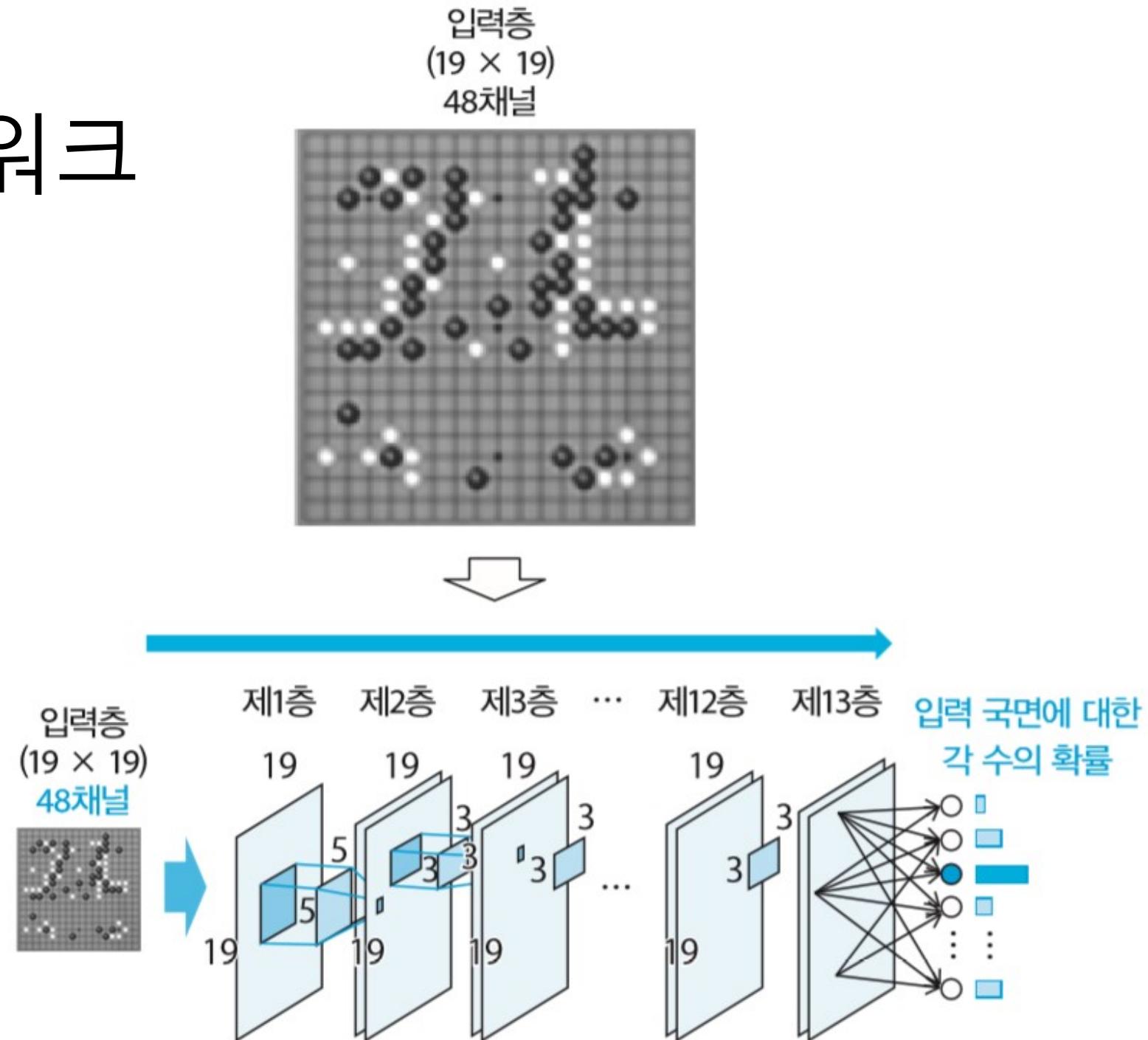
- 강화 학습의 목적: SL 정책 네트워크를 강화 학습함으로써 보다 쉽게 이기기 위한 정책 네트워크를 만든다. ⇒ RL 정책 네트워크
- 학습 방법: SL 정책 네트워크를 초기값으로 하여 '게임의 승리'를 보상으로 해서 정책 경사법에 의해 강화 학습
  - 이겼을 때는 승리에 이르는 수를 최대한 선택하도록 파라미터를 갱신
  - 졌을 때는 패배에 이르는 수를 최대한 피하도록 파라미터를 갱신
- 자기 대전에 의한 게임의 결과를 얻는 처리에 엄청난 시간이 필요하므로 학습에 50GPU라고 해도 약 1일이 소요된다.

# 바둑의 강화 학습 프레임워크



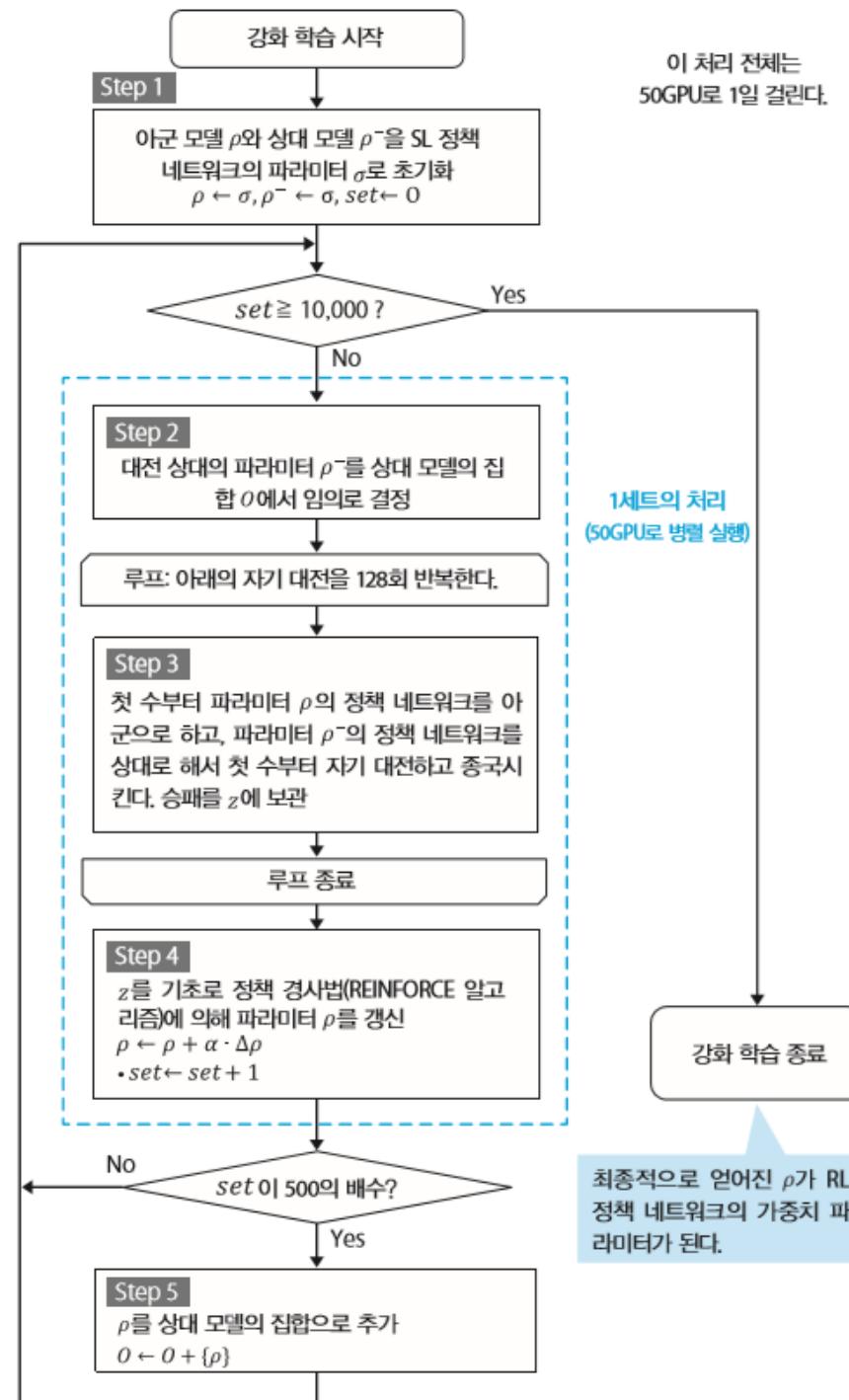
# RL 정책 네트워크

- 정책 함수에 해당하는 CNN
- 알파고의 RL 정책 네트워크의 강화 학습으로 대전 상대에 가능한 한 이기는 것을 목적 으로 어떤 국면에 서 어떤 수를 둘지를 학습한다



# 알파고에서 강화 학습의 플로차트

- REINFORCE 알고리즘
- 학습 시간
  - 종국까지 400수가 걸린다고 하면 아군과 상대의 정책 네트워크를 총 400회 실행해야 한다.
  - 이것을 GPU 1개로 사용하는 경우로 계산하면 종국까지 걸리는 시간은 5밀리초  $\times$  400수 = 2.0초이다.
  - 따라서 자기 대전 128회(1세트)에는 2x128초가 되고, 이것을 10,000세트 반복하려면 단순 계산으로 해서 약 30일( $10,000 \times 2 \times 128$ 초)이 소요된다.
  - 여기에 GPU 50개를 병렬로 사용하여 약 1일이 걸렸다고 한다.



## 강화 학습 시작

Step 1

아군 모델  $\rho$ 와 상대 모델  $\rho^-$ 을 SL 정책  
네트워크의 파라미터  $\sigma$ 로 초기화  
 $\rho \leftarrow \sigma, \rho^- \leftarrow \sigma, set \leftarrow 0$

이 처리 전체는  
50GPU로 1일 걸린다.

$set \geq 10,000 ?$

Yes

No

Step 2

대전 상대의 파라미터  $\rho^-$ 를 상대 모델의 집합  $O$ 에서 임의로 결정

루프: 아래의 자기 대전을 128회 반복한다.

Step 3

첫 수부터 파라미터  $\rho$ 의 정책 네트워크를 아군으로 하고, 파라미터  $\rho^-$ 의 정책 네트워크를 상대로 해서 첫 수부터 자기 대전하고 종국시킨다. 승패를  $z$ 에 보관

루프 종료

1세트의 처리  
(50GPU로 병렬 실행)

Step 3

첫 수부터 파라미터  $\rho$ 의 정책 네트워크를 아군으로 하고, 파라미터  $\rho^-$ 의 정책 네트워크를 상대로 해서 첫 수부터 자기 대전하고 종국시킨다. 승패를  $z$ 에 보관

루프 종료

Step 4

$z$ 를 기초로 정책 경사법(REINFORCE 알고리즘)에 의해 파라미터  $\rho$ 를 갱신  
 $\rho \leftarrow \rho + \alpha \cdot \Delta\rho$   
 $\cdot set \leftarrow set + 1$

No

$set$ 이 500의 배수?

Yes

Step 5

$\rho$ 를 상대 모델의 집합으로 추가  
 $O \leftarrow O + \{\rho\}$

강화 학습 종료

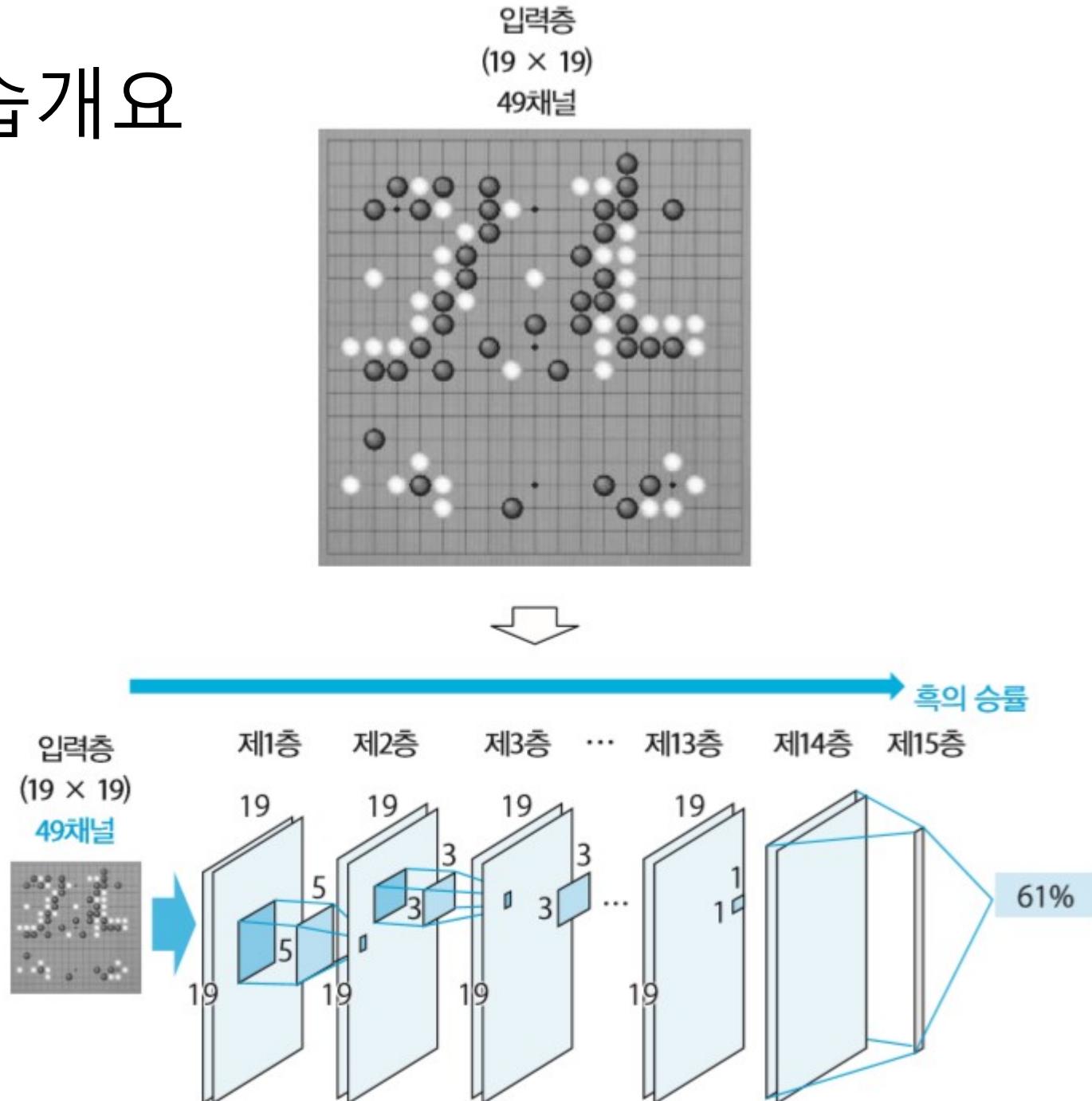
최종적으로 얻어진  $\rho$ 가 RL 정책 네트워크의 가중치 파라미터가 된다.

# RL 정책 네트워크의 성능

- 원래의 SL 정책 네트워크와의 대결에서 80% 승리
- 밸류 네트워크의 학습 데이터를 만드는 용도로 사용
- 몬테카를로 트리 탐색과 조합하여 사용할 때는 원래의 SL 정책 네트워크 쪽이 RL 정책 네트워크보다 궁합이 잘 맞음

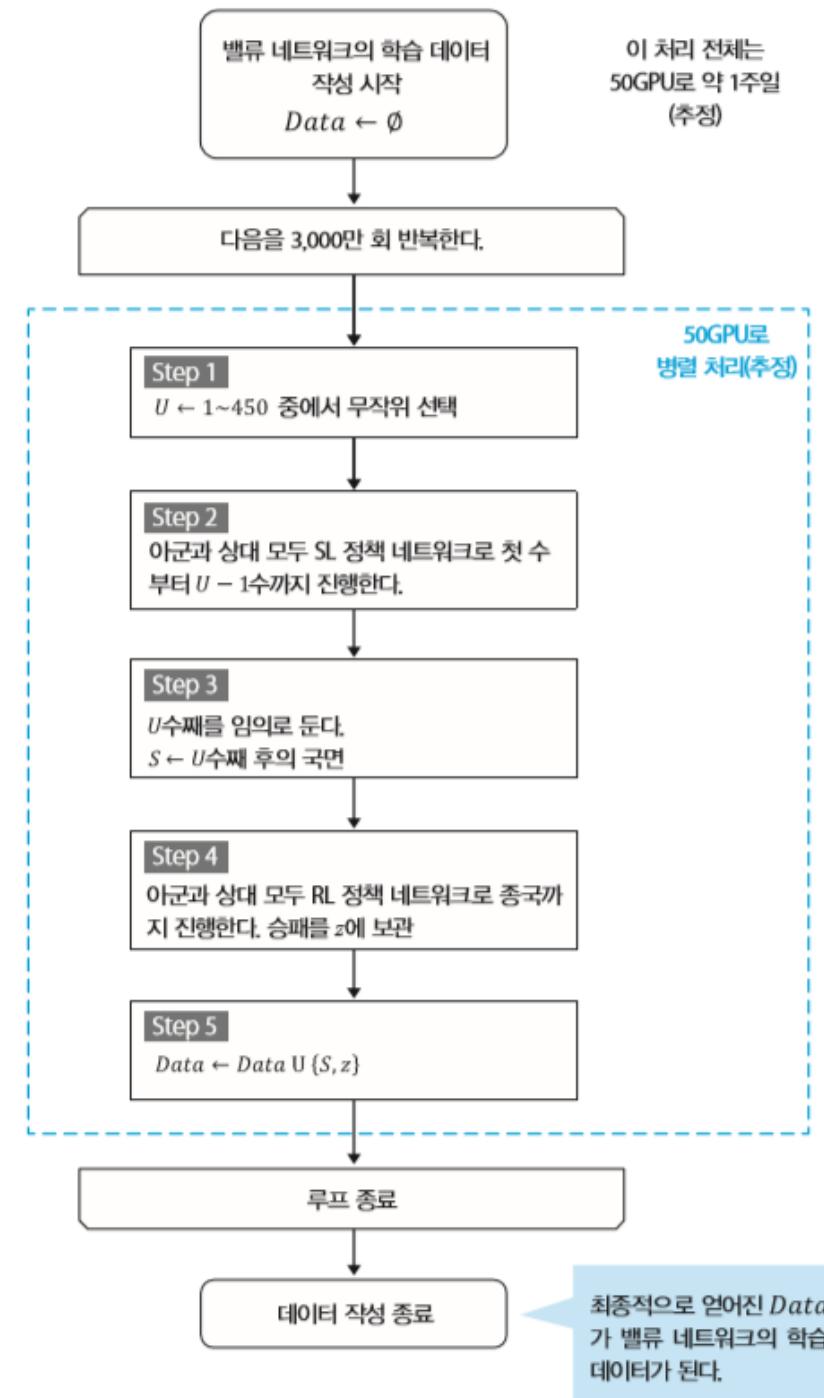
# 밸류 네트워크의 학습개요

- 밸류 네트워크는 국면을 입력하고, 승률 예측값을 출력하는 CNN
- 학습 방법: CNN이 출력하는 승률 예측값이 학습 데이터의 승패에 가까워지도록 필터 가중치를 갱신
- 학습 데이터: 국면과 승패의 조합 3,000만 개
  - 국면을 만드는 방법: SL 정책 네트워크에 의해 U수째까지 진행, 그 후에 임의의 수를 1 수 진행한다.
  - 승패를 만드는 방법: RL 정책 네트워크 끼리의 자기 대전 결과에 의해 근사
  - 이 학습 데이터의 생성 자체에 50GPU로 1주일이 소요(추정)
- 학습 자체에도 50GPU로 1주일 소요
- 기존에 곤란했던 바둑의 평가 함수 완성



# 밸류 네트워크의 학습 데이터 작성의 플로 차트

- SL 정책 네트워크에 사용된 16만개의 기보(3,000만 개의 국면)에서는 각 기보에서 1개의 학습 데이터만 추출할 수 있어서 학습 데이터로는 부족함
  - 동일한 기보에서 여러 국면을 추출하면 입력 데이터 간의 상관 관계가 높아 학습이 잘 안됨
- 알파고에서는 RL 정책 네트워크 끼리의 자기 대전에 근거해 기보를 생성하고, 그 기보에 근거해 학습 자료를 만드는 정책을 채택함



밸류 네트워크의 학습 데이터  
작성 시작  
 $Data \leftarrow \emptyset$

이 처리 전체는  
50GPU로 약 1주일  
(추정)

다음을 3,000만 회 반복한다.

Step 1  
 $U \leftarrow 1 \sim 450$  중에서 무작위 선택

Step 2  
아군과 상대 모두 SL 정책 네트워크로 첫 수  
부터  $U - 1$ 수까지 진행한다.

Step 3  
 $U$ 수째를 임의로 둔다.  
 $S \leftarrow U$ 수째 후의 국면

50GPU로  
병렬 처리(추정)

Step 3

$U$ 수째를 임의로 둔다.  
 $S \leftarrow U$ 수째 후의 국면

Step 4

아군과 상대 모두 RL 정책 네트워크로 종국까  
지 진행한다. 승패를  $z$ 에 보관

Step 5

$Data \leftarrow Data \cup \{S, z\}$

루프 종료

데이터 작성 종료

최종적으로 얻어진  $Data$   
가 밸류 네트워크의 학습  
데이터가 된다.

# 밸류 네트워크의 학습 데이터 작성의 플로 차트

- Step 1에서 최대값이 450은 바둑판의 점의 수인  $19 \times 19 = 361$ 보다도 크다. 승패가 거의 확정적인 종반의 국면을 중점적으로 학습시키고자 하는 의도가 있을지도 모른다.
- Step 2에서 상대적으로 강한 RL 정책 네트워크를 사용하는 것이 아니라 굳이 SL 정책 네트워크를 사용하는 것은 'SL 정책 네트워크의 변화의 폭이 더 컷기 때문'이다.
- Step 3에서 거의 있을 수도 없는 수를 포함하여 무작위로 수를 선택하는 부분에도 여하튼 바리에이션을 크게 하려는 의도가 느껴진다.

# 탐색 - 바둑 AI는 어떻게 예측할까?

알파고의 예측 능력을 지원하는 것은 탐색이라는 기술이다. 게임에서 기준의 검색 기술은 모든 후보를 열거하여 게임 트리를 만들고, 그 속을 '전부 다' 조사해 최선의 것을 선택하는 방식이었다.

고속 검색은 컴퓨터가 자랑하는 분야이며, 장기와 체스가 강해진 것은 이 '완전 탐색 (exhaustive search)'의 기여가 크다. 바둑에서는 오랫동안 '탐색이 어렵다'라고 생각했지만, 2006년에 '몬테카를로 트리 탐색'이라는 획기적인 기술이 탄생하였다.

이 장에서는 랜덤 시뮬레이션의 승패를 바탕으로 조금씩 트리를 성장시키는 몬테카를로 트리 탐색의 원리와 특징에 대해 설명한다.

