

## Machine Learning in Practice

### #8-2: Case Study: ALPHAGO

Sang-Hyun Yoon

Summer 2019

## References

그림 및 내용 출처:

- D. Silver *et al.*, Mastering the game of Go with deep neural networks and tree search, *Nature*, 529:484-489, 2016
  - ▶ <http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources.html>
- M. Chang, AlphaGo in Depth,  
<http://www.slideshare.net/ckmarkohchang/alphago-in-depth>
- S. Moon, How AlphaGo Works,  
<http://www.slideshare.net/ShaneSeungwhanMoon/how-alphago-works>
- D. Lee, 알파고 해부하기
  - <http://www.slideshare.net/DonghunLee20/1-59501887>
  - <http://www.slideshare.net/DonghunLee20/2-59620244>
  - <http://www.slideshare.net/DonghunLee20/3-61454159>

코드 재현: <https://github.com/Rochester-NRT/RocAlphaGo>

## 기존 MCTS의 문제점

무작위 선택의 **임의성**으로 인한 **노이즈/오버헤드** 발생

- 양 플레이어가 최선을 다하여 결정된 경로가 아니어서 **승패 결과의 신뢰성 부족**
- **실험할 가치가 적은 경로**를 걸러 내기 힘듬
- 결과적으로, 무한히 많은 실험을 할 수는 없으므로 **중요한 경로를 놓치기 쉽고** 실험결과의 정확성이 낮을 수 있음
  - ▶ 바둑처럼 전체 경로수가 매우 많은 경우 중요한 경로를 놓칠 가능성 더 높아짐
  - ▶ 체스의 경우와 같이 “**trap state**”(states that leads to losses within a small number of moves)가 많은 경우에도 비효율적

초반 몇 번의 시뮬레이션 결과로 최종 **결과가 편향**되는 경향

- Selection/expansion 스텝에서 **오차가 큰 판세분석** 결과에 기반하여 탐색/확장하므로

## Constituents of AlphaGo

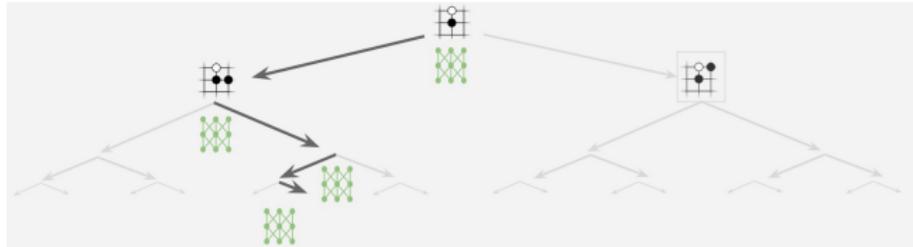
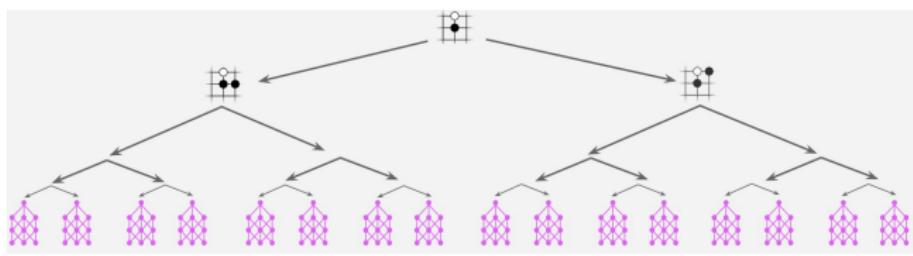
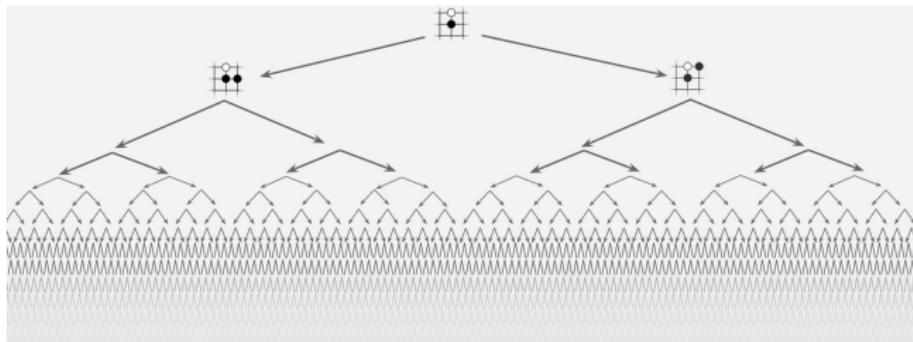
### 4 CNNs for highly-tuned MCTS (before match)