# 4. 예측하는 AI: 탐색

4.1 2인 제로섬 유한 확정 완전 정보

게임

4.2 게임에서의 탐색

4.3 기존의 게임 트리 탐색

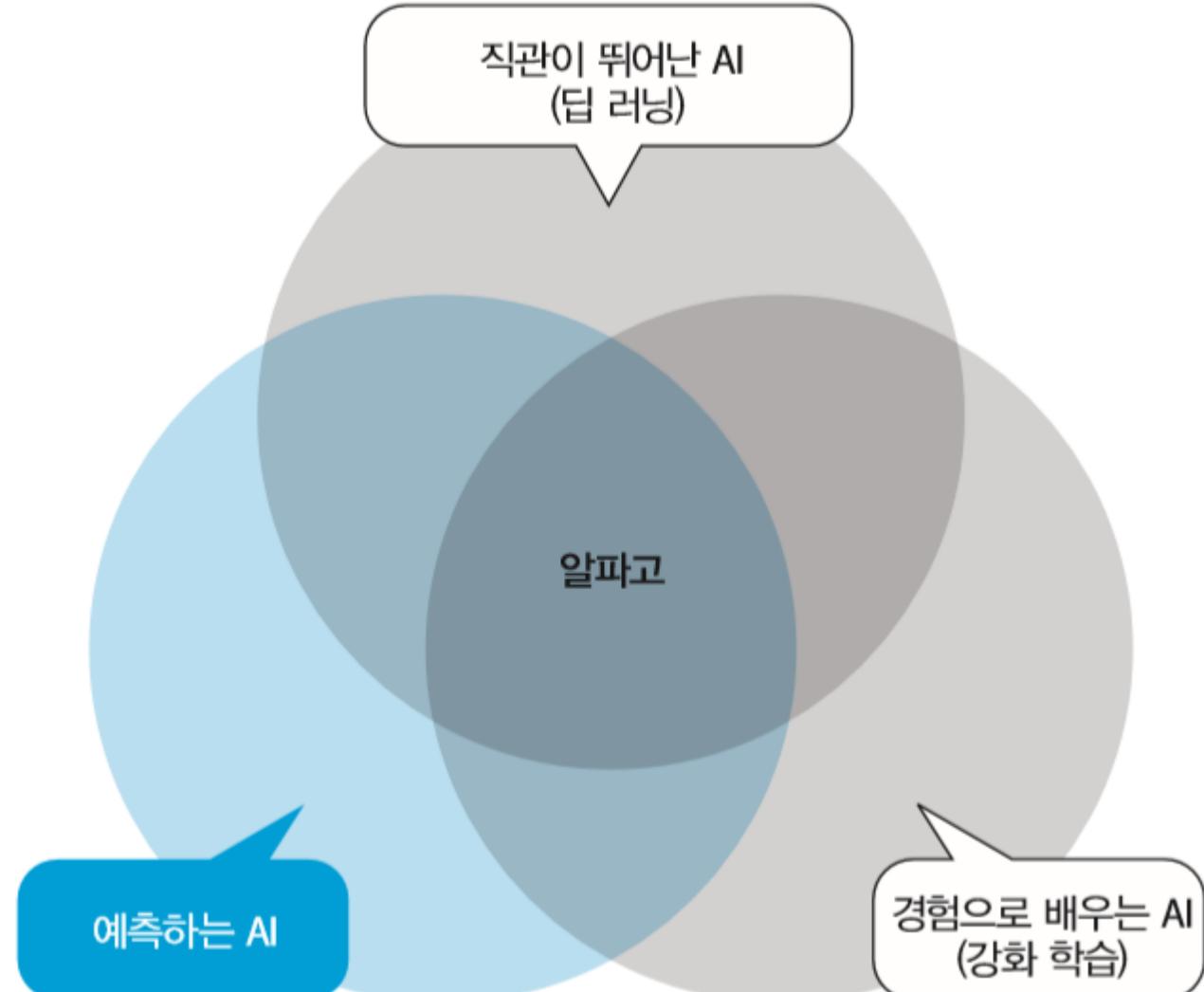
(민맥스 트리 탐색)

4.4 바둑에서의 몬테카를로 트리

탐색

4.5 몬테카를로 트리 탐색의 성공

요인과 과제

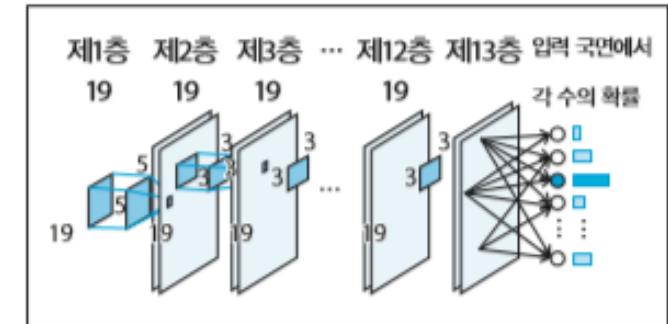


# 정책 네트워크에서 탐색으로

- 게임 트리
- 1수 앞의 예측
  1. 1수 진행한다
  2. 승률을 계산한다
  3. 승률이 가장 높은 수를 선택한다.
- d수 앞을 예측
  - 평가의 정확도가 높아짐
  - 계산량이 많아짐 :  $w^d$  ( $w$ : 어느 국면에서 합법수의 개수)

- SL 정책 네트워크에서 탐색으로

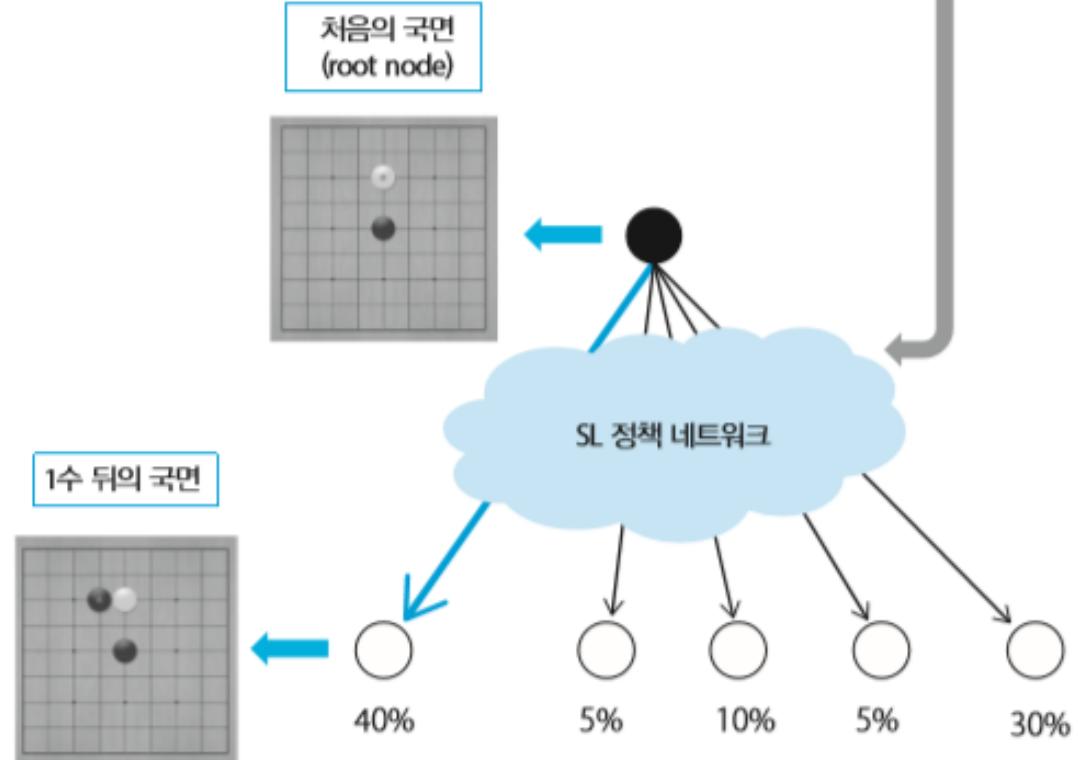
— 정책 네트워크는 90도 회전하면.....



- 이 경우

..... 1수 진행하고..... 승률을 계산하고..... 가장 승률이 높은 수를 고르고 있다.

- 2수 이상 진행하면 보다 정확한 평가를 할 수 있을 것으로 예측된다. ⇒ 탐색

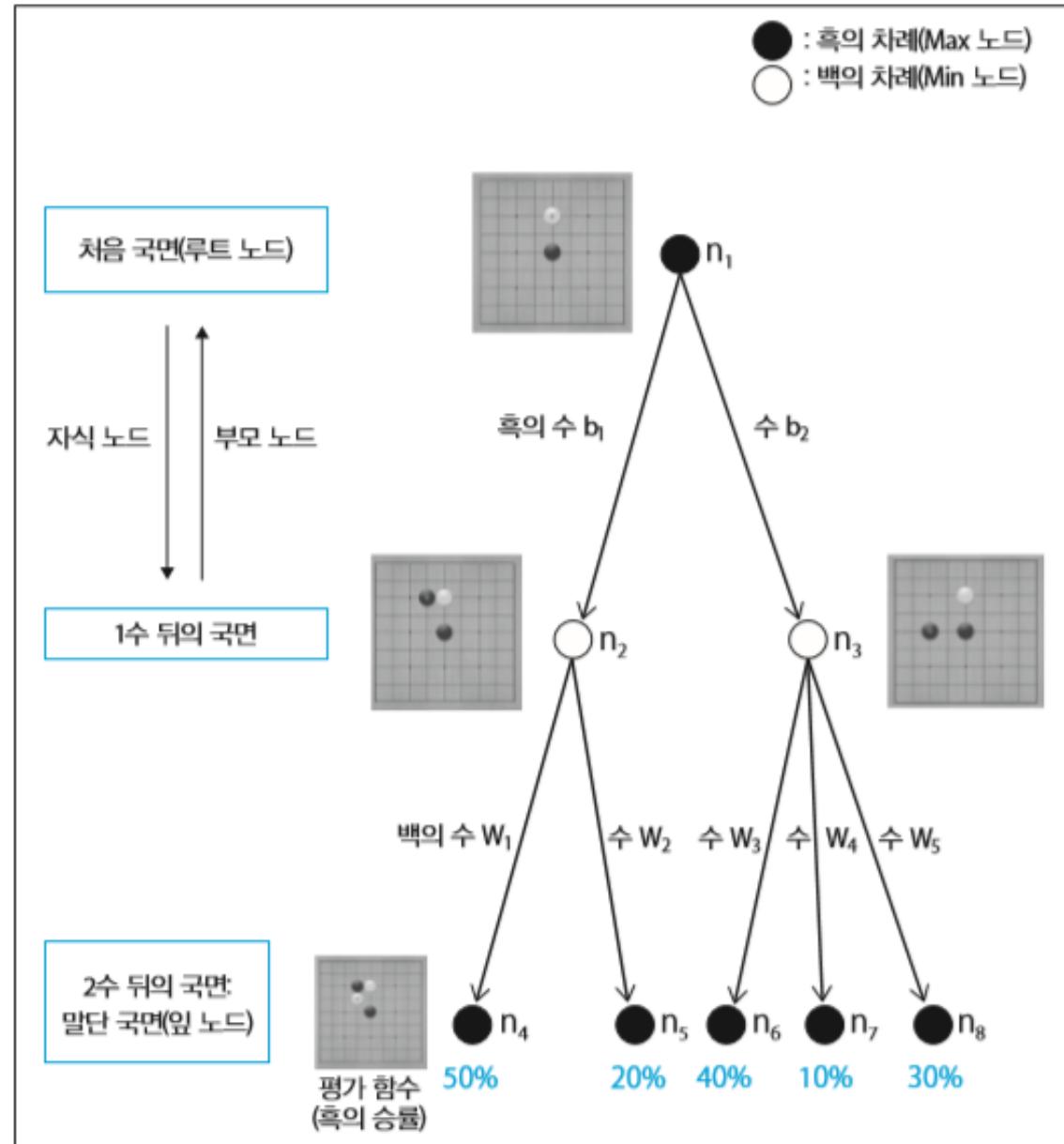


# 기존의 게임 트리 탐색

- 완전 탐색
  - 합법 수의 수가 적은 경우
  - 민맥스 트리

# 민맥스 트리의 예

- 국면이 노드에 대응하고, '플레이어의 수'가 아크에 대응한다. 그리고 첫 번째 국면이 루트 노드가 된다. 바둑 같은 2인 게임의 경우 Max 노드와 Min 노드가 고대로 나타난다.
- 평가 함수가 흑의 승률로 표시될 경우, 흑의 차례(선수)는 승률을 극대화하면 되고, 백의 차례(후수)는 승률을 최소화하면 될 것이다.
- 흑의 차례(선수)의 노드는 Max 노드, 백의 차례(후수)의 노드는 Min 노드라고 부른다.



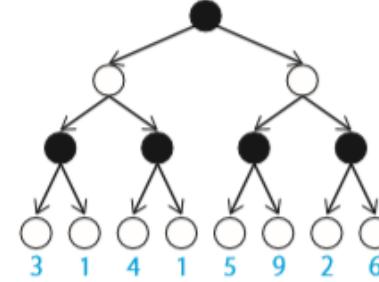
# 민맥스 트리 탐색의 예

흑의 최선의 수를 결정하려면

- 앞 노드에 평가값을 매긴다.
- 자식 노드의 MAX를 취한다.
- 자식 노드의 MIN을 취한다.
- 자식 노드의 MAX를 취한다.
- 최종적으로, 루트 노드의 왼쪽 자식 노드의 평가값은 3, 오른쪽 자식 노드의 평가값은 6이 되어서 최선의 수는 오른쪽 수가 된다.

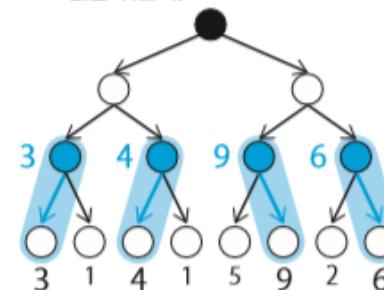
※ 나도 최선을 다하지만 상대도 최선을 다하고 가정함

(a) 앞 노드에 평가값을 매긴다.

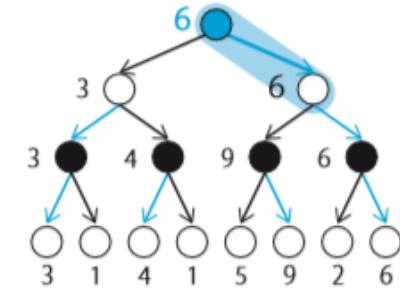


● : 흑의 차례(Max 노드)  
○ : 백의 차례(Min 노드)

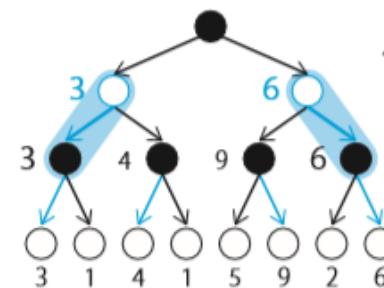
(b) Max 노드에서는 자식 노드 중에서 큰 값을 취한다.



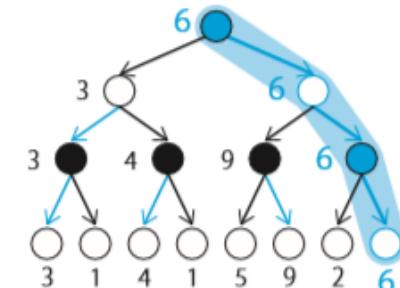
(d) 다시 한번 Max 노드에서는 자식 노드 중에서 큰 값을 취한다.



(c) Min 노드에서는 자식 노드 중에서 작은 값을 취한다.



(e) 탐색 결과



# 알파베타법

- 민맥스 트리 탐색에서 능숙한 가지치기를 이용해 민맥스 트리의 올바른 평가값을 효율적인(즉, 소수의 노드를 검색하는) 탐색을 통해 얻는 방법

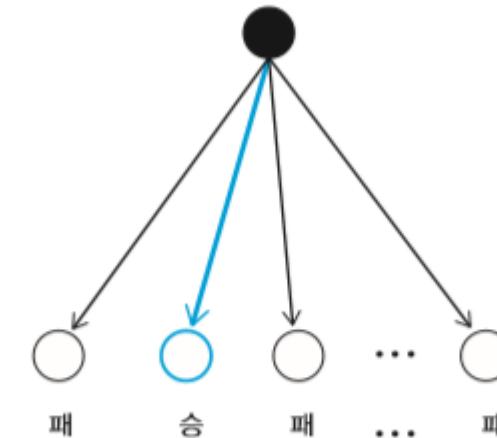
# 게임 트리 탐색의 포인트

- 장기, 체스 등의 '완전' 탐색에서는
  - (1) 탐색의 깊이(얼마나 중요한 변화를 깊게 탐색할까?)
  - (2) 평가 함수의 질(얼마나 정확하게 우열을 평가할까?)

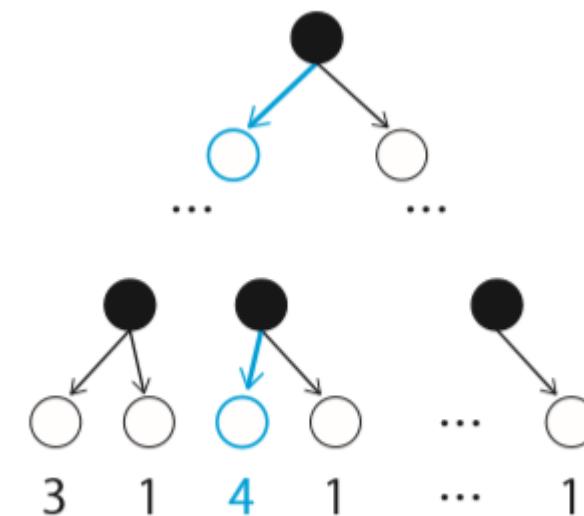
이 둘 다 중요

- 완벽한 평가 함수를 만드는 것은  
일반적으로 불가능하므로 깊이 제  
어와 평가 함수의 질을 함께 높여  
야 한다

(a) 완벽한 평가 함수의 경우 → 깊이 1의 탐색으로 충분



(b) 평가 함수가 랜덤인 경우 → 아무리 깊게 탐색해도 무의미



- 한편, 바둑은 합법 수가 많고 정확한 평가가 어려우므로 '완전' 탐색이 어렵다.

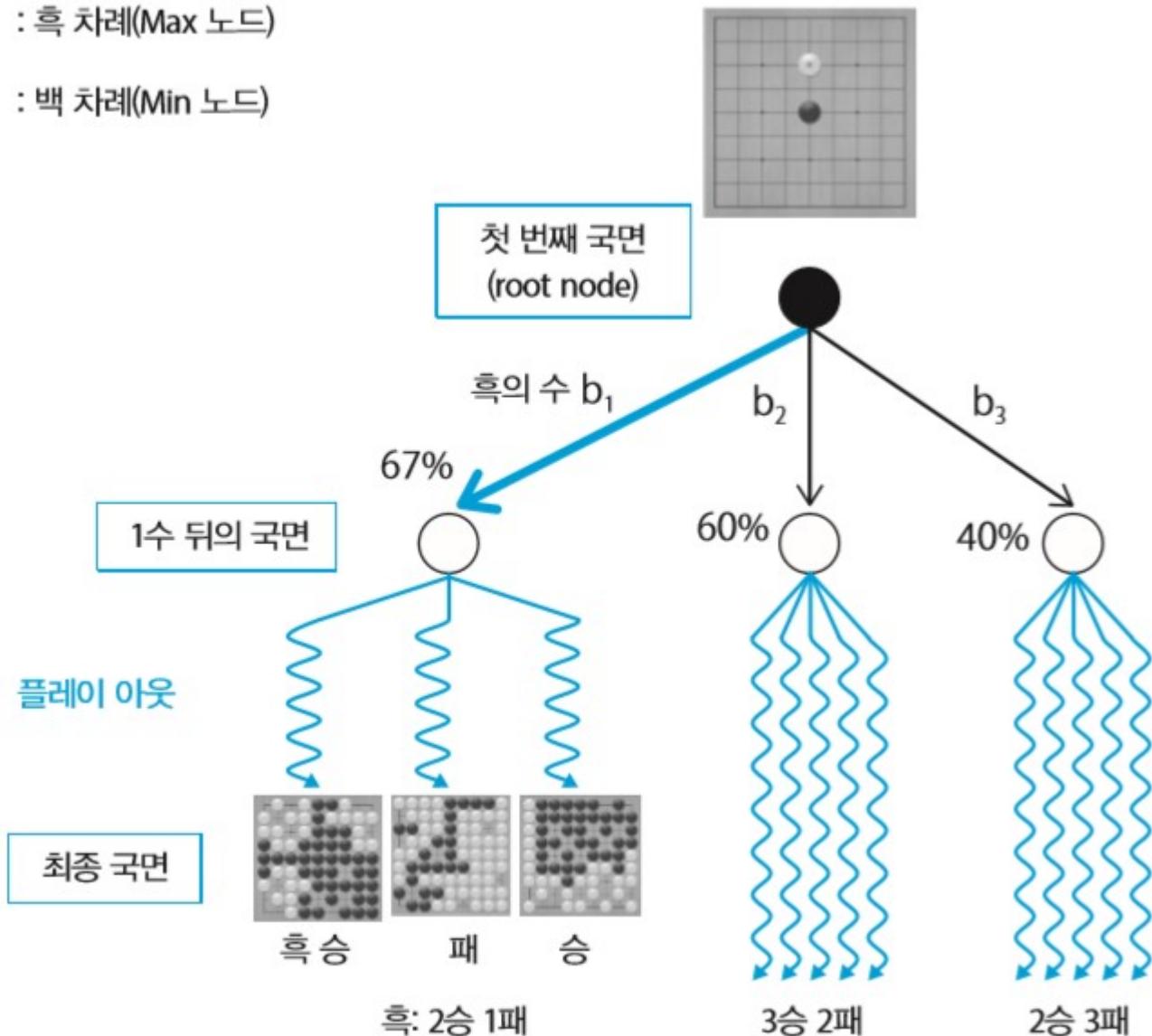
# 원시 몬테카를로

- 원시 몬테카를로란 랜덤 대전(플레이 아웃)의 결과를 바탕으로 루트 노드의 1수 뒤의 국면의 승률을 결정하는 기법

※ 몬테카를로 방법 : 랜덤 시뮬레이션을 기반으로 하는 방법

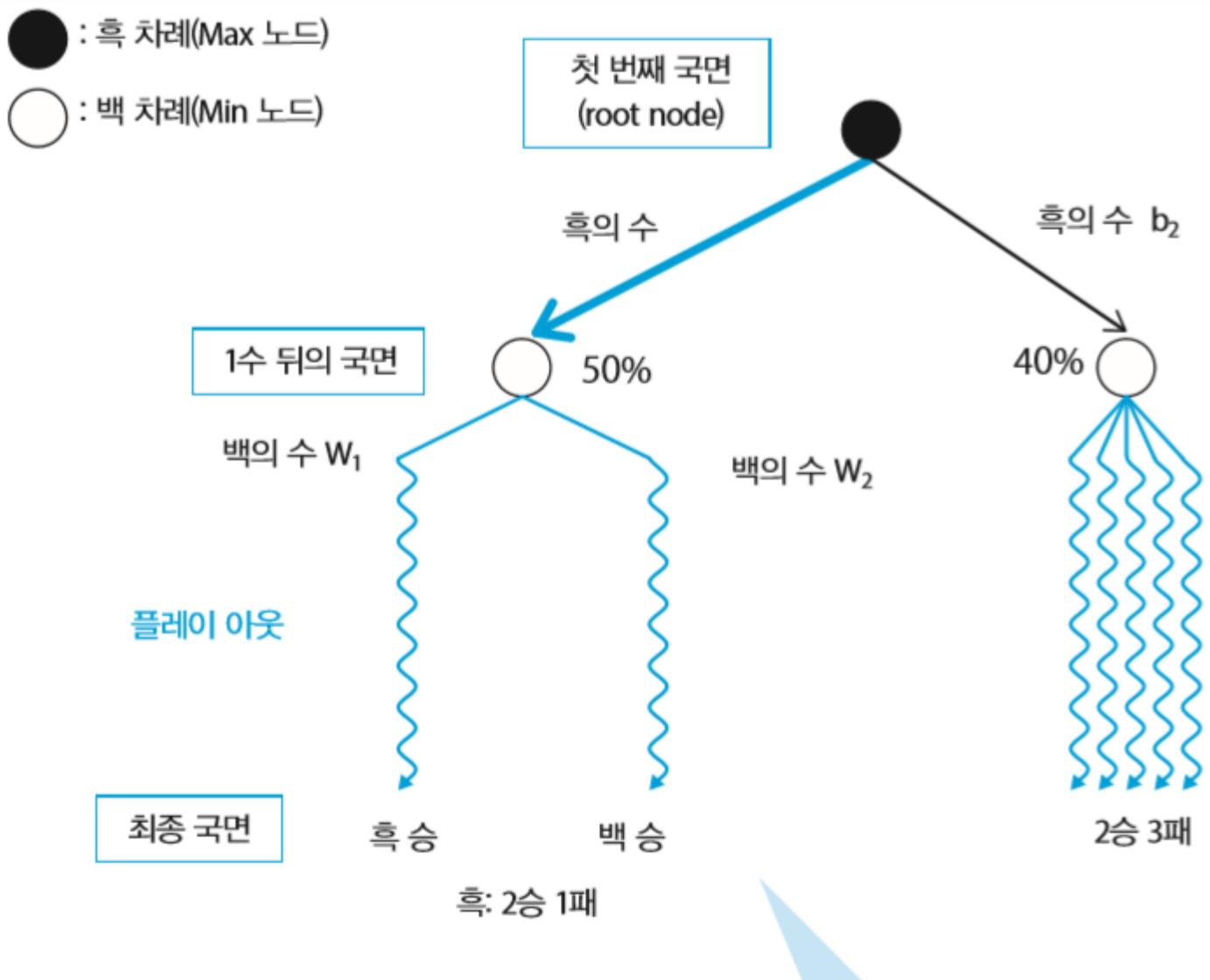
※ 플레이 아웃 : 특정 국면에서 흑의 순서와 백의 순서가 짧은 시간으로 교대로 수를 선택하고, 종국까지 진행하는 기법

● : 흑 차례(Max 노드)  
○ : 백 차례(Min 노드)



# 원시 몬테카를로

- 원시 몬테카를로의 과제:  
상대에게 좋은 수가 하나  
만 있는 경우를 간과하기  
때문에 잘못된 선택을 하  
고 만다



- 백이 수  $w_2$ 를 선택했을 때 반드시 백의 승리라면 수  $b_1$ 의 진정한 승률은 0%
- 그러나 임의의 플레이 아웃에서는 수  $w_2$ 는 2회에 1회밖에 선택되지 않으므로 보여지는 승률은 50%로, 수  $b_1$ 이 좋은 수로 보인다.

# 몬테카를로 트리 탐색

- 유망한 수를 더 깊게 조사함으로써 탐색의 정확도를 높여서 원시 몬테카를로의 문제를 해결
- 4가지 Step으로 구성됨

## 1. 선택(selection)

잎 노드에 이르기까지 UCB1(승률+바이어스)이 가장 큰 자식 노드를 선택하여 수를 진행해 트리를 내려간다

## 2. 확장(expansion)

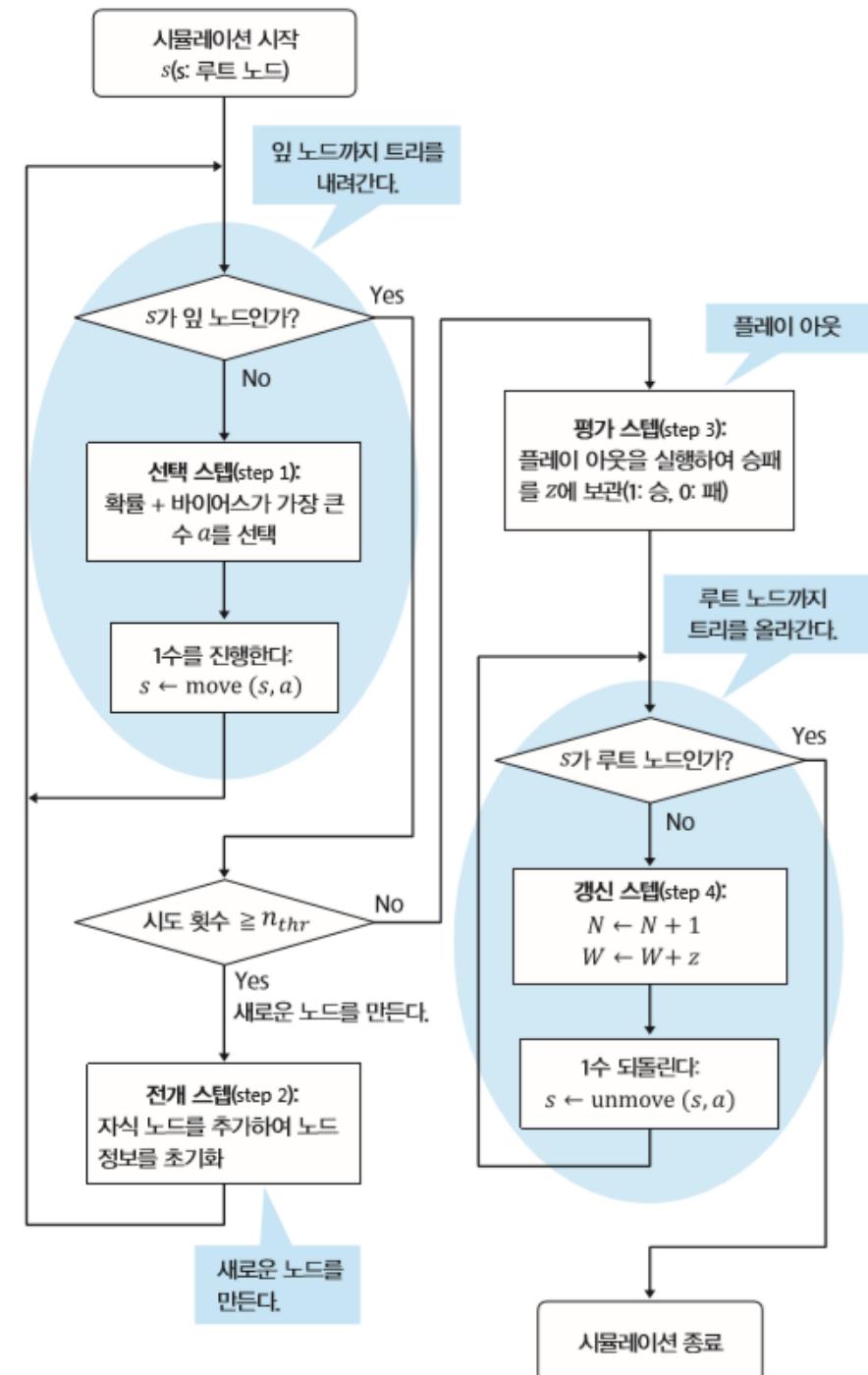
시도 횟수가  $n_{thr}$  이상이 된 경우 새로운 노드를 작성한다

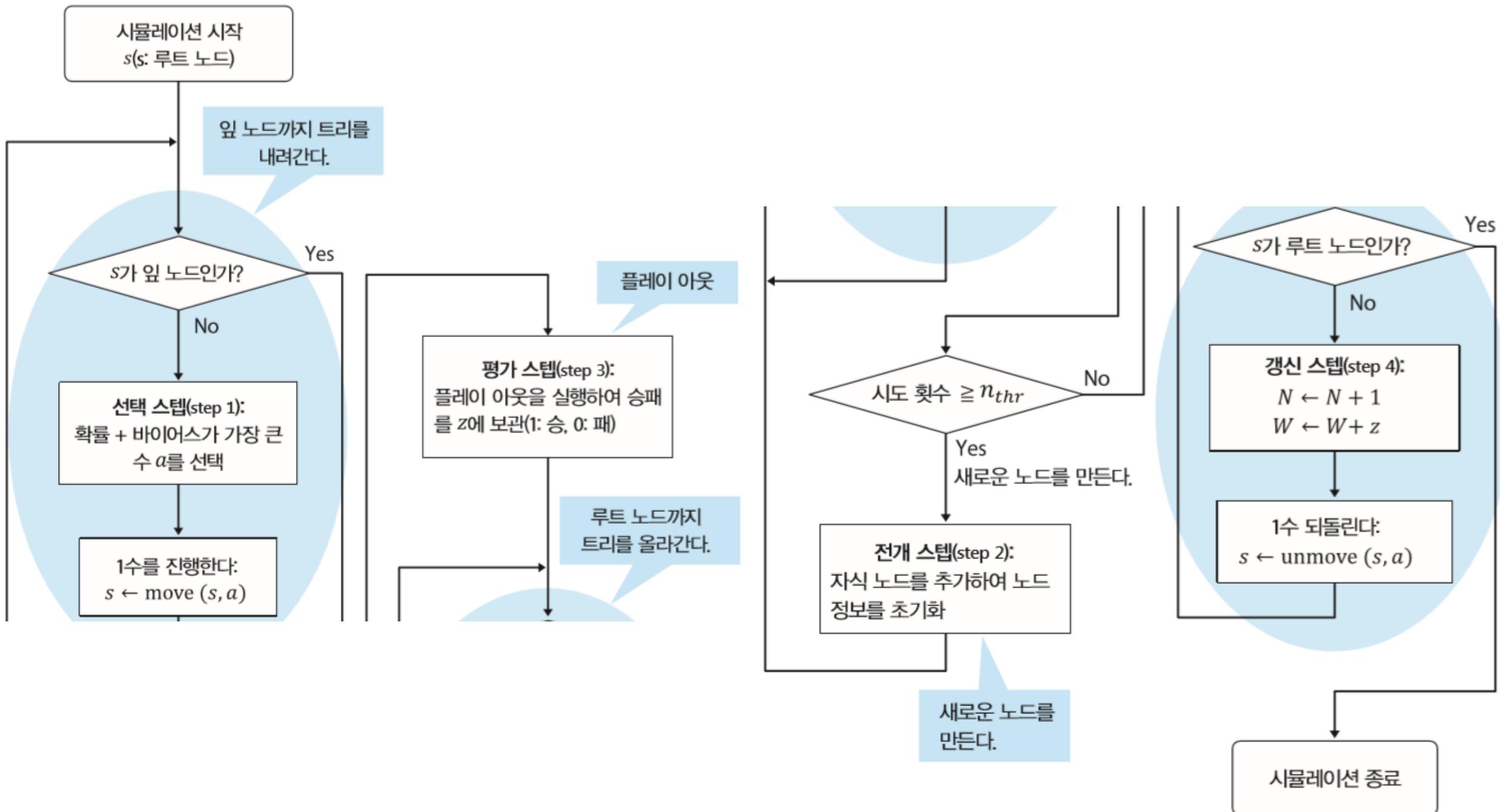
## 3. 평가(evaluation)

Step 2까지의 처리에 의해 잎 노드에 도달한 후에는 플레이 아웃을 실행한다

## 4. 갱신(backups)

플레이 아웃이 끝난 후에는 루트 노드에 이르는 모든 노드에 대해 플레이 아웃의 승패를 게임 트리에 반영한다.





# 몬테카를로 트리 탐색

- 몬테카를로 트리 탐색이란 — UCB 정책에 따라 트리를 깊게 전개하는 방법
- Step 1 ~ Step 4의 시뮬레이션을 제한 시간까지 반복하여 최종적으로 가장 시도 횟수가 많은 노드를 선택하는 것이 몬테카를로 트리 탐색의 흐름이다.

Step 1(선택): UCB1이 최대가 되는 자식 노드를 따라 트리를 내려간다.

승률: 이 국면 이후의 승률

바이어스: 탐색 횟수가 적은 경우에 커진다.

$$\text{UCB1} = (w/n) + (2 \log t/n)^{1/2}$$

$n$ : 이 국면의 총 플레이 아웃 횟수

$w$ : 이 국면의 승리 수

$t$ : 형제 노드 전체의 총 플레이 아웃 횟수

Step 2(전개): 탐색 노드 수가 일정 수( $n_{thr}$ )를 넘으면 자식 노드를 전개

Step 3(평가): 플레이 아웃을 실시

Step 4(갱신): 승패를 각 노드에 갱신하면서 트리를 올라간다.

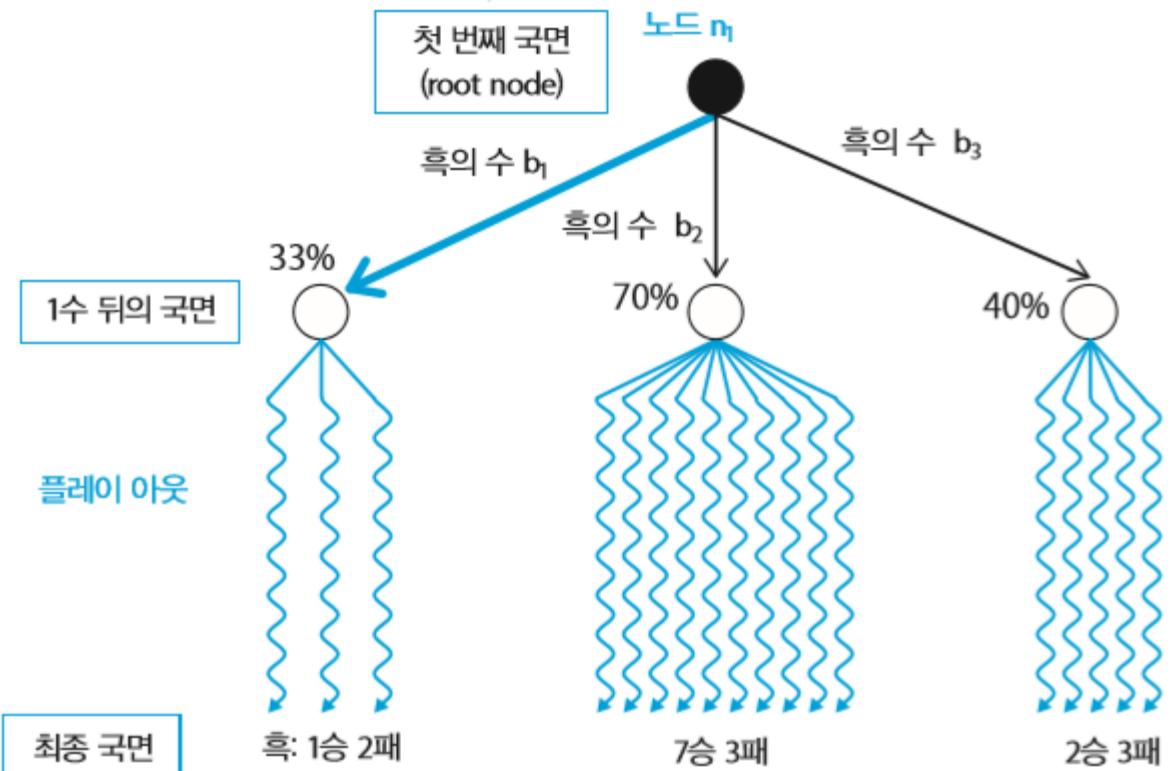
# 몬테카를로 트리 탐색

- 몬테카를로 트리 탐색(Step 1: 자식 노드의 선택 처리). UCB1이 최대의 노드를 선택하면서 루트 노드에서 잎 노드까지 추적한다

측의 수	승률	바이어스	UCB1
$b_1$	0.33	1.76	2.10
$b_2$	0.70	1.01	1.71
$b_3$	0.40	1.49	1.89

- : 흑 차례(Max 노드)
- : 백 차례(Min 노드)

Step 1: 노드  $n_1$ 에서는 승률이 높은  $b_2$ 가 아니라, UCB1이 큰 수  $b_1$ 을 선택한다.

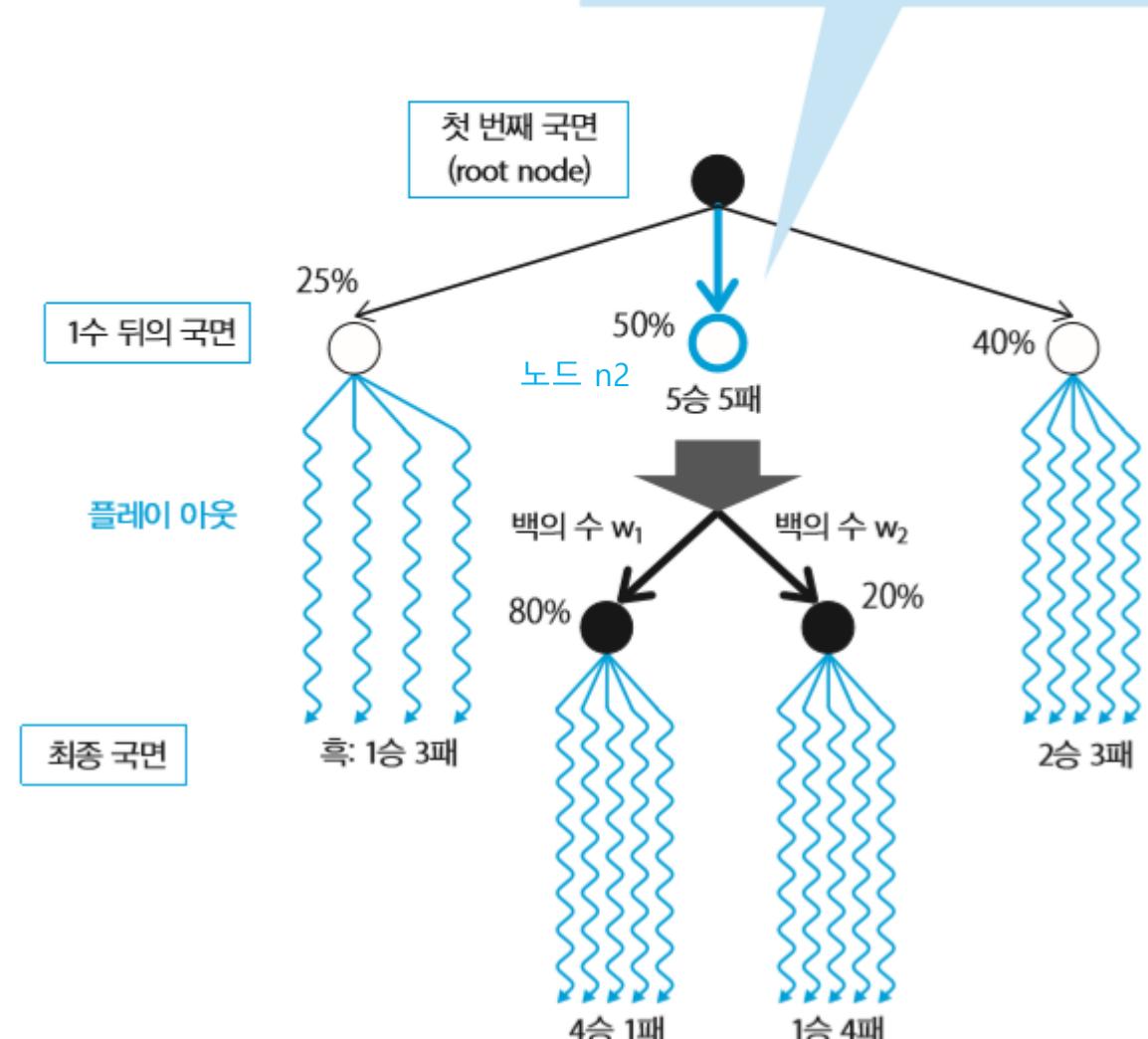


# 몬테카를로 트리 탐색

- 몬테카를로 트리 탐색  
(Step 2: 새로운 노드의 전개 처리). 앞 노드의 시도 횟수가 임계값 이상인 경우는 자식 노드로 전개하여 한층 더 트리를 내려간다

● : 흑 차례(Max 노드)  
○ : 백 차례(Min 노드)

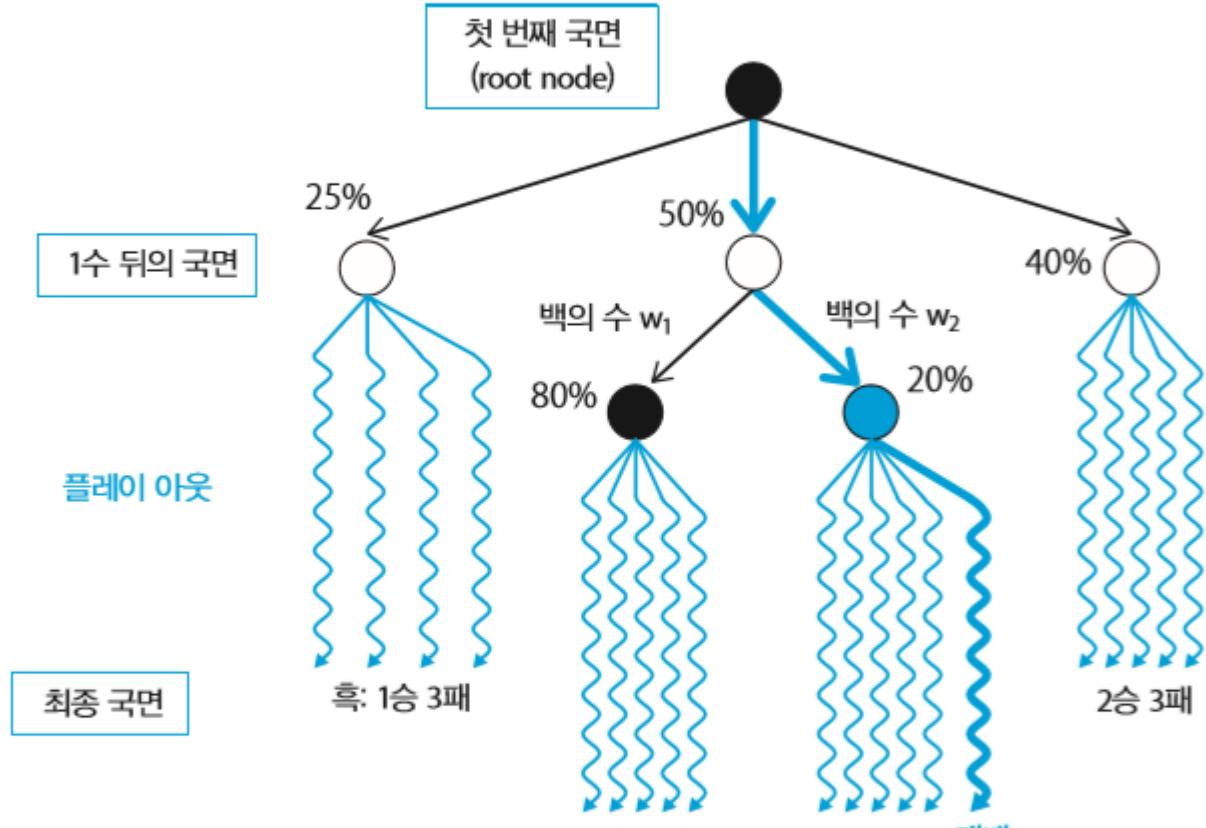
Step 2: 시도 횟수가 임계값(= 10) 이상인 노드  $n_2$ 는 자식 노드를 전개



# 몬테카를로 트리 탐색

- 몬테카를로 트리 탐색(Step 3: 자식 노드의 평가 처리). 잎 노드에서 플레이 아웃을 실행하여 승부의 결과를 얻는다
- 플레이아웃에서 각 국면에서는 좋은 수를 우선적으로 생성하는 확률 모델(예를 들면, 롤 아웃 정책)에 따라 수를 생성한다

● : 흑 차례(Max 노드)  
○ : 백 차례(Min 노드)



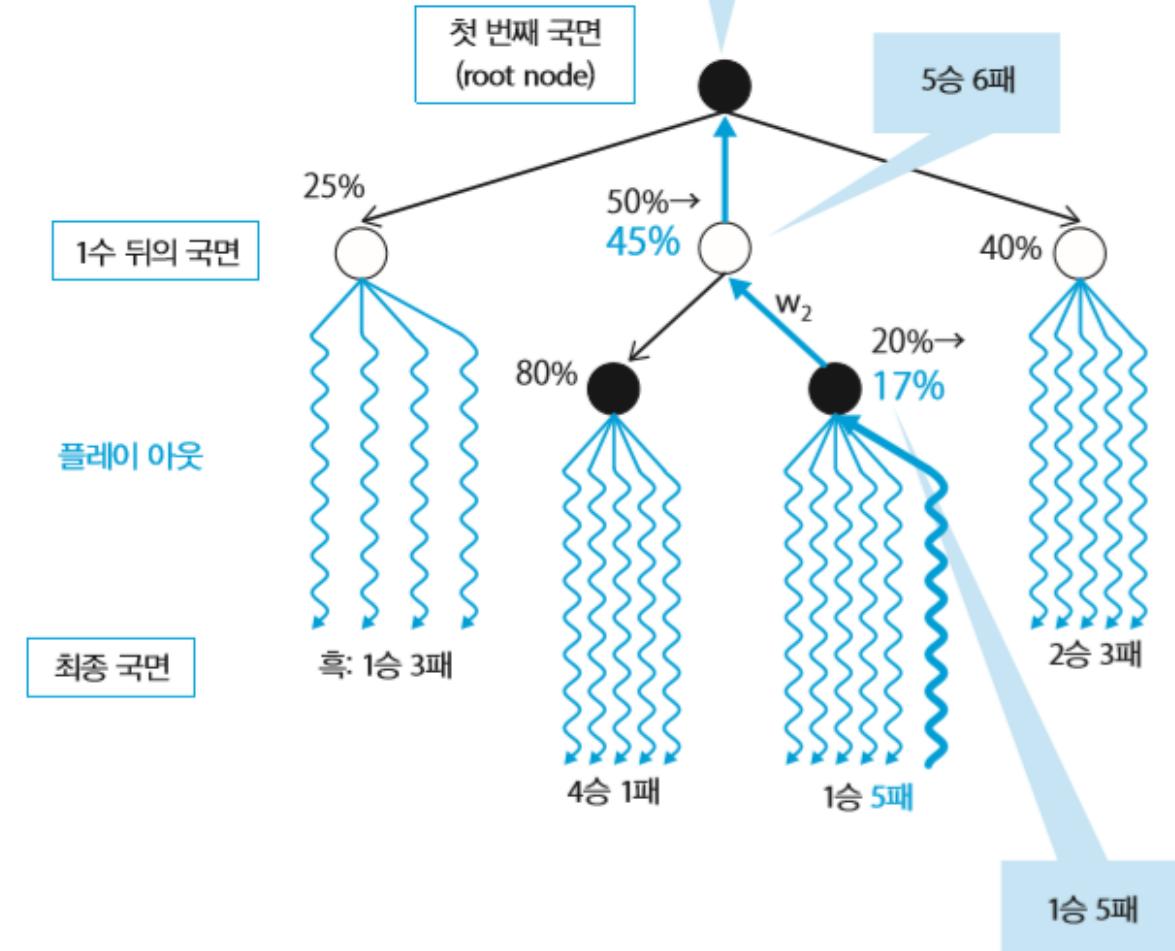
Step 3: 플레이 아웃을 실행하고 승부의 결과를 얻는다.