- Policy network (정책망)  $p_\sigma$ ,  $p_\pi$ ,  $p_\rho$ : 다음 함수를 근사화:
  - ▶  $p : \{\text{백,흑,무}\}^{[19]^2} \rightarrow [19]^2$ ;  $p(s) = "s\text{에서 최적의 다음 수}"$
  - $p : \{\text{백,흑,무}\}^{[19]^2} \rightarrow \{x \in \mathbb{R} | 0 \leq x \leq 1\}^{[19]^2}$ 로 사용하기도 함
  - ▶  $p_\sigma$ : (아마추어) 인간의 기보로 학습 (supervised learning)
  - ▶  $p_\pi$ :  $p_\sigma$ 를 단순화한 빠른 버전 (supervised)
  - ▶  $p_\rho$ :  $p_\sigma$ 간의 자가대결 기보로 학습 (reinforced)
- Value network (가치망)  $v_\theta(s) = "s\text{로 시작하면 이길 확률}"$ 
  - ▶  $p_\rho$ 간의 자가대결 결과를 입력 데이터로 학습하여 근사화
- Match 도중에는 CNN을 변경하지 않고 훈련된 그대로 사용

### MCTS with policy/value networks (during match)

- 기존 MCTS의 expansion/expansion/evaluation(simulation)  
각 step에서  $p_\sigma$ ,  $p_\pi$ ,  $v_\theta$  사용해서 MCTS를 정교하게
  - ▶  $p_\rho$ 는  $v_\theta$  훈련을 위한 입력 데이터 생성에만 사용
- 비동기 분산처리 방식으로 HW 자원 활용 극대화

## MCTS with Policy/Value Networks: Intuition



- Value network  
으로 탐색  
depth 줄이기

- Policy network  
으로 탐색  
width 줄이기

MCTS for AlphaGo  
oooooooooooo

Policy/Value Networks  
oooooooooooooooooooo

AlphaGo Zero  
ooooooo

# Outline

1 MCTS for AlphaGo

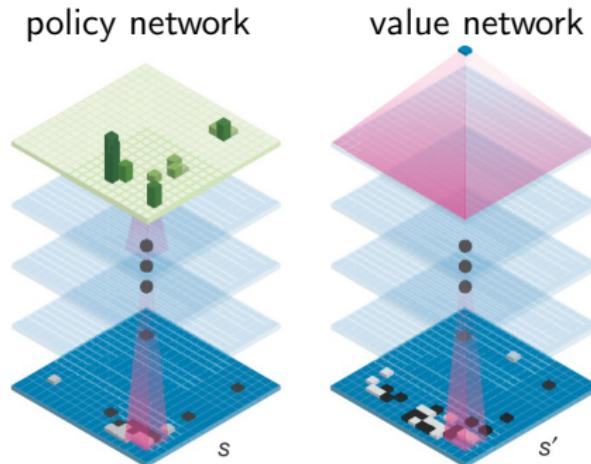
2 Policy/Value Networks

3 AlphaGo Zero

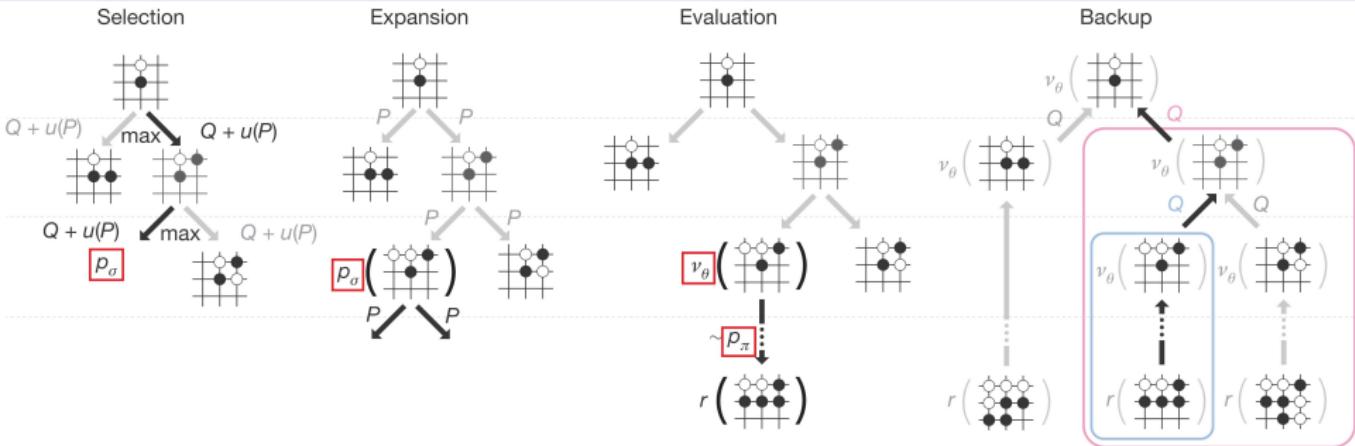
## Recall: Policy/Value Networks for MCTS

모두 13층 CNN으로 구현. 경기전 training하고 경기중 변경 안함

- Policy network:  $p(s) = "s\text{에서 최적의 다음 수}"$  (softmax)
  - ▶  $p_\sigma$ : (아마추어) 인간의 기보로 학습 (supervised)
  - ▶  $p_\pi$ :  $p_\sigma$ 를 단순화한 빠른 버전 (supervised)
  - ▶  $p_\rho$ :  $p_\sigma$ 간의 자가대결 기보로 학습 (reinforced)
- Value network:  $v_\theta(s) = "s\text{로 시작하면 이길 확률}"$ 
  - ▶  $p_\rho$ 간의 자가대결 결과를 입력 데이터로 학습 (supervised)