# 몬테카를로 트리 탐색

- 몬테카를로 트리 탐색  
(Step 4: 탐색 결과의 갱신 처리). 잎 노드의 플레이 아웃 결과(이번에는 '패배')를 트리를 오르면서 각 노드에 기록한다

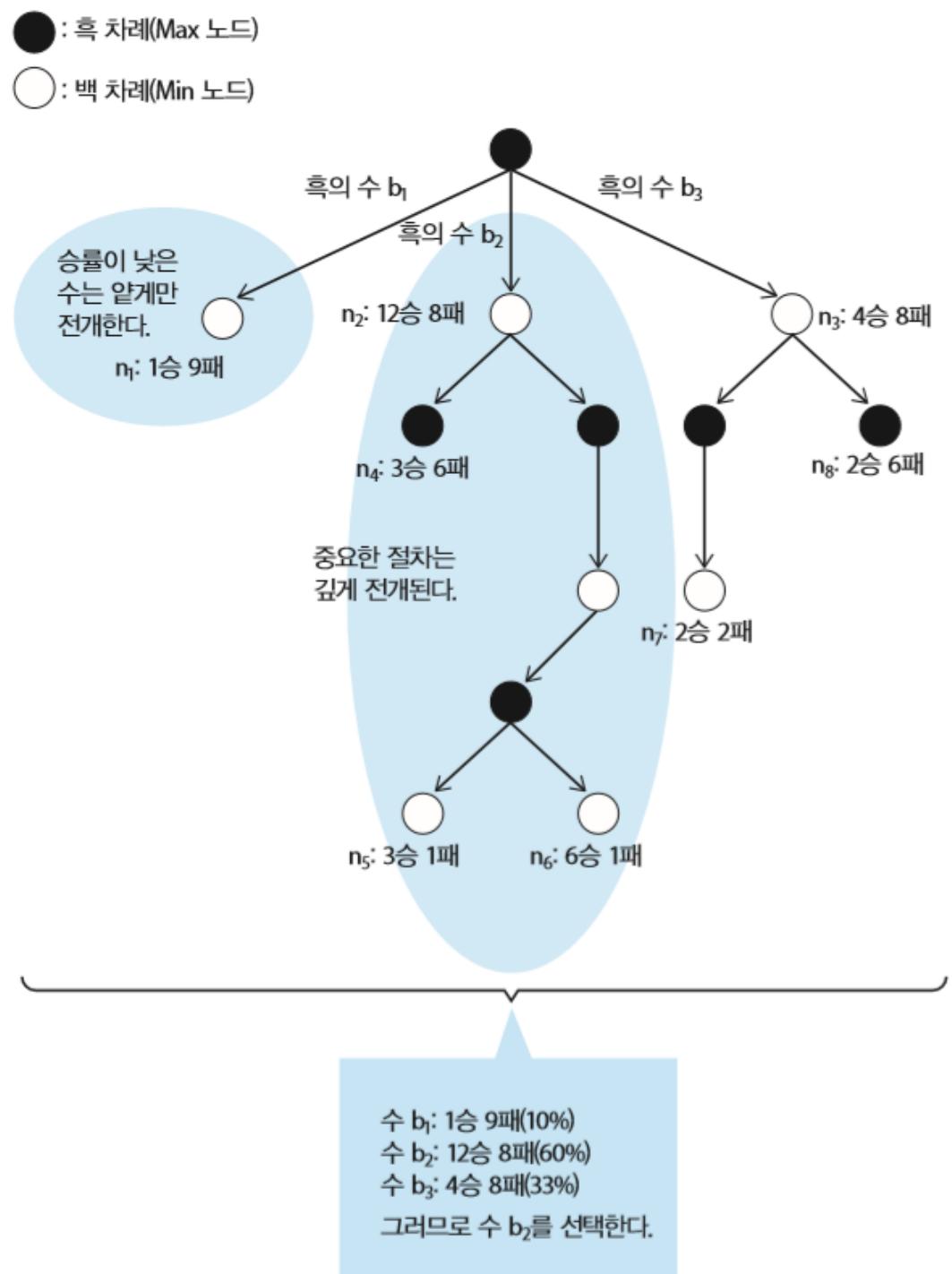
● : 흑 차례(Max 노드)  
○ : 백 차례(Min 노드)

Step 4: 순서대로 트리를 올라가 결과를 노드에 기록한다(8승 12패).



# 몬테카를로 트리 탐색

- 몬테카를로 트리 탐색 (최종적인 수의 선택 처리). 루트 노드의 자식 노드 중에서 가장 시도 횟수가 많은 수  $b_2$ 를 선택 한다



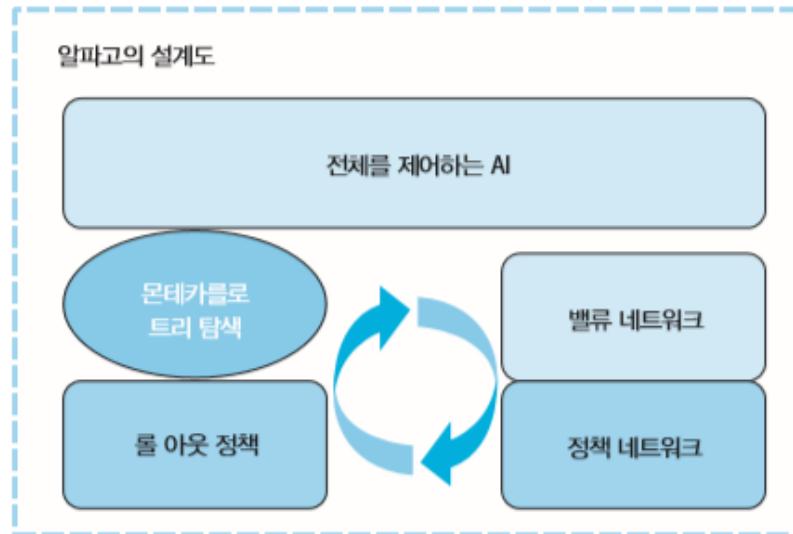
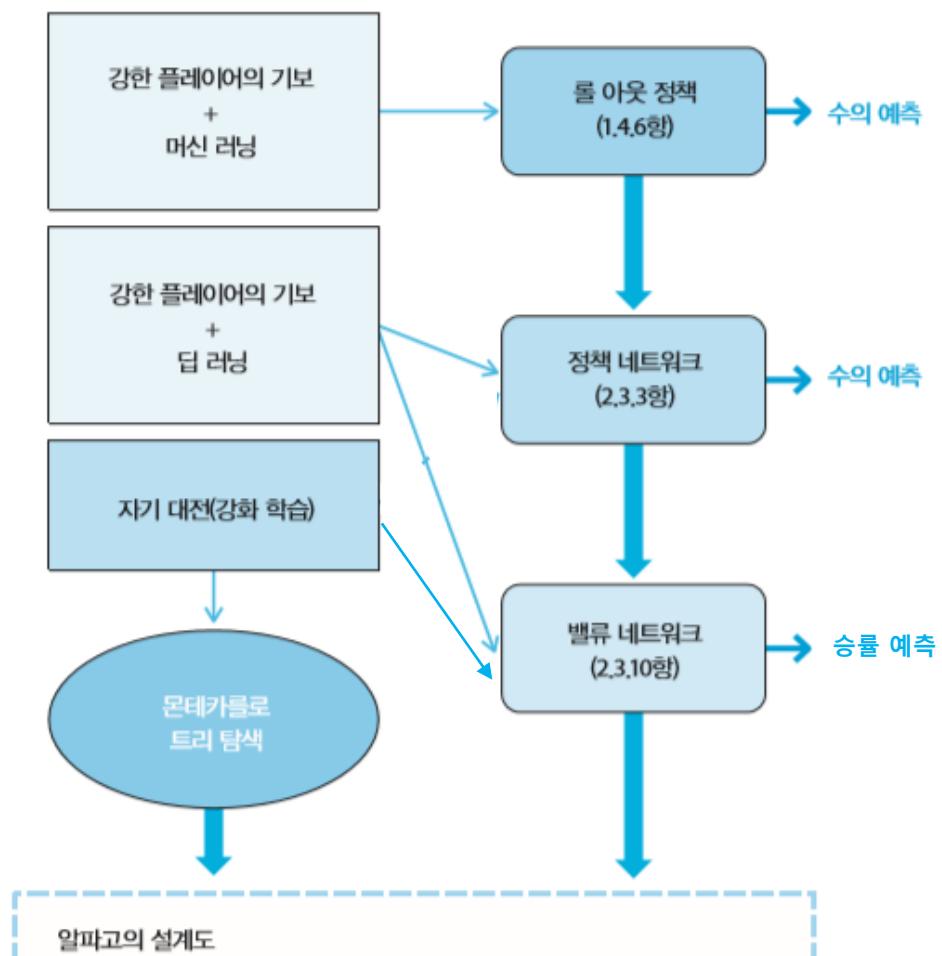
## 알파고의 완성

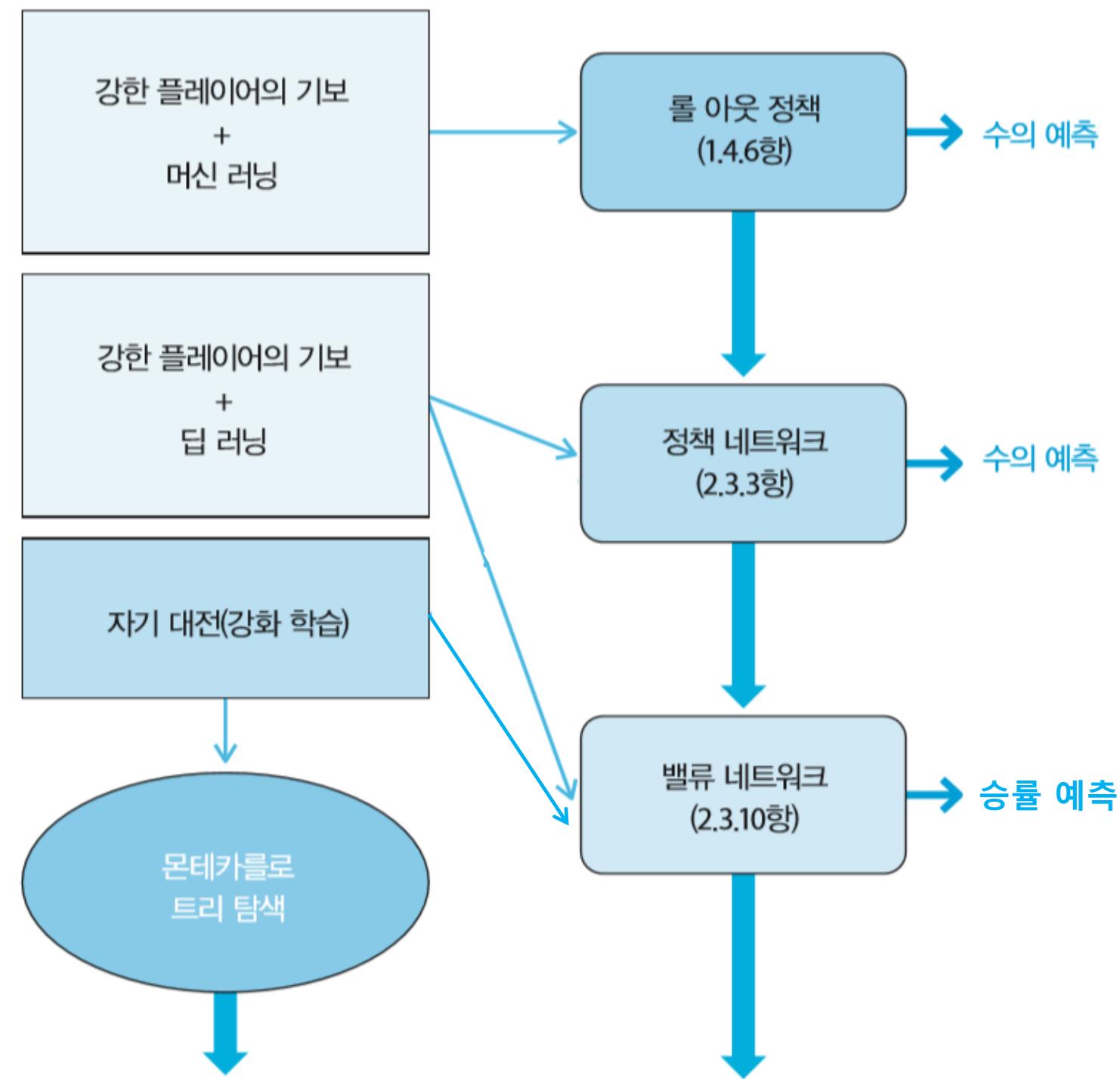
직관력의 딥 러닝, 경험으로 배우는 강화 학습, 예측을 잘하는 탐색, 세 개의 도구를 잘 조합  
함으로써 알파고가 완성된다.

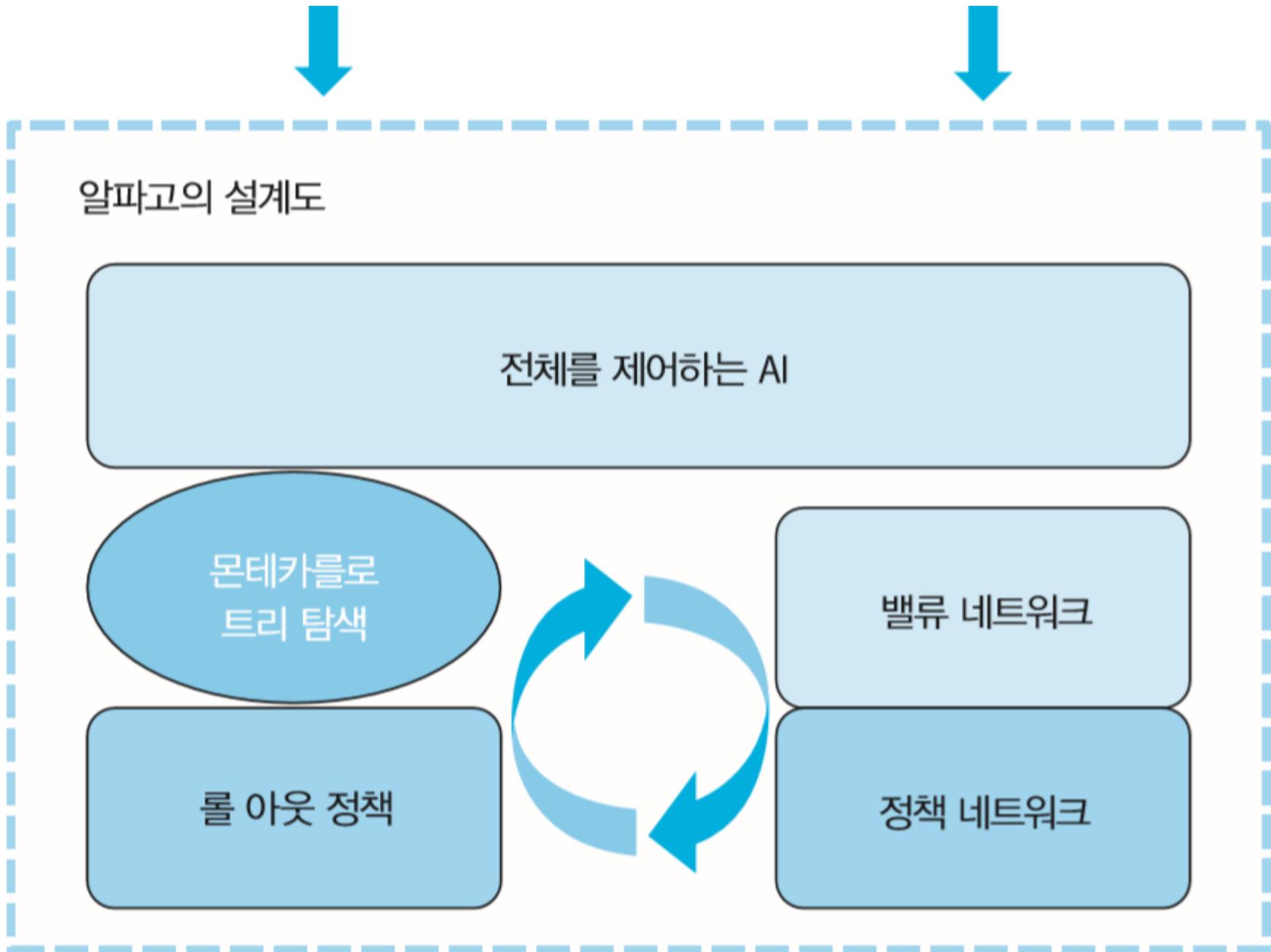
어떻게 하면 SL 정책 네트워크와 밸류 네트워크를 몬테카를로 트리 탐색에 통합할지, 엄청  
난 CPU와 GPU를 어떻게 활용할지 지금까지의 기술의 정수를 모은 알파고 힘의 비밀에 다  
가선다.

# 알파고의 설계도

- 세 가지 정책의 과정과 알파고의 설계도

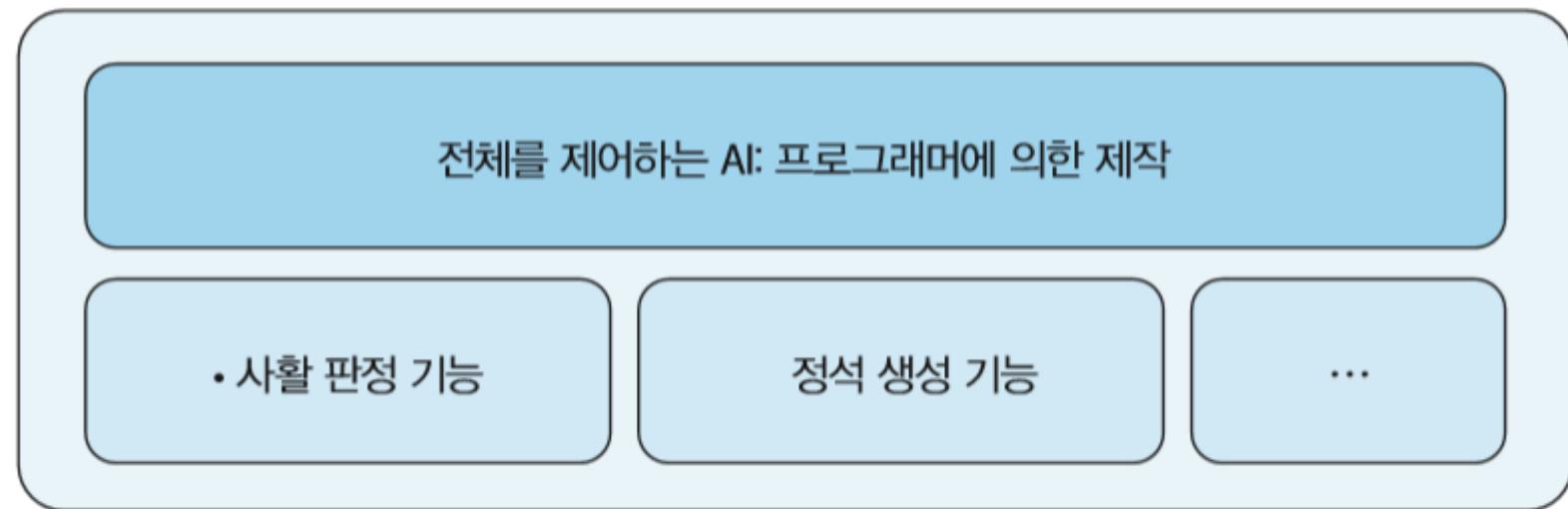






# 바둑 AI의 진화와 '전 체를 제어 하는 AI' 역 할의 변천

(a) 몬테카를로 트리 탐색 이전의 시대(~2006)

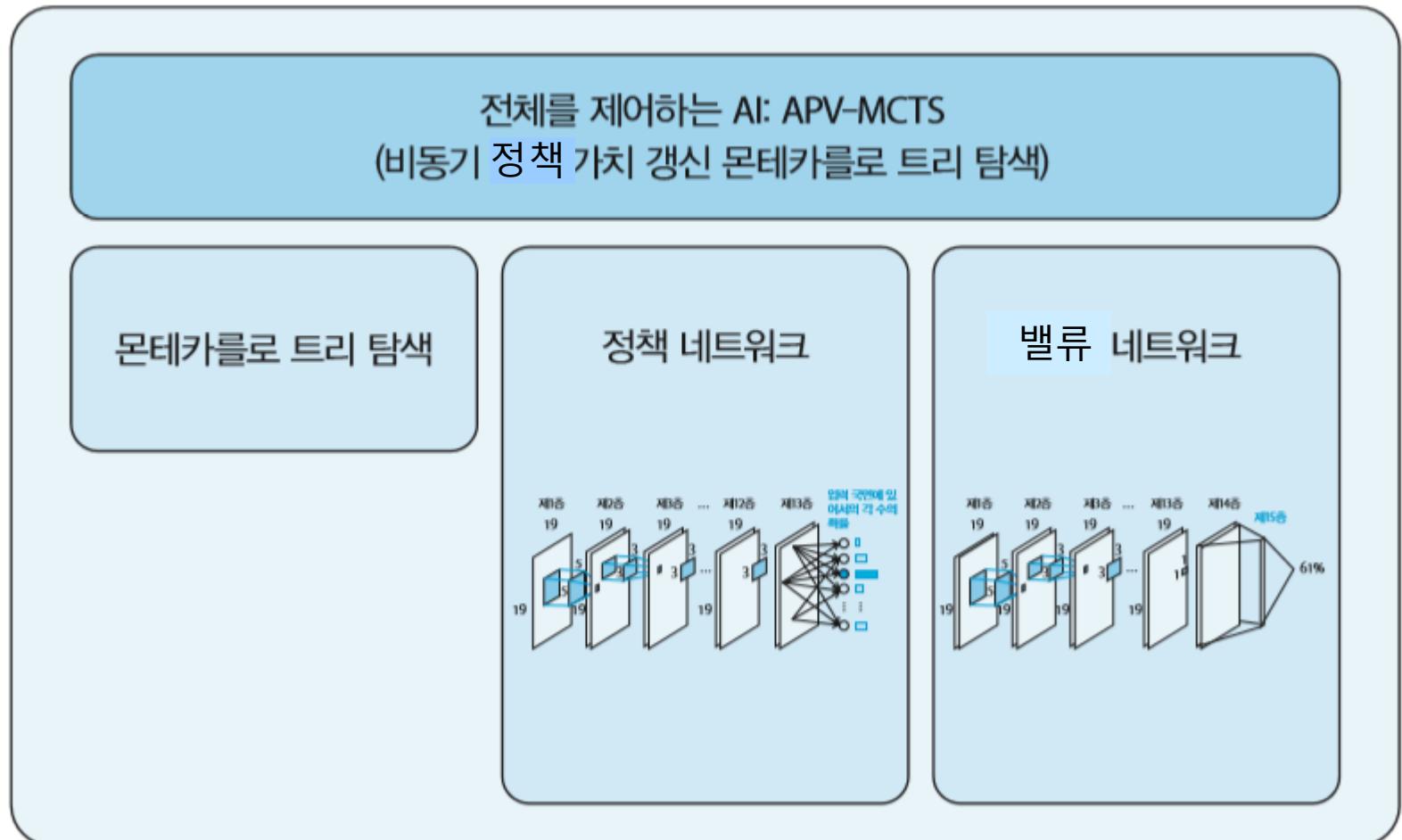


(b) 몬테카를로 트리 탐색의 시대(2006~2016)



# 바둑 AI의 진화와 '전 체를 제어 하는 AI' 역 할의 변천

(c) 알파고(2016~)



# 비동기 정책 가치 갱신 몬테카를로 트리 탐색

- APV-MCTS (Asynchronous Policy and Value MCTS)

# 세 가지 정책의 비교

## (a) 세 가지 정책

- **롤 아웃 정책:**

로지스틱 회귀에 의해 고속으로 각 후보 수를 선택하는 확률을 부여한다.  
→ [플레이 아웃에 이용](#)

- **정책 네트워크:**

CNN에 의해 각 후보 수를 선택하는 확률을 부여한다.  
→ 노드의 선택 및 전개 처리에 이용

- **밸류 네트워크**

} → [노드의 선택 및 전개 처리에 이용](#)

# 세 가지 정책의 비교

- 롤 아웃 정책은 고속이지만, 일치율이 낮다. 한편, 정책 네트워크, 밸류 네트워크는 계산 시간이 걸린다

(b) 각 정책의 특징

	롤 아웃 정책	정책 네트워크	밸류 네트워크
이용하는 모델	로지스틱 회귀	13층의 CNN	15층의 CNN
1국면 평가에 걸리는 시간	2마이크로초	5밀리초	5밀리초(추정)
1회의 플레이 아웃에 걸리는 시간(종국까지의 수를 200수로 가정)	0.4밀리초	1.0초	-
1초간에 플레이 아웃할 수 있는 횟수	약 2,500회	약 1회	-
일치율	24%	57%	-

# 기존의 몬테카를로 트리 탐색에 대한 알파고의 개선점

- 바이어스 계산에서 SL 정책 네트워크의 이용
- 플레이 아웃의 승률과 밸류 네트워크를 병용하는 것에 의한 승률 평가의 개선
- 다수의 CPU와 GPU의 이용에 의한 고속화

포인트 1: 바이어스 평가의 개선

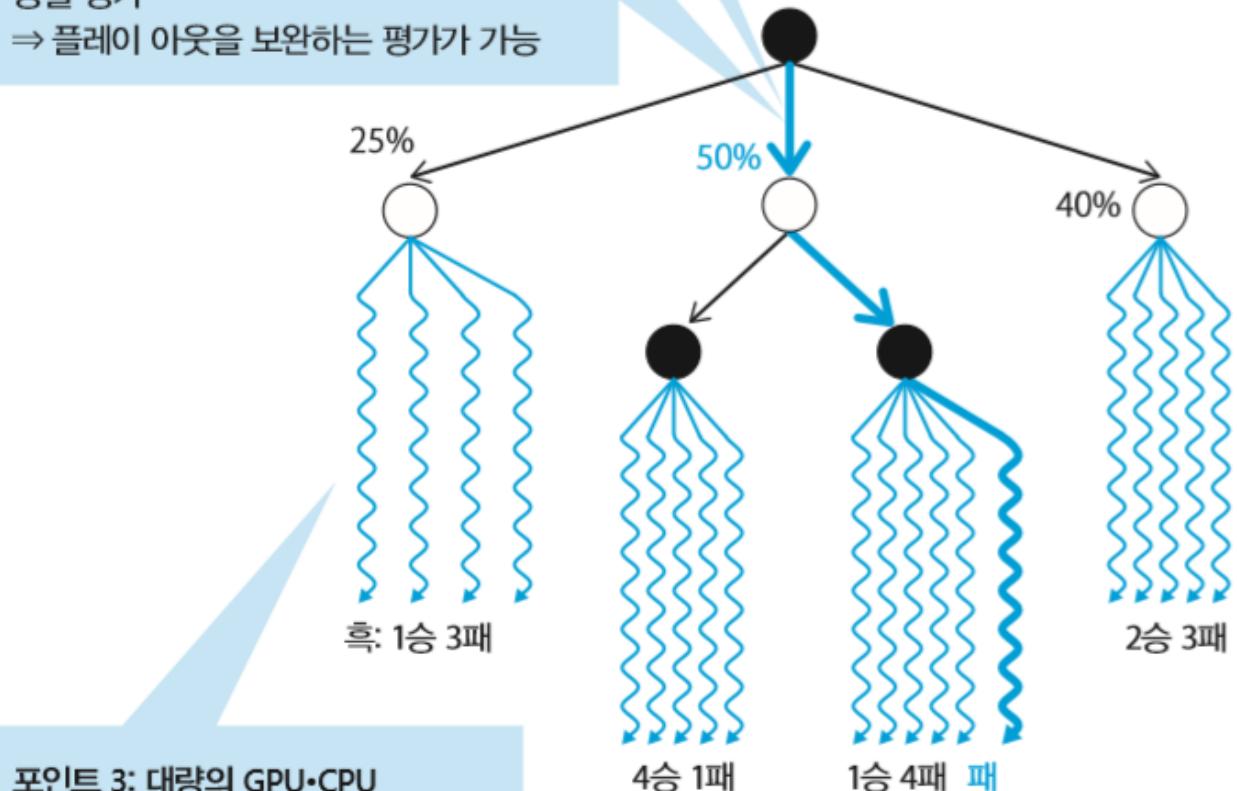
바이어스 + SL 정책 네트워크에 의한 수의 사전 확률  
⇒ 있음직한 수순을 보다 더 깊게 전개할 수 있다.

포인트 2: 승률 평가의 개선:

플레이 아웃 승률 + 밸류 네트워크에 의한 승률 평가

⇒ 플레이 아웃을 보완하는 평가가 가능

기존의 몬테카를로 트리 탐색:  
 $UCB1 = \text{승률} + \text{바이어스가 높은 수를 선택}$



# 비동기 정책 가치 갱신 몬테카를로 트리 탐색(APV- MCTS)의 상세

- 일반 몬테카를로 트리 탐색과 비교하면 정책 네트워크와 밸류 네트워크의 정보를 이용하여 선택 처리의 정확도를 높임

Step 1(선택): 국면  $s$ 에서  $Q(s, a) + u(s, a)$ 가 최대가 되는 자식 노드  $a$ 를 따라 트리를 내려온다.

승률 :

몬테카를로 트리에 의한 승률과 밸류 네트워크에 의한 승률을 통합

바이어스 :

기존의 바이어스에 더하여 SL 정책 네트워크에 의한 수의 사전 확률을 고려

밸류 네트워크에 의한 승률  
몬테카를로 트리 탐색에  
의한 승률

$$Q(s, a) = (1 - \lambda) \frac{W_v(s, a)}{N_v(s, a)} + \lambda \frac{W_r(s, a)}{N_r(s, a)}$$

$$u(s, a) = P(s, a) \frac{\sqrt{\sum_b N_r(s, b)}}{1 + N_r(s, a)}$$

SL 정책 네트워크에 의한 사전  
확률

탐색 횟수가 적은 경우에 커진다.  
→ 기존의 바이어스항에 대응

# 비동기 정책 가치 갱신 몬테카를로 트리 탐색(APV-MCTS)의 상세

**Step 2(전개): 탐색 노드 수가 일정 수를 넘으면 자식 노드를 전개**  
+ 새 노드를 만들 때 정책 네트워크 + 밸류 네트워크의 값을 계산

여러 CPU와 GPU에서 동작시키는 경우에는 대기 시간이 발생하지 않도록 하기 위해 CPU에 의한 플레이 아웃 속도와 GPU에 의한 정책 네트워크, 밸류 네트워크의 처리 속도를 맞출 필요가 있다.

그래서 초기값이 40인 새로운 노드 생성의 임계값  $n_{thr}$ 을 동적으로 조정하고 있다.

- ☞ GPU의 대기열이 길어지는 경우에는  $n_{thr}$ 을 늘림으로써 새로운 노드의 생성 속도를 떨어뜨린다.
- ☞ GPU가 대기 상태가 된 경우에는  $n_{thr}$ 을 줄임으로써 새로운 노드의 생성 속도를 올린다.

# 비동기 정책 가치 갱신 몬테카를로 트리 탐색(APV-MCTS)의 상세

Step 3(평가): 플레이 아웃을 실시

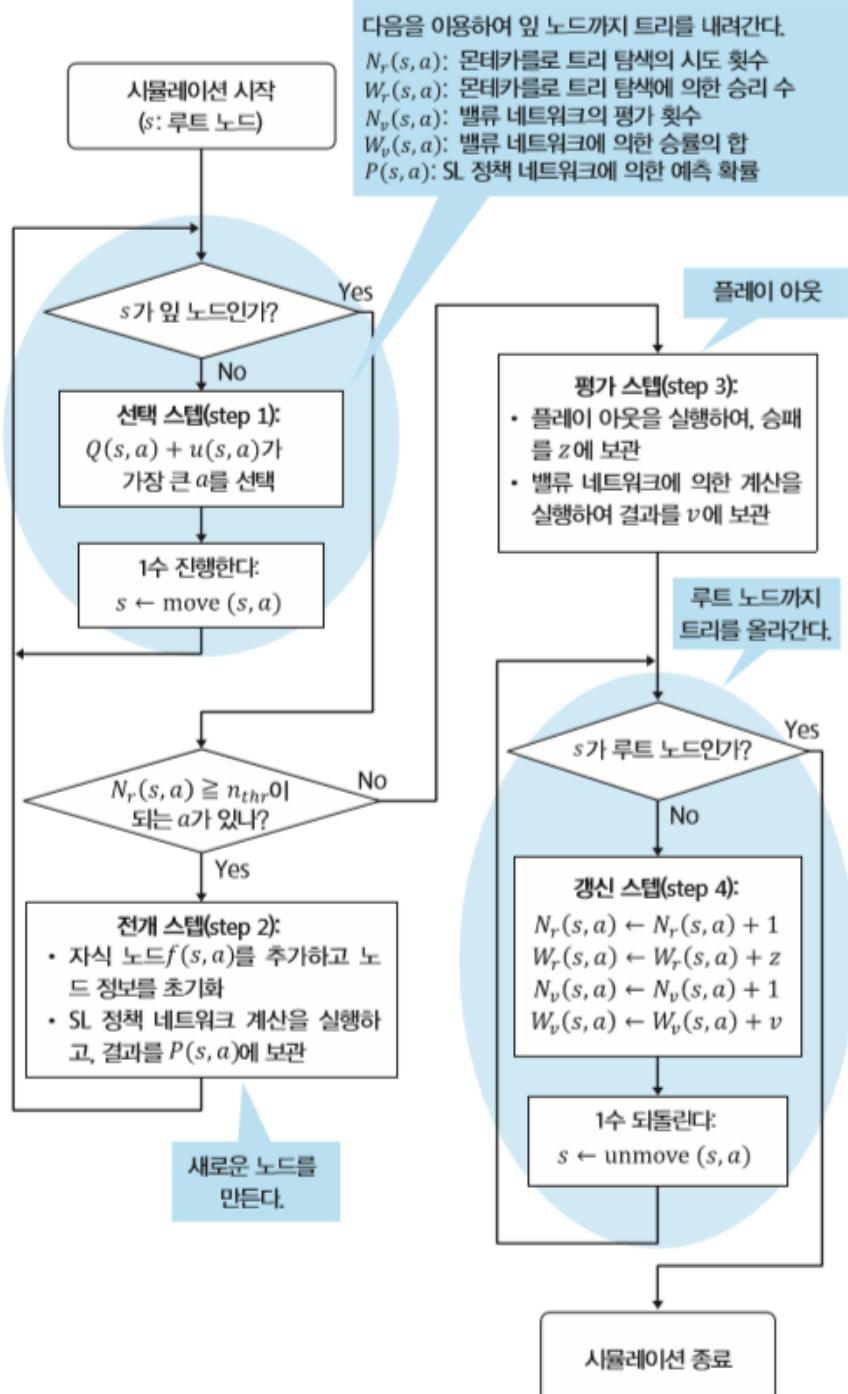
+ 가상 손실(Virtual loss)을 이용하여 플레이 아웃 시작 노드를 조정

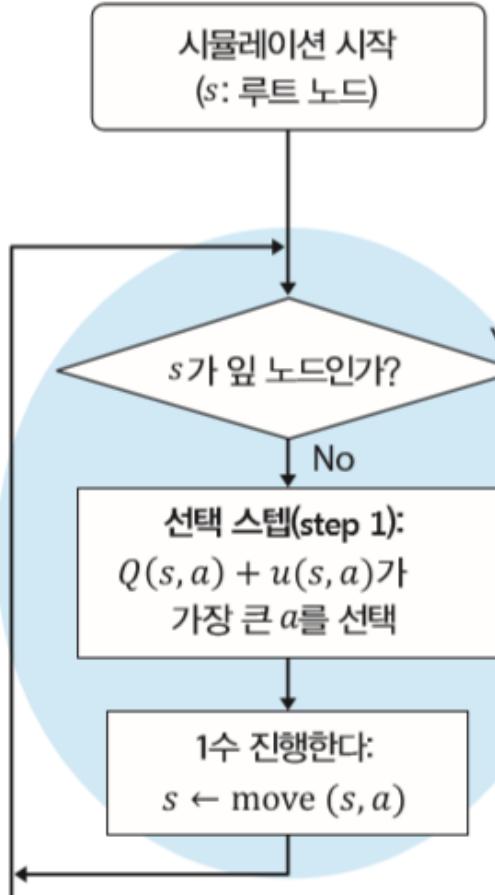
Step 4(기록): 승패를 각 노드에 기록하면서 트리를 올라감

+ 비동기로 갱신할 수 있도록 로크리스 해시를 이용

# APV-MCTS의 플로 차트

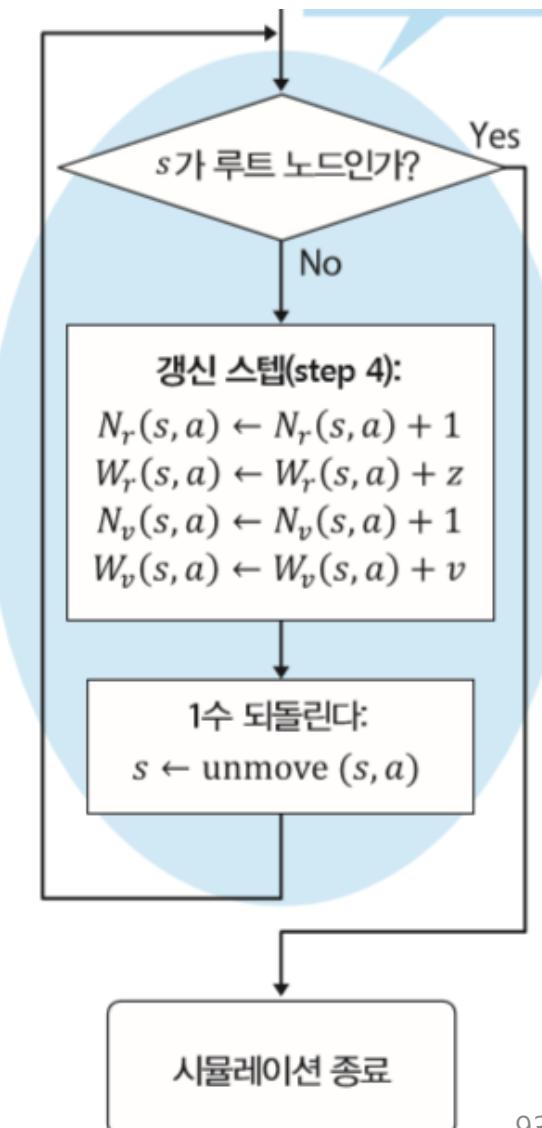
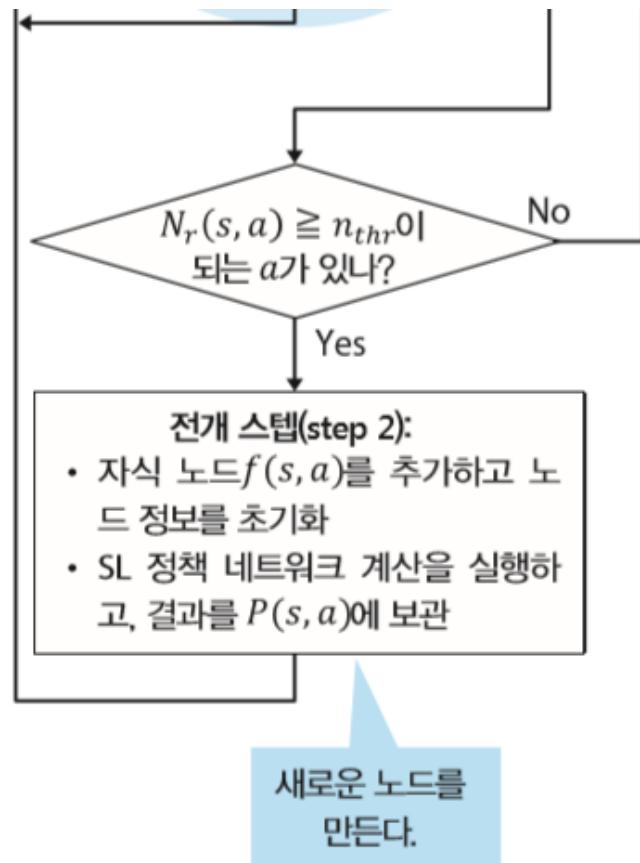
- 일반적인 몬테카를로 트리 탐색의 플로 차트와 비교하면 플레이 아웃의 승률과 밸류 네트워크의 승률을 병용하고 있는 점이 나타나고 있다





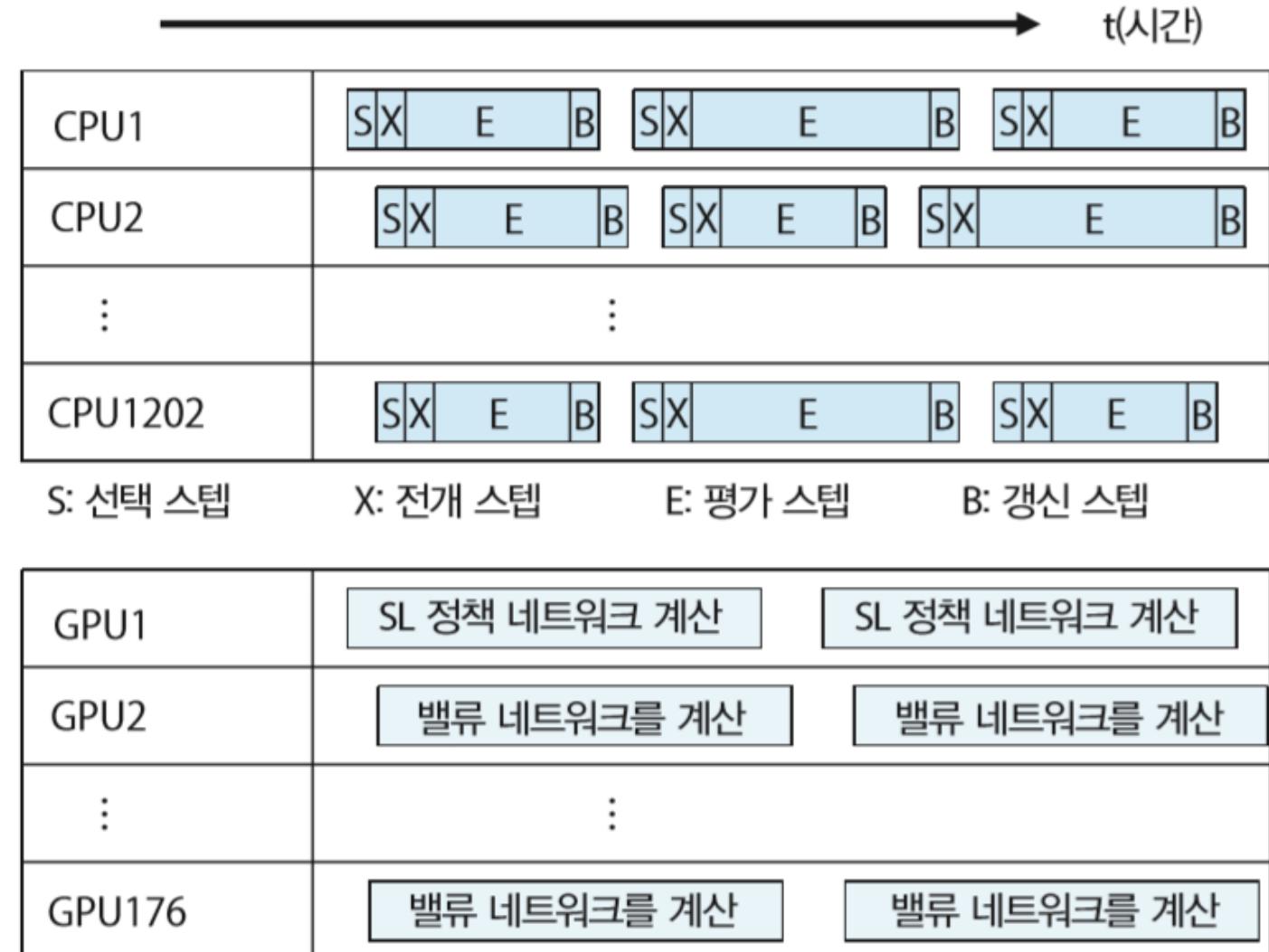
다음을 이용하여 잎 노드까지 트리를 내려간다.

$N_r(s, a)$ : 몬테카를로 트리 탐색의 시도 횟수  
 $W_r(s, a)$ : 몬테카를로 트리 탐색에 의한 승리 수  
 $N_v(s, a)$ : 밸류 네트워크의 평가 횟수  
 $W_v(s, a)$ : 밸류 네트워크에 의한 승률의 합  
 $P(s, a)$ : SL 정책 네트워크에 의한 예측 확률



# 대량 CPU와 GPU에 의한 병렬 탐색

- CPU는 선택, 전개, 평가, 갱신으로 구성된 시뮬레이션 처리를 실행한다.
- GPU는 전개 처리 중의 SL 정책 네트워크와 밸류 네트워크의 계산 처리를 실행한다.
- 1,202개의 CPU와 176개의 GPU에 의한 병렬 처리를 나타낸다



# 로크리스 해시

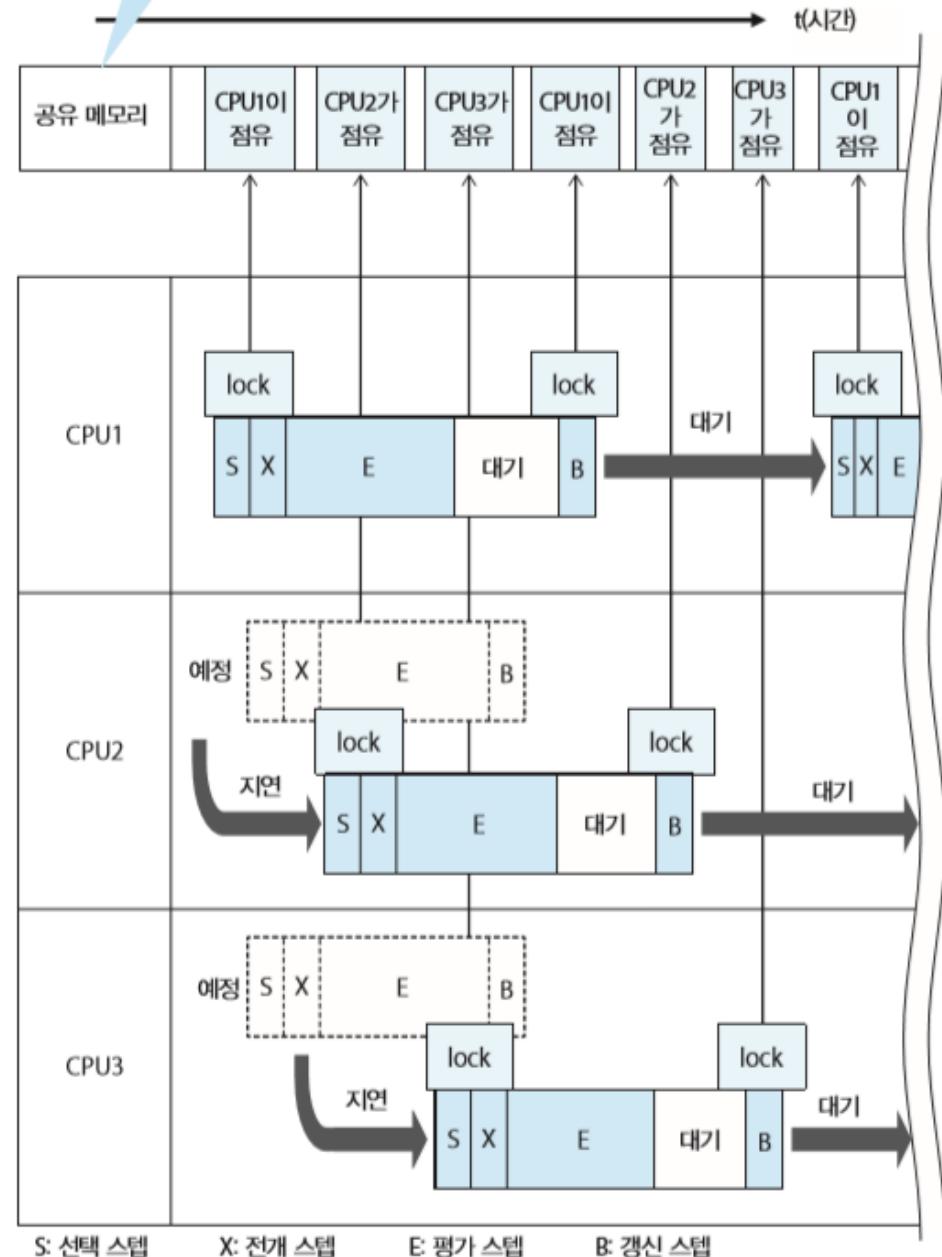
- 공유 메모리를 잠금에 의해 배타 제어하는 방법

➤ 어떤 CPU가 메모리를 잠그는 경우 그 동안 다른 CPU는 공유 메모리에 액세스할 수 없어 대기한다. 따라서 다른 CPU의 처리 지연, 불필요한 대기 시간이 발생한다.

- 로크리스 해시

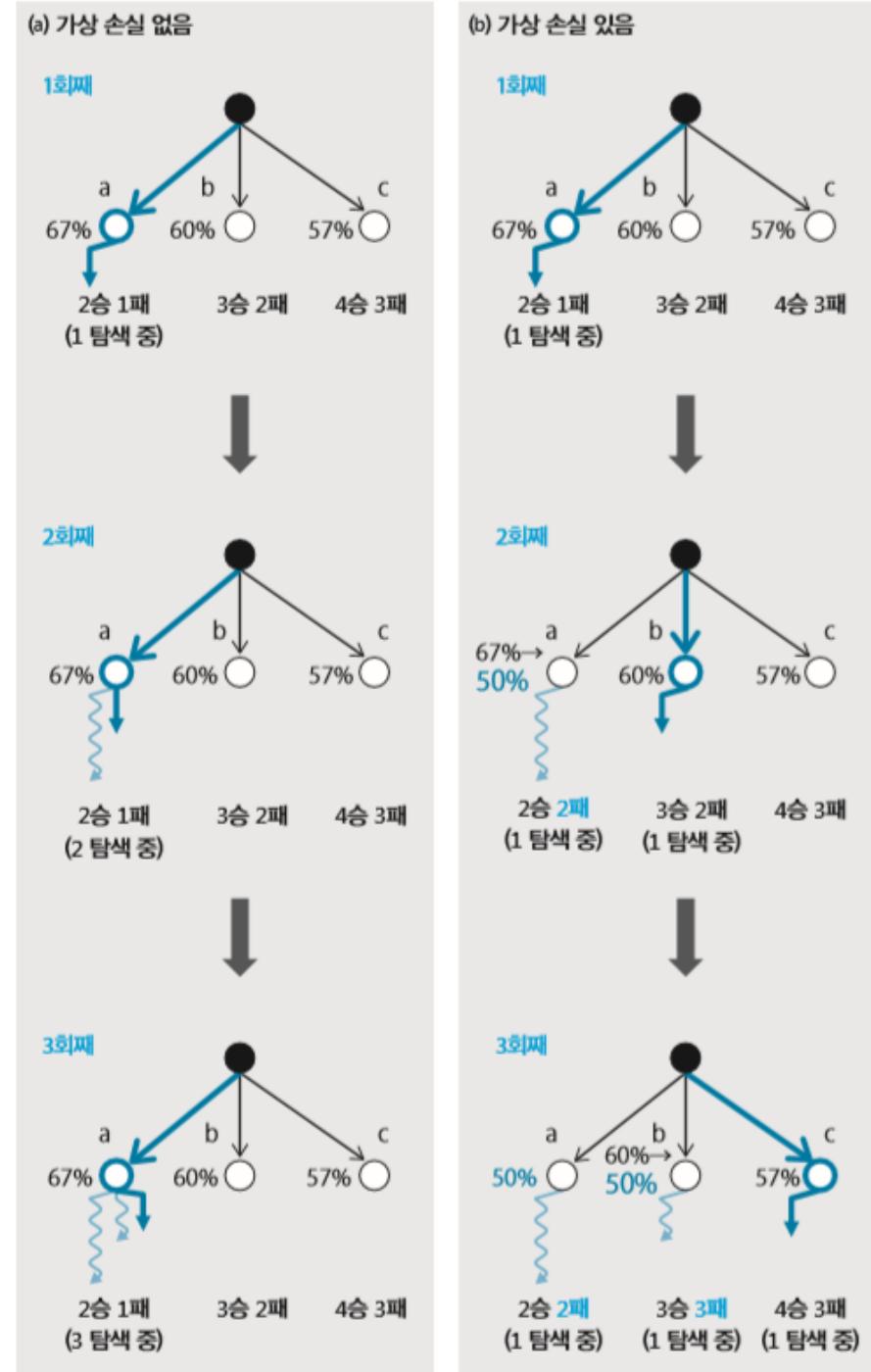
➤ 해시 자료구조를 이용해 잠금을 걸지 않고 공유 메모리를 읽고 쓰는 방법  
➤ 데이터 액세스를 최소(atomic) 단위로 만 실시함으로써 어떤 CPU가 갱신 중인 데이터는 다른 CPU에게는 보이지 않도록 하여, 공유 메모리 액세스 시 데이터 손실 위험 방지

몬테카를로 트리의 탐색 결과  
가 보관되어 있다.



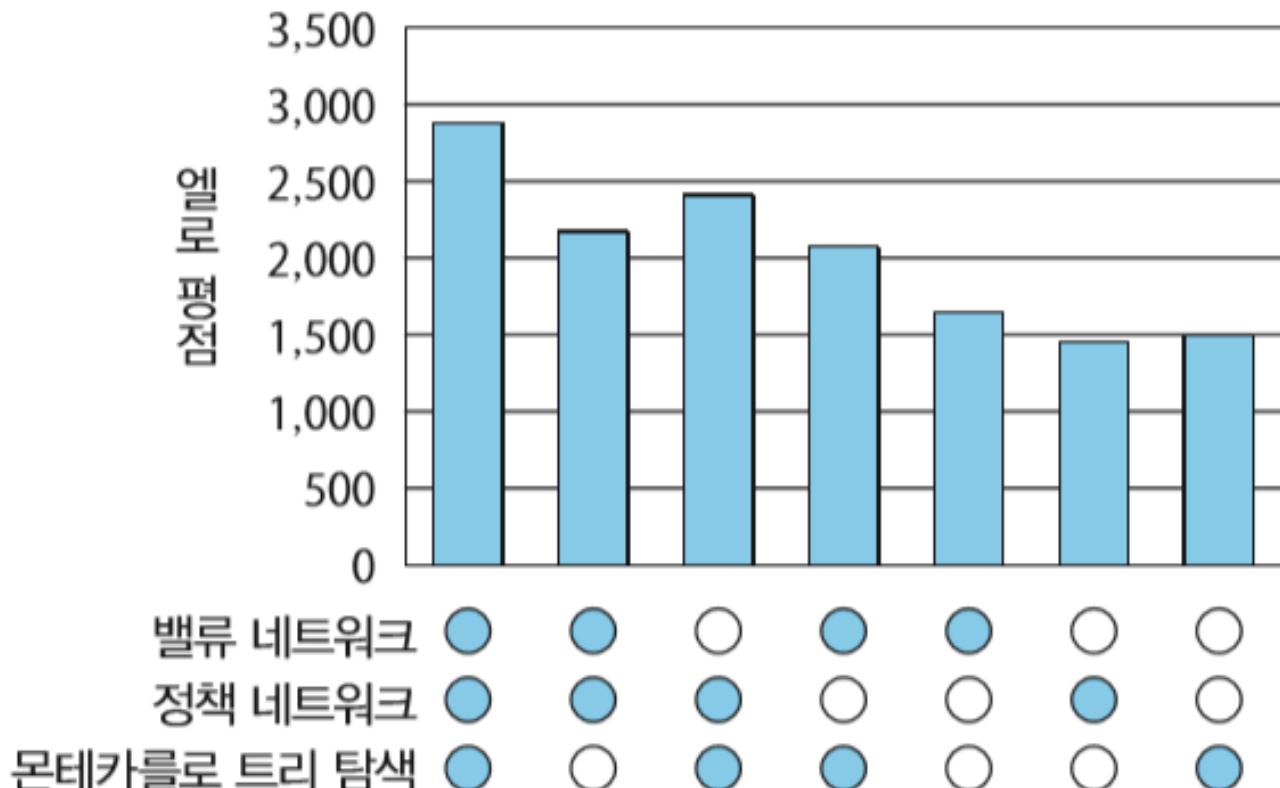
# 가상 손실

- 여러 개의 CPU가 시뮬레이션을 실시할 경우에 일정 시간 동안 몬테카를로 트리 정보가 갱신되지 않은 경우가 있다.
- (a)와 같이 순차적으로 CPU가 시뮬레이션을 시작해도 몬테카를로 트리의 정보가 통일하므로 같은 노드에 도달해 같은 플레이 아웃을 시작한다.
- 결과적으로 시뮬레이션을 시작하는 노드가 특정 노드에 집중되어 버린다는 문제가 있다.
- 가상 손실에서는 어떤 CPU가 플레이 아웃을 시작할 경우 이 시작 노드에 가상으로 패배 주를  $n_{v1}$  개로 하는 기법이다.
- (b)는  $n_{v1}=1$ 인 경우를 보여준다. 가상 손실의 효과로 탐색 노드를 흩어 놓을 수 있다.
- 플레이 아웃 종료 시에는 가상 손실 만큼 원래대로 되돌림으로써 올바른 승패로 되돌아간다.



# 알파고의 엘로 평점

- 정책 네트워크, 밸류 네트워크, 몬테카를로 트리 탐색의 조합에 의한 엘로 평점의 차이
  - ▶ 컴퓨터 1대(48CPU, 8GPU)로 2초간 탐색
  - ▶ 여러 시스템을 클러스터로 구성했을 경우 3,150점 이상으로 판 후 이 2단에게 이기는 수준에 도달



## 알파고에서 알파고 제로로

2017년 10월 19일, 마침내 알파고의 전모가 밝혀졌다. 새로운 《네이처》 논문 〈Mastering the game of Go without human knowledge(인간의 지식 없이 바둑을 연구하기)〉가 발표되었다.

새로운 논문에서는 지금까지 비밀의 베일에 싸여 있던 강화 학습 기법을 중심으로 제로부터 바둑 AI를 배울 수 있는 알파고 제로 기술에 대해 설명되어 있다. 논문은 '단 3일에' '제로부터' '1대의 기계로도 동작하는' 최강 바둑 AI가 생긴 것을 어필한다. 알파고 제로는 지금까지 설명 한 기존 버전의 알파고를 바탕으로 딥 러닝, 탐색, 강화 학습의 각 기술을 개량하여 만든 것이며, 결코 어렵지 않다. 기존 버전의 알파고에서 알파고 제로에 이르는 기술을 이 장에서 설명 한다.

# 기존 버전 알파고와 비교해서 알파고 제 로의 세 가 지 개선점

## 포인트 1:

### 딥 러닝의 개선

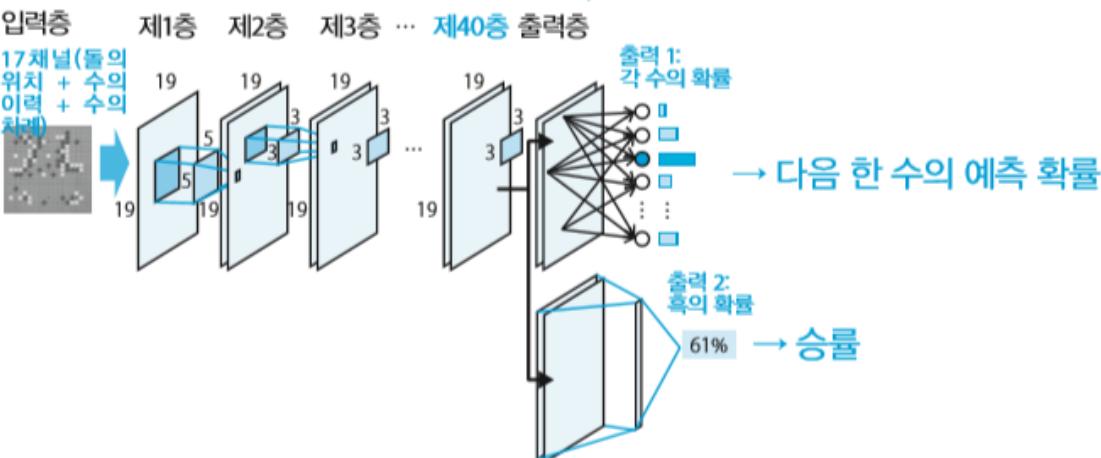
- 정책 네트워크(수의 예측)와 밸류 네트워크(승률 예측)가 하나의 네트워크에 통합
- 잔차 블록을 활용한 40층 이상의 네트워크  
→ 6.2절

## 포인트 3:

### 강화 학습의 개선

- 네트워크의 파라미터를 정교한 기법에 의해 획득  
→ 6.4절

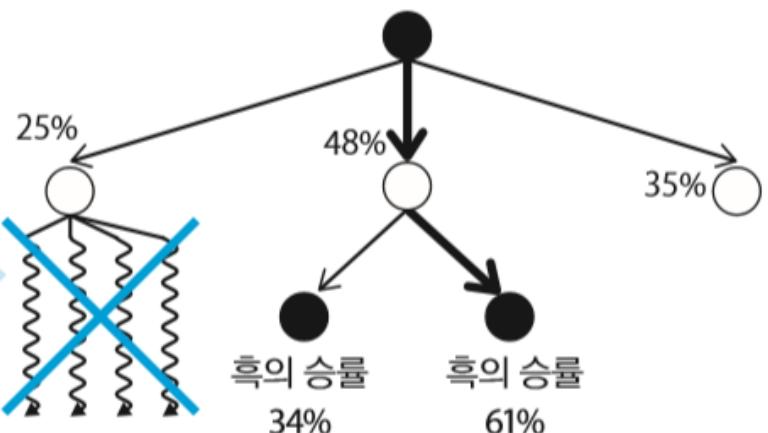
## 알파고 제로의 네트워크 모델



## 포인트 2:

### 몬테카를로 트리 탐색의 개선

- 기존 버전에서 실시하던 플레이 아웃이 불필요해짐
- 승률은 밸류 네트워크만으로 계산  
→ 6.3절

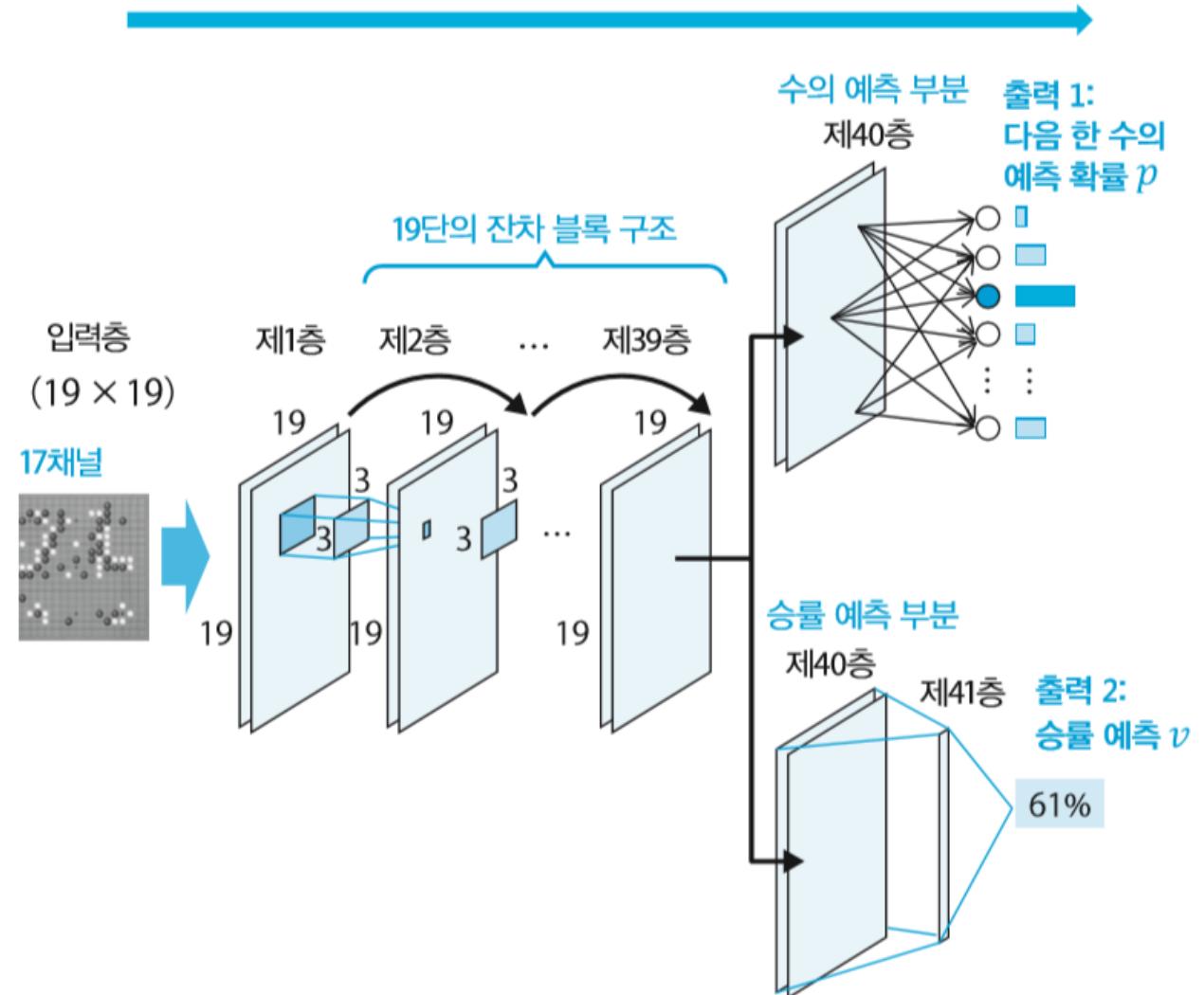


# 듀얼 네트워크의 구조

- 기존 버전 알파고의 정책 네트워크와 밸류 네트워크를 통합한 구조로 되어 있다

## • 듀얼 네트워크의 구조

- 입력은 17채널(0~7수 앞의 흑돌/백돌의 위치, 수의 차례)
- 총 40층 이상
  - 각 층은 기본적으로  $3 \times 3$ 의 컨볼루션층 + 배치 정규화 + ReLU로 이루어짐
  - 제2 ~ 39층까지는 솟컷을 갖는 잔차 네트워크(ResNet)
- 제40층에서는 다음 한 수의 예측 부분과 승률 예측 부분으로 나뉜다.



# 입력 채널

(a) 기존 버전 알파고에서 사용된 입력 48 채널과

(b) 알파고 제로 듀얼 네트워크의 입력 17채널의 내역.

(b)가 (a) 보다 더 간단하다

(a) 기존 버전 알파고에서 사용된 입력 48채널

입력 채널의 종류	채널 수
흑돌의 위치	1
백돌의 위치	1
빈 곳의 위치	1
$k$ 수 앞에 둔 위치( $k = 1\sim 8$ )	8
돌이 있는 경우의 해당 연의 호흡점 수( $k = 1\sim 8$ )	8
거기에 둔 후, 돌을 취할 수 있는가?(취하는 수: $k = 1\sim 8$ )	8
거기에 둔 후에 해당 연을 빼앗길 경우 몇 개의 돌을 빼앗기는가? (돌의 수: $k = 1\sim 8$ )	8
거기에 둔 후의 해당 연의 호흡점의 수(호흡점의 수: $k = 1\sim 8$ )	8
거기에 둔 후 인접하는 상대의 연을 축으로 취할 수 있는가?	1
거기에 놓인 후 인접하는 아군의 연을 축으로 빼앗기는가?	1
합법 수인가?	1
모두 1로 채운다.	1
모두 0으로 채운다.	1
합계	48

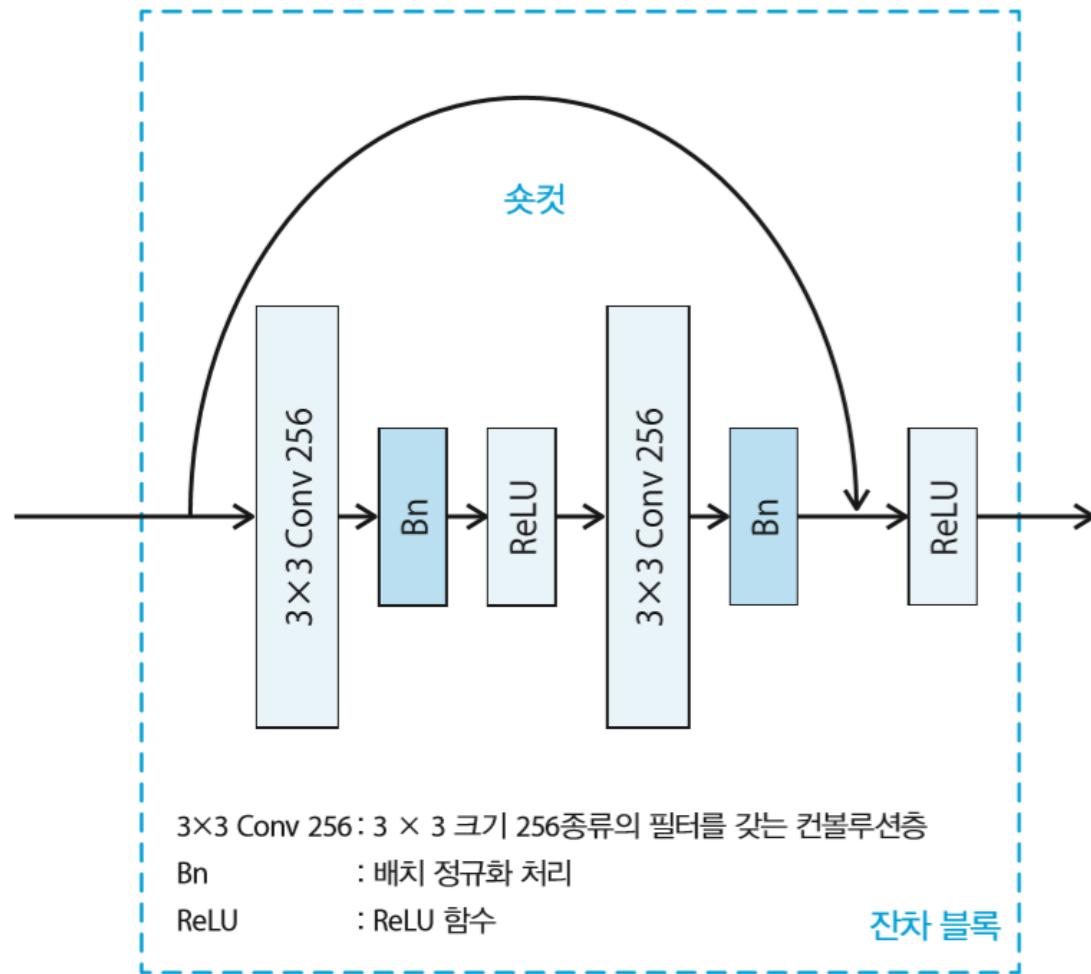
(b) 알파고 제로의 듀얼 네트워크에서 사용된 입력 17채널

입력 채널의 종류	채널 수
흑돌의 위치	1
백돌의 위치	1
$k$ 수 앞의 흑돌의 위치( $k = 1\sim 7$ )	7
$k$ 수 앞의 백돌의 위치( $k = 1\sim 7$ )	7
수의 차례(흑의 차례라면 모두 1, 백의 차례라면 모두 0)	1
합계	17

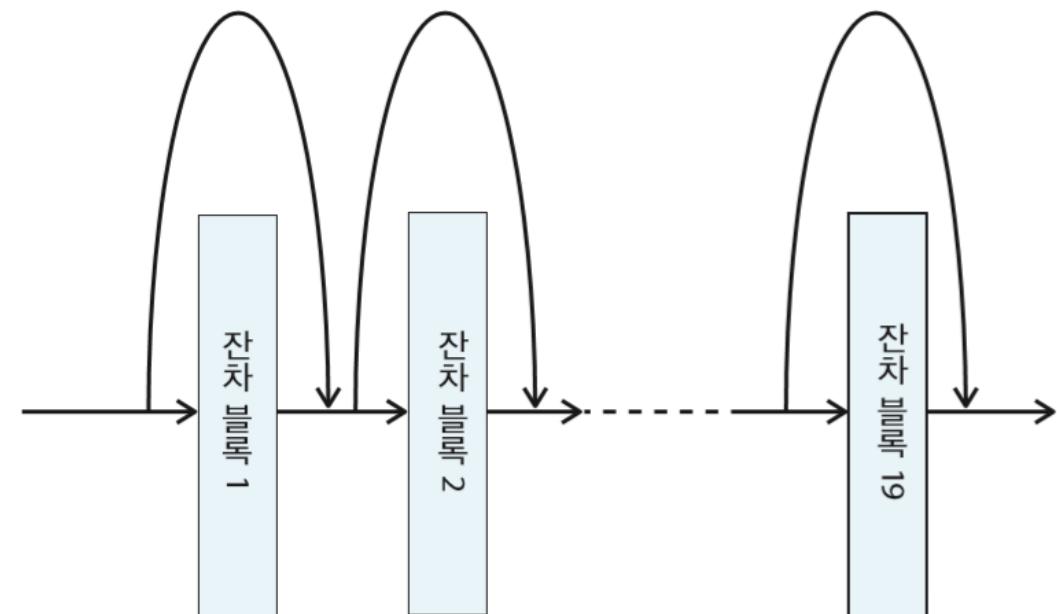


# 듀얼 네트워크의 잔차 네트워크 부분

(a) 잔차 블록의 구조 상세



(b) 듀얼 네트워크는 잔차 블록을 19회 반복한다.



# 듀얼 네트워크의 학습

- 듀얼 네트워크 :  $f_\theta(s)$
- 출력과 정답 레이블  
 $(p, v) \rightarrow (\pi, z)$
- 듀얼 네트워크의 파라미터  $\theta$  의 갱신식

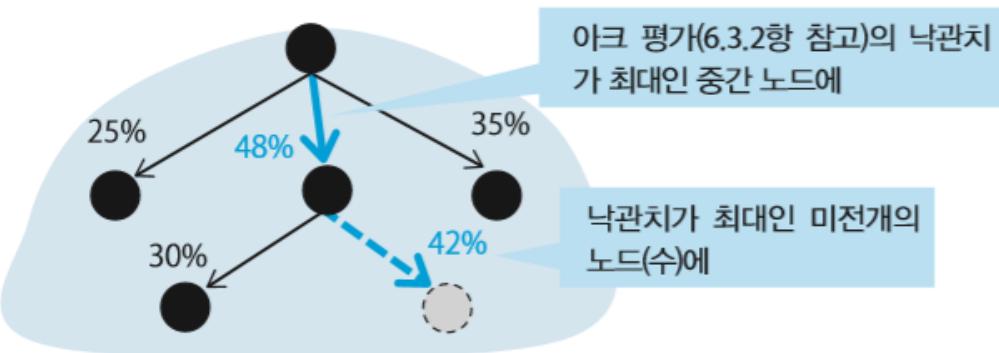
$$\theta \rightarrow \theta - \alpha \cdot \Delta\theta, \quad \Delta\theta = \frac{\partial L_\theta}{\partial \theta}$$
$$L_\theta = \sum_{k=1}^M \left\{ (z^k - v^k)^2 - \sum_{a=1}^A \pi_a^k \log p_a^k \right\} + c \sum_i \theta_i^2$$

- 학습 효과
  - 기존 알파고와 마찬가지로 3,000만 국면을 사용하여 지도 학습한 경우 강한 플레이어 수 와의 일치율이 60.4%가 된 것으로 나타났다.
  - 이것은 최고 57.0%였던 기존 알파고의 일치율보다도 높은 값이다.
  - 잔차 블록을 도입하고 네트워크의 깊이를 깊게 한 효과라고 말할 수도 있겠다.

# 알파고 제로의 몬테 카를로 트리 탐색

- (a) 루트 노드에서 아크 평가의 낙관치  $Q(s,a) + u(s,a)$ 가 최대가 되는 수  $a$ 를 따라 트리를 내려간다.
- (b) 새로운 노드를 만들고 듀얼 네트워크에 의해  $p, v$ 를 계산한다.
- (c) 각 노드의 승률을 갱신하면서 트리를 올라간다

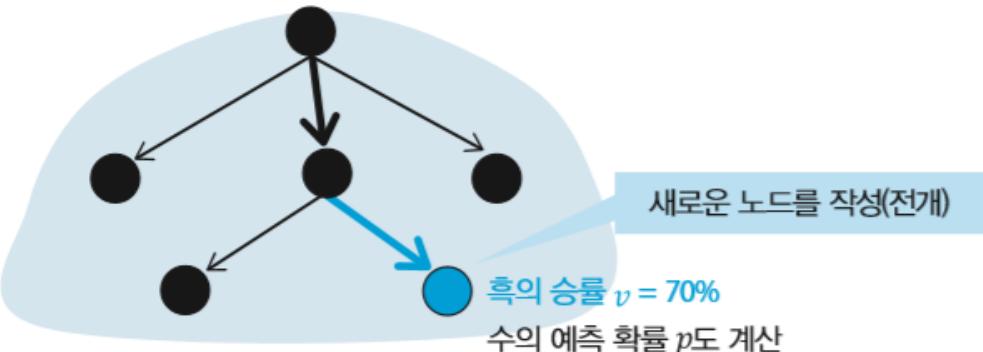
(a) Step 1: 루트 노드에서 아크 평가의 낙관치( $Q(s, a) + u(s, a)$ )가 최대가 되는 수  $a$ 를 따라 트리를 내려간다.



아크 평가(6.3.2항 참고)의 낙관치  
가 최대인 중간 노드에

낙관치가 최대인 미전개의  
노드(수)에

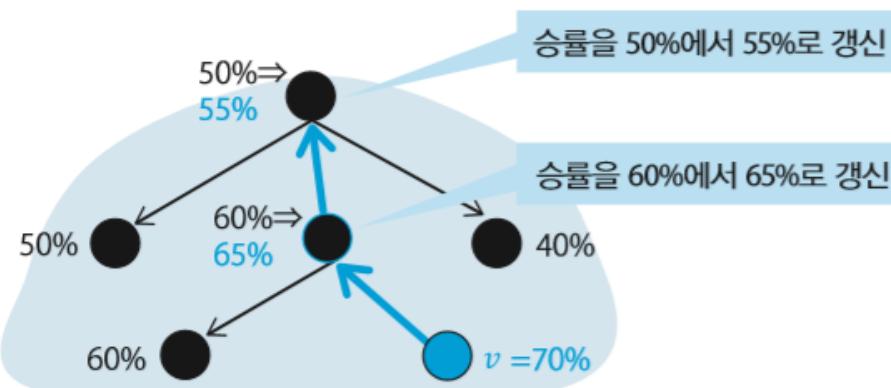
(b) Step 2: 새로운 노드를 만들고 듀얼 네트워크에 의해  $p, v$ 를 계산한다.



새로운 노드를 작성(전개)

흑의 승률  $v = 70\%$   
수의 예측 확률  $p$ 도 계산

(c) Step 3: 각 노드의 승률을 갱신하면서 트리를 올라간다.

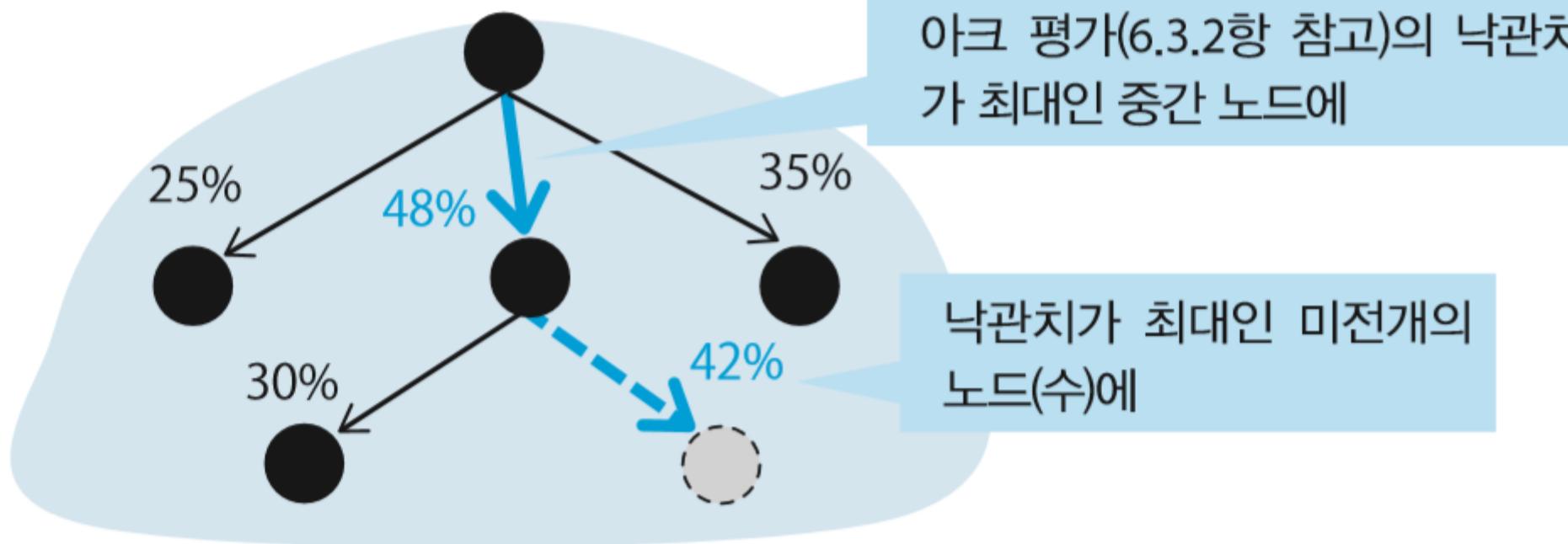


승률을 50%에서 55%로 갱신

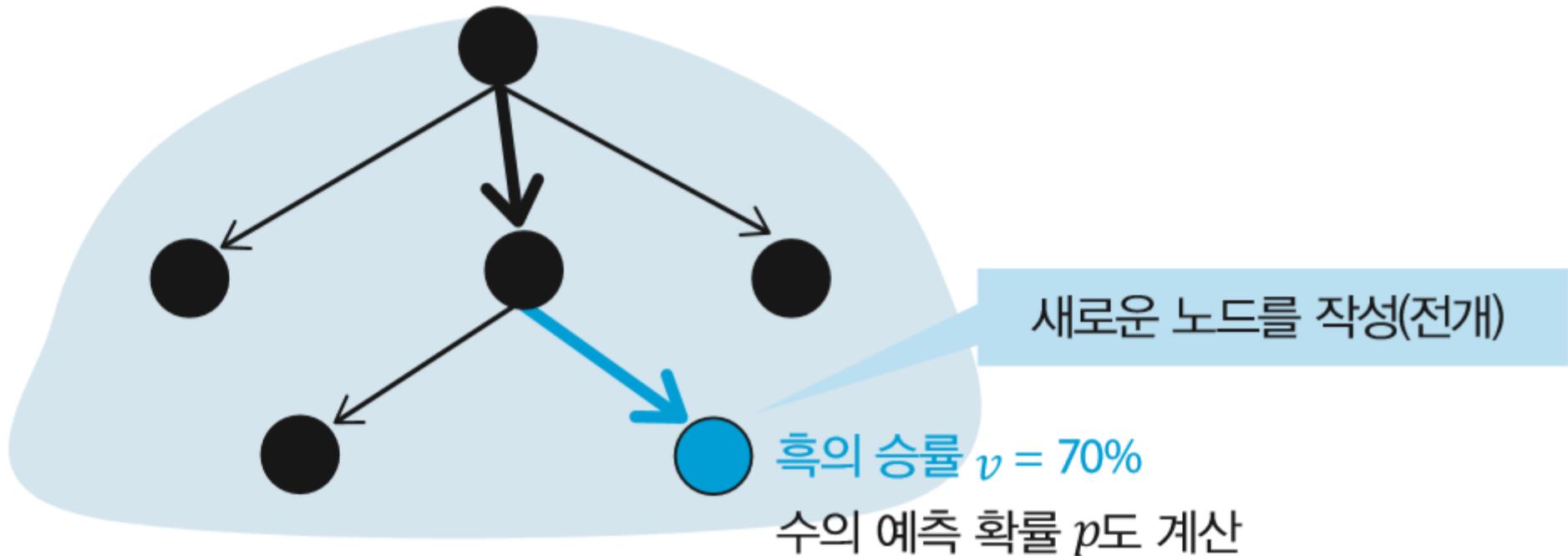
승률을 60%에서 65%로 갱신

50%  $\Rightarrow$  55%  
60%  $\Rightarrow$  65%  
40%  
60%  
 $v = 70\%$

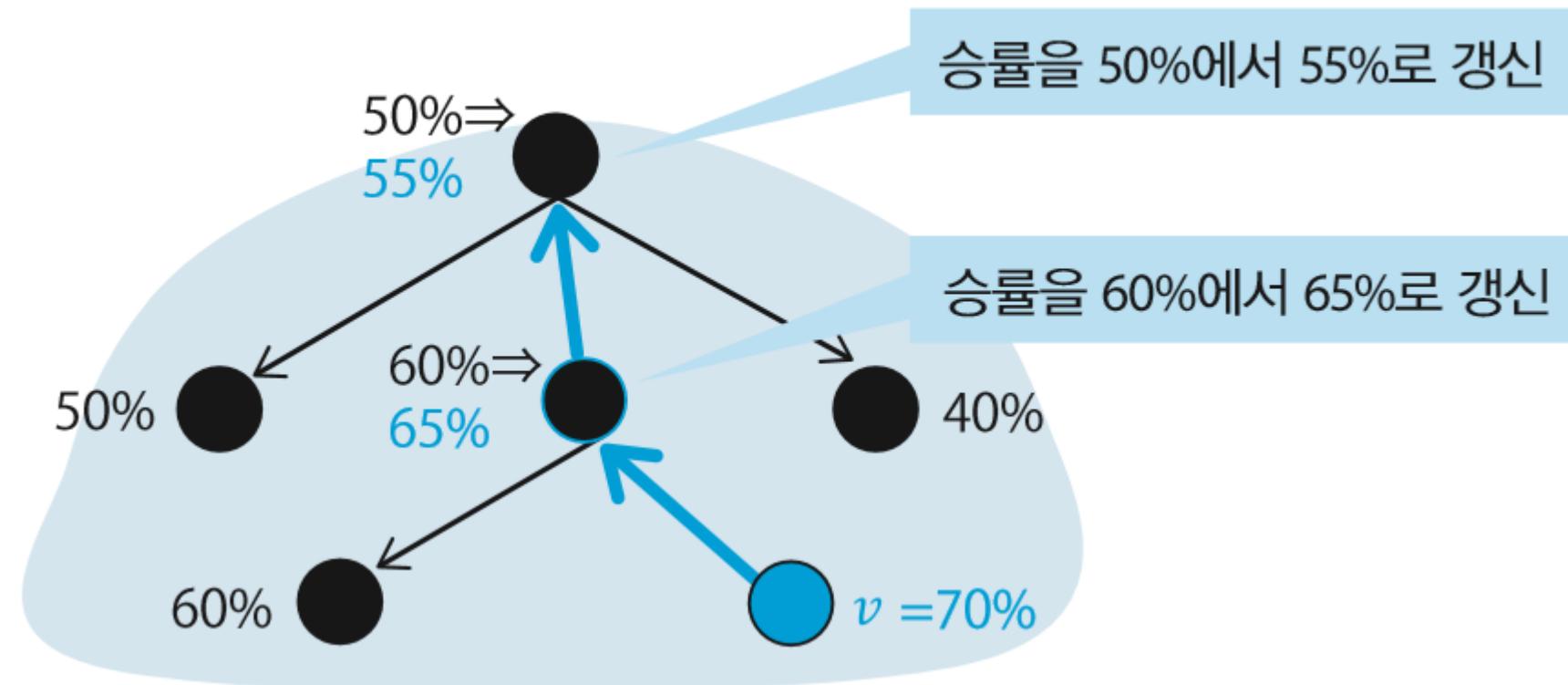
(a) Step 1: 루트 노드에서 아크 평가의 낙관치( $Q(s, a) + u(s, a)$ )가 최대가 되는 수  $a$ 를 따라 트리를 내려간다.



(b) Step 2: 새로운 노드를 만들고 듀얼 네트워크에 의해  $p, v$ 를 계산한다.

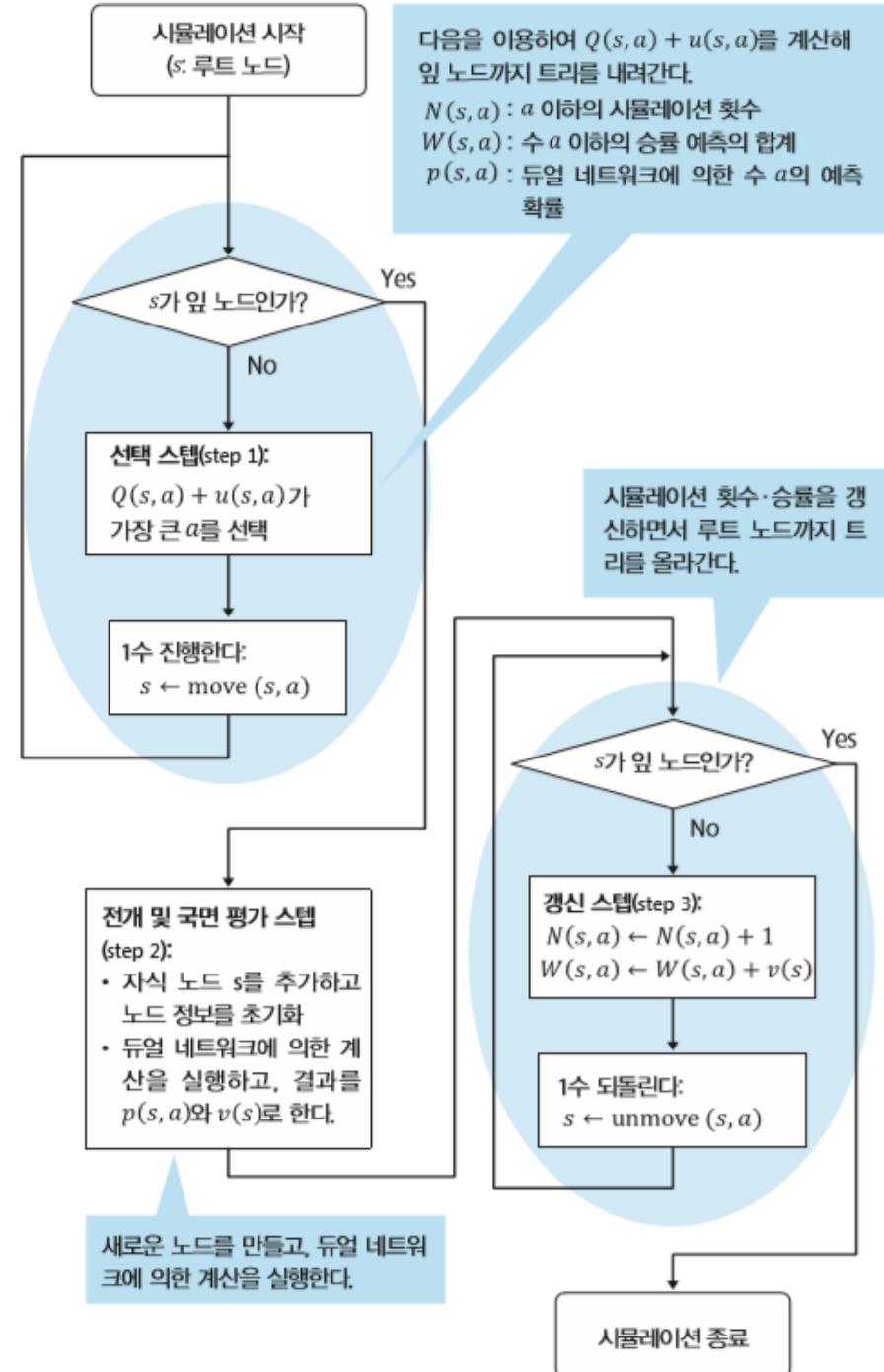


(c) Step 3: 각 노드의 승률을 갱신하면서 트리를 올라간다.



# 알파고 제로의 몬테 카를로 트리 탐색의 시뮬레이션 1회의 처리에 대한 플로차트

- 기존 버전 알파고가 채용한 APV-MCTS의 플로 차트와 비교하면 간단하다
- Step 1~3의 시뮬레이션을 1600회 반복한 후에 루트 국면에 있어서 가장 시뮬레이션 횟수가 많았던 수를 채용한다.



### 6.3.2 몬테카를로 트리 탐색의 플로 차트

다음으로, 몬테카를로 트리 탐색의 처리에 대해 자세히 살펴보자.

**Step 1~3**로 구성된 시뮬레이션 처리를 반복한다.

## Step 1(선택 처리)

먼저, Step 1에서는 국면  $s$ 에서 아래 식으로 계산되는 아크 평가치  $Q(s, a) + u(s, a)$ 가 최대가 되는 수  $a$ 를 거쳐서 트리를 내려간다. 또한, 평가라는 용어는 혼동을 야기하므로 선택 처리에 의한 각 수의 평가는 ‘아크 평가’라고 부르기로 한다. 한편, 국면(노드)를 평가하는 경우 ‘국면 평가’라고 부르겠다.

$$Q(s, a) + u(s, a)$$

승률  

$$Q(s, a) = \frac{W(s, a)}{N(s, a)},$$
  

$$u(s, a) = c_{puct} \cdot p(s, a)$$
 $\sqrt{\frac{\sum_b N(s, b)}{1 + N(s, a)}}$   
수  $a$ 의 예측 확률  
바이어스

### 식 6.2 아크 평가치의 계산 방법

여기에서  $Q(s, a)$ 는 승률이다(식 6.2),  $u(s, a)$ 는 듀얼 네트워크를 출력하는 수  $a$ 의 예측 확률과 바이어스의 곱으로 되어 있다. 바이어스는 수  $a$ 에서 시작하는 시뮬레이션의 횟수가 적을 때에는 큰 값이 되고, 횟수가 많아지면 작아진다. 즉, 신뢰 구간의 크기를 나타내는 것으로 간주할 수 있다. 따라서  $u(s, a)$  전체로 봤을 때는 예측 확률  $p(s, a)$ 가 크거나 혹은 수  $a$ 에서 시작하는 시뮬레이션의 횟수가 작을 때에 큰 값이 된다. 마지막으로,  $c_{puct}$ 는 승률  $Q(s, a)$ 와  $u(s, a)$ 의 균형을 결정하는 파라미터를 나타낸다. 이상의

## Step 2(전개 및 국면 평가 처리)

Step 2에서는 자식 노드  $s'$ 를 전개하여 듀얼 네트워크  $f_\theta$ 에 의해  $p(s', a)$ ,  $v(s')$ 를 계산한다. 기존 버전 알파고에서는 노드 전개의 빈도는  $n (= 40)$  시뮬레이션에 1회였지만, 알파고 제로에서는 매번 노드가 만들어진다.

### Step 3(갱신 처리)

Step 3에서는 루트 노드까지 도중의 모든 노드  $s$ 에 대해 승률의 합계  $W(s, a)$ 와 시뮬레이션 횟수의 합계  $N(s, a)$ 를 [식 6.3](#)과 같이 개시하면서 트리를 올라간다.

$$W(s,a) = W(s,a) + v(s)$$

### 식 6.3 시뮬레이션 횟수의 합계 $N(s, a)$ 와 승률의 합계 $W(s, a)$

이 처리에 의해 결과적으로  $N(s, a)$ 와  $W(s, a)$ 에 각각 국면  $s$ 의 수  $a$ 부터 시작하는 시뮬레이션의 횟수와 승률의 합계가 보관되는 것을 확인하길 바란다. 기존 버전의 알파고에서는 밸류 네트워크에 의한 승률 평가와 플레이 아웃의 승률 평가가 모두 필요했지만(그림 5.6 Step 4 참고), 알파고 제로는 듀얼 네트워크가 출력하는 승률 평가  $v(s)$ 에 관한 것만으로도 괜찮기 때문에 간단하다.

이 개선 처리의 결과로  $W(s, a)/N(s, a)$ 에 의해 해당 국면  $s$ 의 수  $a$ 부터 시작하는 모든 시뮬레이션 승률의 평균을 계산할 수 있다. MCTS의 중요한 변화가 중점적으로 전개되는 성질과 결부하면 시뮬레이션을 거듭할수록  $W(s, a)$ 와  $N(s, a)$ 에 의한 승률의 정확도가 높아질 것으로 생각된다. 참고로, 알파고 제로의 MCTS에서는 승부가 날 때까지 플레이 아웃을 하고 있는 것이 아니라, 단순히 듀얼 네트워크의 승률 예측에 근사하고 있으므로 기존의 MCTS에서 보인 이론적으로 최선 수에 수렴하는 성질(4.4.4항 메모 참고)은 없다.

시뮬레이션 시작  
( $s$ : 루트 노드)

다음을 이용하여  $Q(s, a) + u(s, a)$ 를 계산해  
잎 노드까지 트리를 내려간다.

$N(s, a)$ :  $a$  이하의 시뮬레이션 횟수  
 $W(s, a)$ : 수  $a$  이하의 승률 예측의 합계  
 $p(s, a)$ : 듀얼 네트워크에 의한 수  $a$ 의 예측  
확률

$s$ 가 잎 노드인가?

Yes

선택 스텝(step 1):

$Q(s, a) + u(s, a)$ 가  
가장 큰  $a$ 를 선택

1수 진행한다:  
 $s \leftarrow \text{move}(s, a)$

시뮬레이션 횟수·승률을 갱  
신하면서 루트 노드까지 트  
리를 올라간다.

$s$ 가 잎 노드인가?

No

전개 및 국면 평가 스텝  
(step 2):

- 자식 노드  $s$ 를 추가하고  
노드 정보를 초기화
- 듀얼 네트워크에 의한 계  
산을 실행하고, 결과를  
 $p(s, a)$ 와  $v(s)$ 로 한다.

새로운 노드를 만들고, 듀얼 네트워  
크에 의한 계산을 실행한다.

$s$ 가 잎 노드인가?

No

갱신 스텝(step 3):  
 $N(s, a) \leftarrow N(s, a) + 1$   
 $W(s, a) \leftarrow W(s, a) + v(s)$

1수 되돌린다:  
 $s \leftarrow \text{unmove}(s, a)$

시뮬레이션 종료

# 알파고 제로의 몬테카를로 트리 탐색 정리

- 플레이 아웃을 없앰으로써 초당 생성할 수 있는 노드 수가 늘어남
- 4,000노드/초 정도의 속도로 노드를 생성
- 지금까지의 검색 결과( $W(s,a)$ ,  $N(s,a)$ )와 그 노드에서 계산된 값( $p(s,a)$ ,  $v(s)$ )만으로 탐색 순서가 결정되므로, 일종의 최선 우선 탐색(Best-first search)으로 간주될 수 있음

※ 최선 우선 탐색(Best-first search)이란 어떤 평가 기준을 바탕으로 가장 바람직한 노드부터 순서대로 탐색해 나가는 방법이다.

# 알파고 제로의 강화 학습

- 강화 학습이란?

- AI가 성공 경험을 바탕으로 행동을 개선해 나가는 비지도 학습의 일종

- 교사 데이터가 없는 경우(프로를 초과하는 경우) 등에 유효

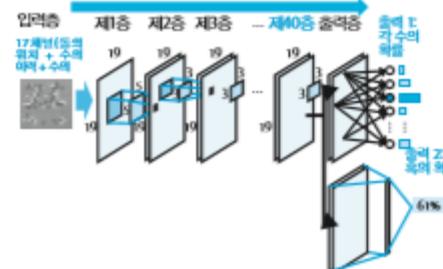
- 알파고 제로에서는

- '셀프 플레이' ⇒ '가능한 한 이긴 쪽의 수를 쉽게 두도록 파라미터 갱신'을 반복

- 랜덤 플레이의 초기 상태에서 교사 없이 프로를 뛰어넘는 수준까지 학습하는 데 성공

듀얼 네트워크의 파라미터를 갱신

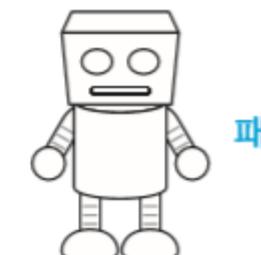
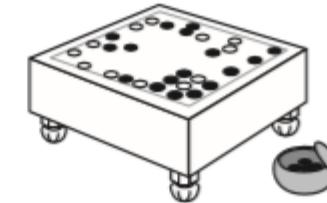
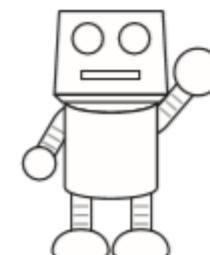
AlphaGo



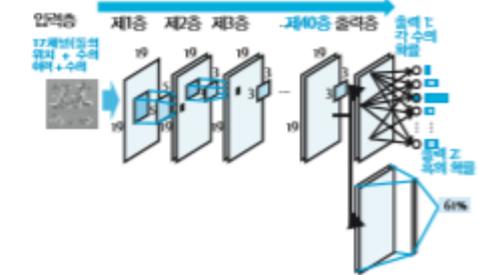
AlphaGo

셀프 플레이  
(자기 대전)

승

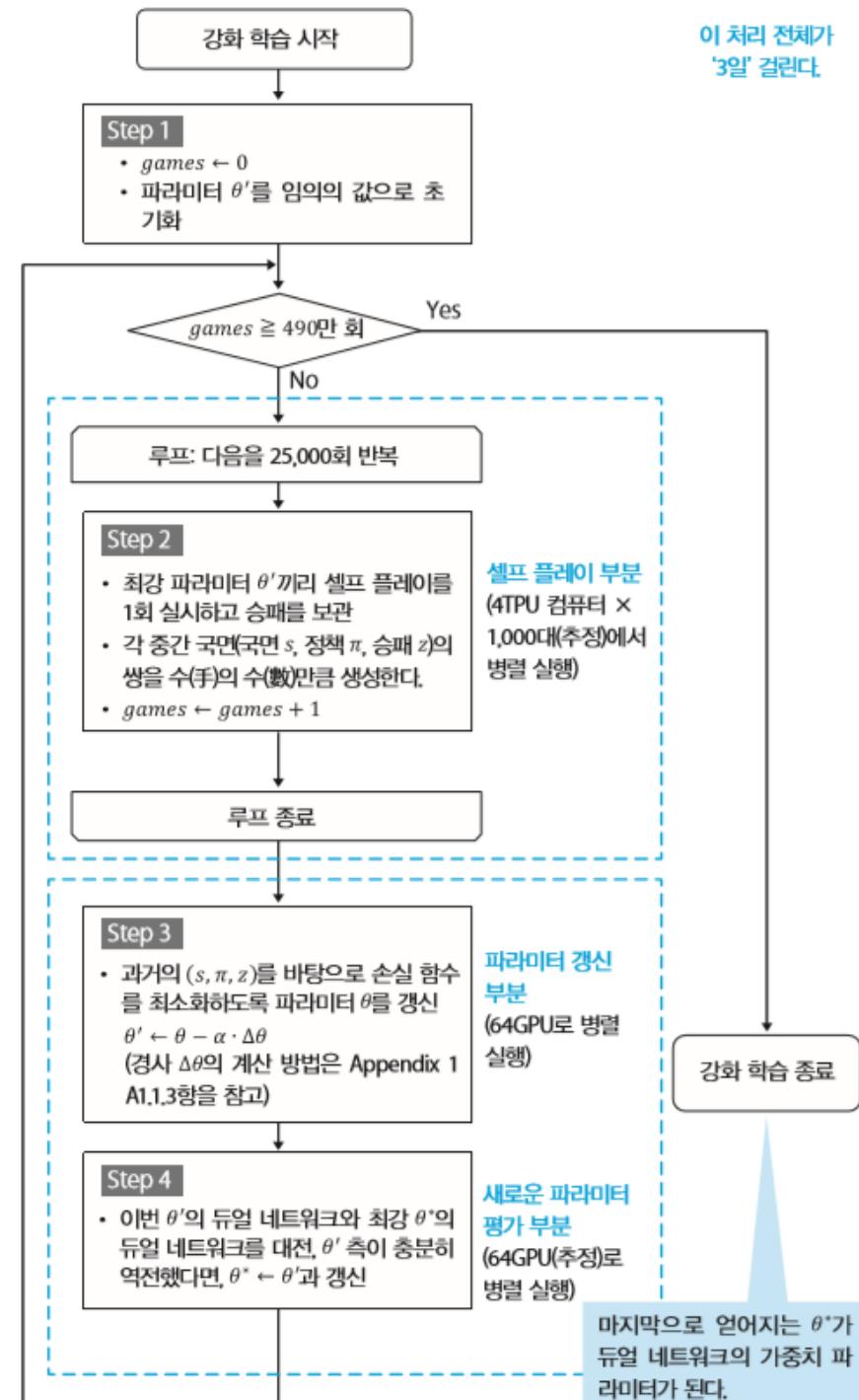


패



# 알파고 제로의 강화 학습 플로 차트

- 셀프 플레이 부분, 파라미터 갱신 부분, 새로운 파라미터 평가 부분의 세 가지 부분의 반복으로 이뤄졌다



## 강화 학습 시작

이 처리 전체가  
'3일' 걸린다.

### Step 1

- $games \leftarrow 0$
- 파라미터  $\theta'$ 를 임의의 값으로 초기화

$games \geq 490\text{만 회}$

No

루프: 다음을 25,000회 반복

### Step 2

- 최강 파라미터  $\theta'$ 끼리 셀프 플레이를 1회 실시하고 승패를 보관
- 각 중간 국면(국면  $s$ , 정책  $\pi$ , 승패  $z$ )의 쌍을 수(手)의 수(數)만큼 생성한다.
- $games \leftarrow games + 1$

셀프 플레이 부분  
(4TPU 컴퓨터 ×  
1,000대(추정)에서  
병렬 실행)

루프 종료

### Step 3

- 과거의  $(s, \pi, z)$ 를 바탕으로 손실 함수를 최소화하도록 파라미터  $\theta$ 를 갱신  
 $\theta' \leftarrow \theta - \alpha \cdot \Delta\theta$   
(경사  $\Delta\theta$ 의 계산 방법은 Appendix 1 A1.1.3항을 참고)

파라미터 갱신  
부분  
(64GPU로 병렬  
실행)

### Step 4

- 이번  $\theta'$ 의 듀얼 네트워크와 최강  $\theta^*$ 의 듀얼 네트워크를 대전,  $\theta'$  측이 충분히 역전했다면,  $\theta^* \leftarrow \theta'$ 과 갱신

새로운 파라미터  
평가 부분  
(64GPU(추정)로  
병렬 실행)

마지막으로 얻어지는  $\theta^*$ 가  
듀얼 네트워크의 가중치 파  
라미터가 된다.

강화 학습 종료

# 강화학습에서의 셀프 플레이 부분의 처리

- 일반적인 MCTS에서는 최종적인 수의 선택으로 시뮬레이션 횟수가 가장 큰 자식 노드의 수를 선택
- 셀프 플레이 시의 MCTS는
  - 첫수에서 30수째까지 한정해서 시뮬레이션 횟수에 비례한 확률로 자식 노드의 수를 선택
  - 그 이후에는 시뮬레이션 횟수가 가장 큰 자식 노드의 수를 선택
- 한 수마다 각 루트 국면  $s$ 에 대한 각 수  $a$ 가 시뮬레이션된 횟수  $N(s,a)$ 와 최종적인 승패  $z$ 을 보관해 둔다.

# 강화학습에서의 파라미터 갱신 부분의 처리

- 듀얼 네트워크의 지도 학습에는 다음으로 어느 수를 선택할 것인가라는 정답 데이터  $\pi$ 와 승률의 정답 데이터  $z$ 가 필요함
- $z$ 에 관해서는 셀프 플레이의 승패를 이용
- $\pi$ 에 관해서는
  - 일반적인 지도 학습의 경우에는 0-1 방식을 채용
  - 알파고 제로에서는 확률 분포 방식을 채용

※ 0-1 방식 : 정답 수(강한 플레이어가 둔 수)만을 100%로 하고, 나머지를 0%로 하는 방식

※ 확률 분포 방식 : 모든 후보 수에 확률값을 매긴 벡터를 이용하는 방식

# 강화학습에서의 파라미터 갱신 부분의 처리

- 확률 분포 방식에서의 각 수  $a$ 의 확률

$$\pi_a = \frac{N(s, a)^{1/\gamma}}{\sum_b N(s, b)^{1/\gamma}}$$

- 온도 파라미터 :  $\gamma$

- $\gamma = 0$  :  $N(s, a)$ 가 최대인 수  $a$ 를 100% 선택 (0-1 방식과 동일)
- $\gamma = 1$  :  $N(s, a)$ 의 크기에 비례하여 수  $a$ 를 선택

# 강화 학습에서의 새로운 파라미터 평가 부분에 대한 처리

- 새로 얻은 듀얼 네트워크  $f_{\theta'}$ 와 기존 최강의 듀얼 네트워크  $f_{\theta^*}$  사이에서 400회 셀프 플레이를 시킨다.
- 그 결과  $f_{\theta'}$ 측이  $f_{\theta^*}$ 에 대해 220승 이상 이긴 경우에  $\theta'$ 에서  $\theta^*$ 으로 잠정 최강의 파라미터를 갱신한다.

# 강화 학습의 계산 시간

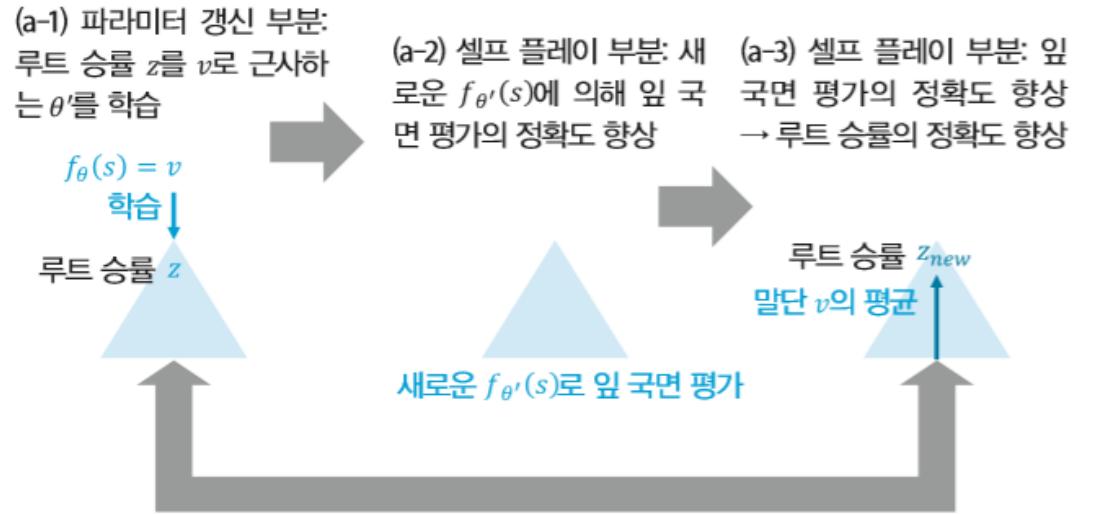
- 셀프 플레이 부분, 파라미터 갱신 부분, 새로운 파라미터 평가 부분의 세 가지 중에 대부분의 시간은 셀프 플레이 부분에서 소요된다.
- 셀프 플레이에서 1수당 1600회의 시뮬레이션을 수행하여, 계산 시간은 0.4초 정도 소요됨.
- 490만 국의 셀프 플레이에서 1국의 셀프 플레이가 150수에서 종료했다고 가정하면 약 10년 가까이 걸림
  - $0.4 \text{ 초/수} \times 150 \text{ 수/국} \times 490\text{만 국} \approx 2.9\text{억 초} \approx 3400\text{일} \approx 9.3\text{년}$
  - 약 1000대의 병렬화로 3일에 끝난 것으로 추정

# 알파고 제로의 강화학습은 무엇을 하고 있나?

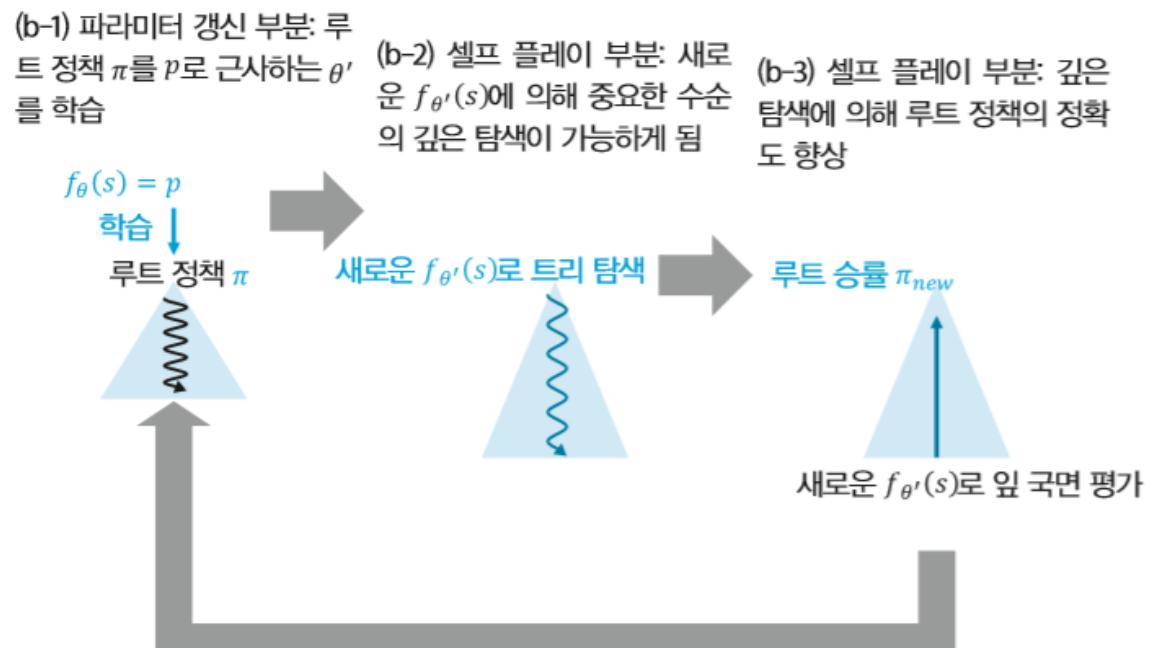
- 어떤 구조로 듀얼 네트워크  $f_\theta$ 의 파라미터  $\theta$ 는 강한 파라미터가 되어 가는 것일까?
  - 국면 평가함수의 질을 높이면 검색의 질이 좋아지고, 깊이 제어의 질을 높이면 검색의 질이 좋아진다.
  - 게임 트리 탐색의 성질
  - 피드백 구조 (승률 예측 부분, 다음의 한수 예측 부분)
- 게임 트리 탐색에서 중요한 두 가지
  - 중요한 변화를 깊게 읽기 위한 기법
  - 승률을 정확하게 예측할 수 있는 평가 함수

# 알파고 제로의 강화 학습에서 의 두 가지 피 드백 구조

(a) 승률 예측에 관한 정상적인 피드백



(b) 다음의 한 수에 관한 정상적인 피드백



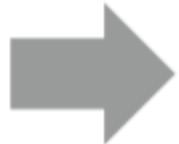
(a) 승률 예측에 관한 정상적인 피드백

(a-1) 파라미터 갱신 부분:  
루트 승률  $z$ 를  $v$ 로 근사하  
는  $\theta'$ 를 학습

$$f_{\theta}(s) = v$$

학습

루트 승률  $z$



(a-2) 셀프 플레이 부분: 새  
로운  $f_{\theta'}(s)$ 에 의해 잎 국  
면 평가의 정확도 향상

(a-3) 셀프 플레이 부분: 잎  
국면 평가의 정확도 향상  
→ 루트 승률의 정확도 향상

새로운  $f_{\theta'}(s)$ 로 잎 국면 평가



루트 승률  $z_{new}$   
말단  $v$ 의 평균



## (b) 다음의 한 수에 관한 정상적인 피드백

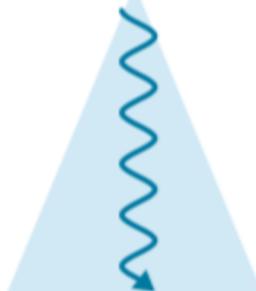
(b-1) 파라미터 갱신 부분: 루트 정책  $\pi$ 를  $p$ 로 근사하는  $\theta'$ 를 학습

$f_{\theta}(s) = p$   
학습  
루트 정책  $\pi$



(b-2) 셀프 플레이 부분: 새로운  $f_{\theta'}(s)$ 에 의해 중요한 수순의 깊은 탐색이 가능하게 됨

새로운  $f_{\theta'}(s)$ 로 트리 탐색



(b-3) 셀프 플레이 부분: 깊은 탐색에 의해 루트 정책의 정확도 향상

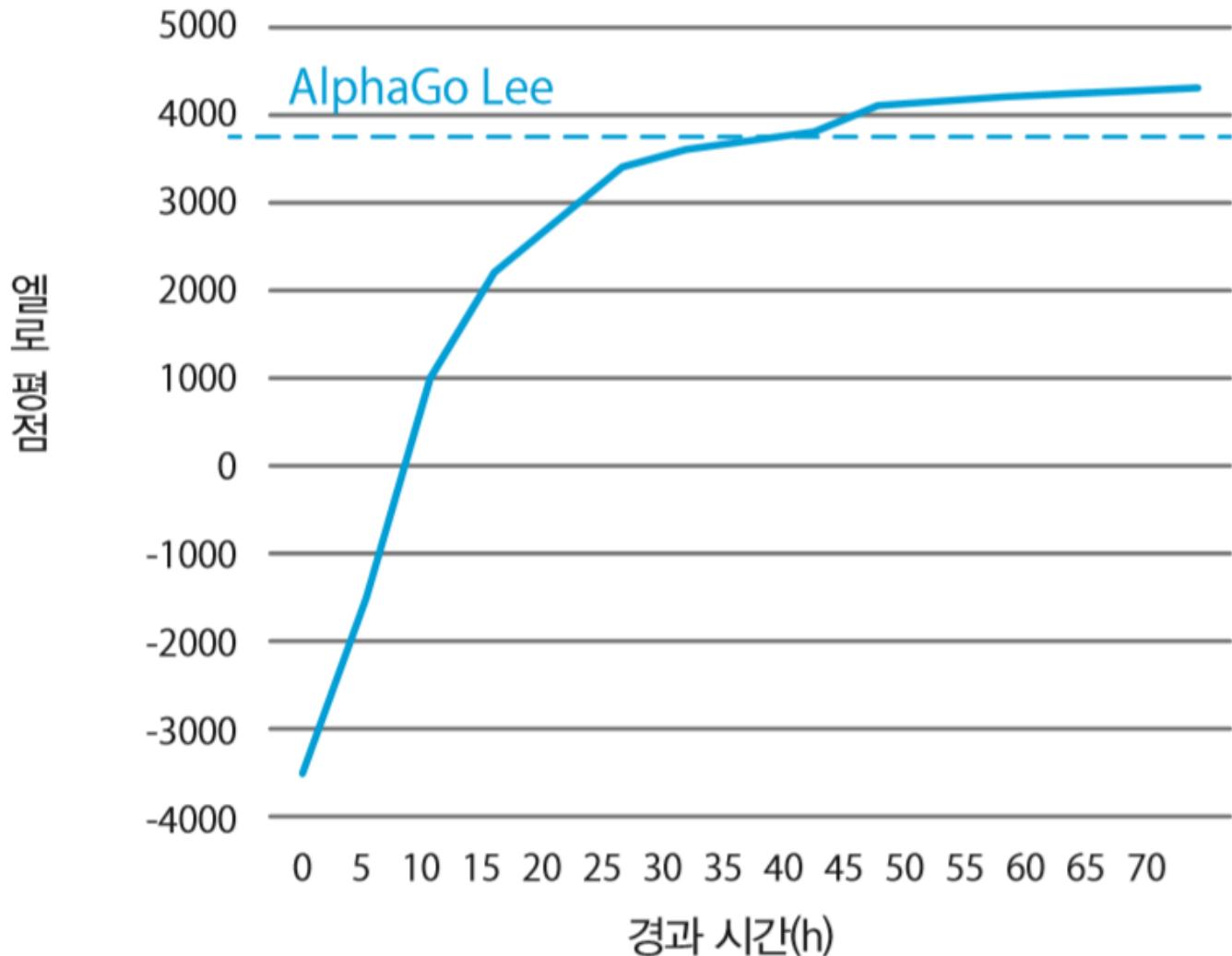
루트 정책  $\pi_{new}$

새로운  $f_{\theta'}(s)$ 로 잎 국면 평가



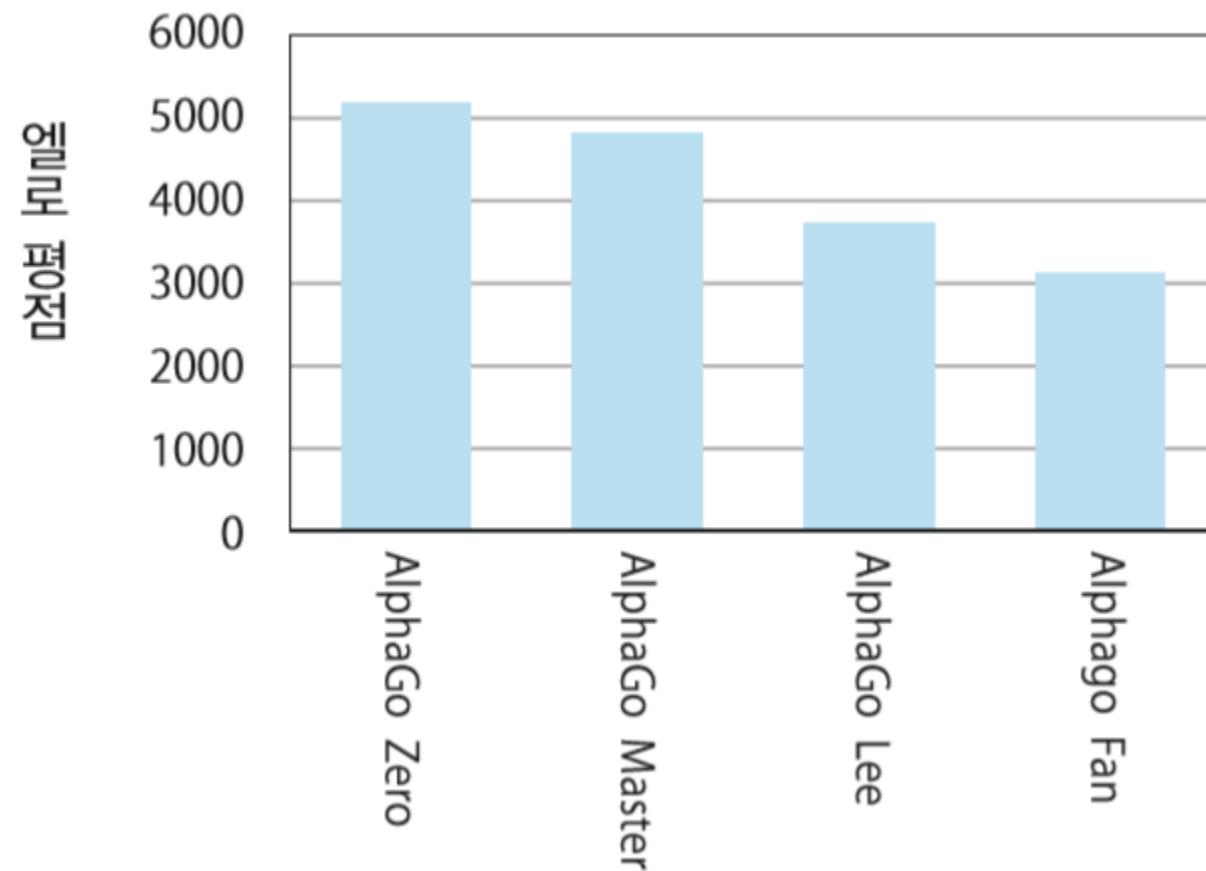
# 알파고 제로의 강화 학습의 결과

- 처음에는 랜덤 파라미터인 평점 -3,500 점에서 시작하지만, 점차 강해져 72시간 후에는 인류 최강 수준에 도달한다



# 알파고의 강함의 변천

(a) 알파고의 네 가지 버전의 평점



# 알파고 의 각 버전의 개요

(b) 알파고의 네 가지 버전의 상세

	대전 성적	사용된 기술	대전에 사용한 HW 사양
AlphaGo Fan	2015년 10월에 판 후이 2단에게 승리	알파고 논문(이 책 5장까지 설명한 것)	176GPU, 48TPU
AlphaGo Lee	2016년 3월에 이세돌 9단에게 4:1로 승리	<p>AlphaGo Fan과는 다음과 같은 점에서 차이가 있다.</p> <ul style="list-style-type: none"> <li>• 자기 대전에 의한 강화 학습을 통해 얻은 정책 네트워크를 이용</li> <li>• 14층보다 깊은 네트워크를 이용</li> </ul>	176GPU, 48TPU
AlphaGo Master	2017년 1월에 바둑 대전 사이트에서 60승 0패로 인간에게 승리	<p>AlphaGo Lee와는 다음과 같은 점에서 차이가 있다.</p> <ul style="list-style-type: none"> <li>• MCTS에는 AlphaGo Fan 수준의 특징량을 이용한 플레이 아웃을 이용</li> </ul>	4TPU
AlphaGo Zero	AlphaGo Master에 89승 11패	AlphaGo Master에 89승 11패(단, 잔차 블록 39단의 네트워크를 이용)	4TPU



The End!