# MCTS for AlphaGo: Overview



- ① **Selection:** 바둑판 상태  $s$ 에서 택할 수 있는 다음 수  $a$  중  $Q(s, a) + u(s, a)$ 를 최대로 만드는  $a$ 를 선택해가며 leaf까지
  - ▶  $Q$ : 대략 이길 확률 /  $u$ :  $p_\sigma$ 에 비례, 방문빈도에 반비례
- ② **Expansion:**  $p_\sigma$ 에 기반하여 확장 (일반적 MCTS는 무작위 확장)
- ③ **Evaluation:**  $v_\theta$  값과  $p_\pi$ 를 이용한 **fast roll-out** 결과를 가중합
  - ▶ roll-out:  $p_\sigma$ 보다 빠른  $p_\pi$ 로 빠르게 게임 진행해서 얻은 승률
  - ▶  $v_\theta$  값과 roll-out 결과를 1:1로 하면 실험적으로 최적
- ④ **Backup:** 경로의 각 노드의  $Q$  값을 적절히 update

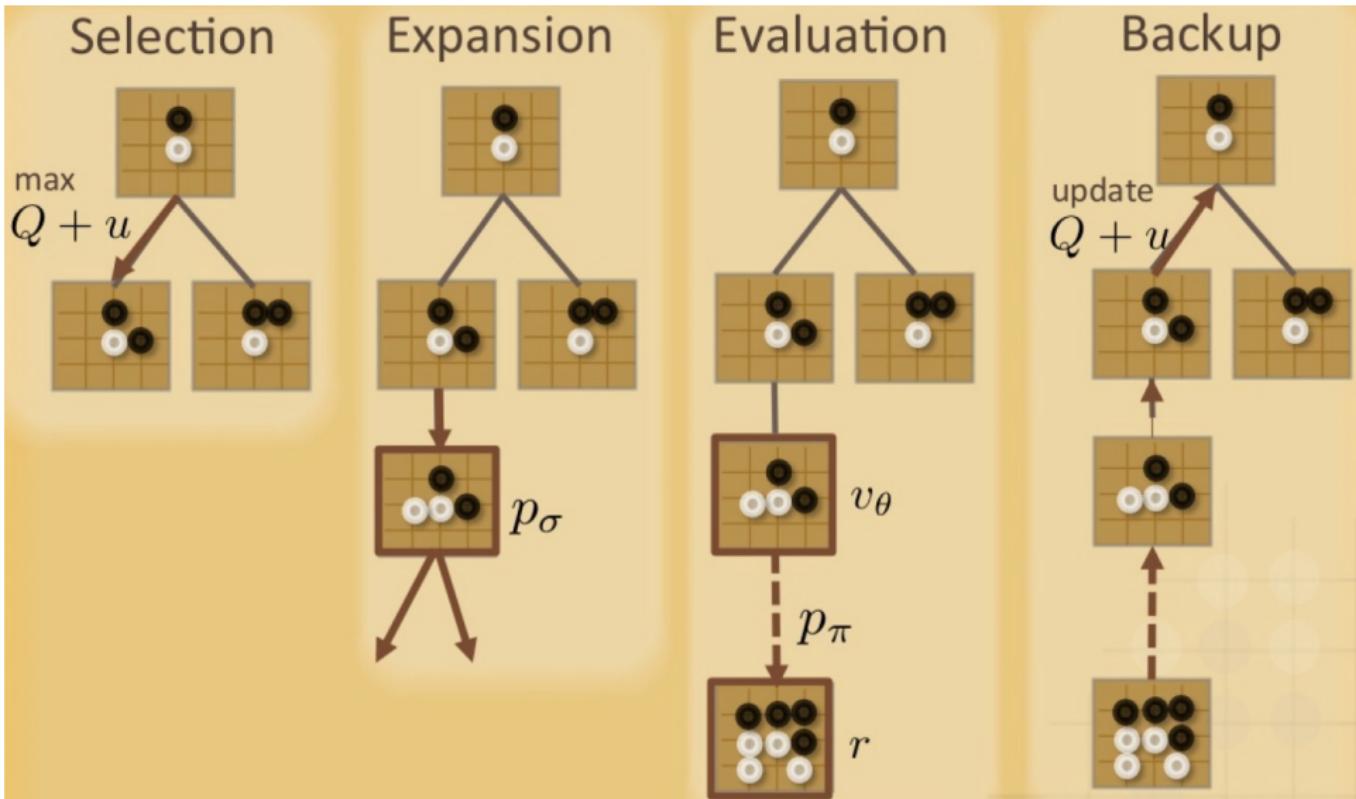
MCTS for AlphaGo  
oo●oooooooo

## Policy/Value Networks



AlphaGo Zero  
oooooo

# MCTS for AlphaGo

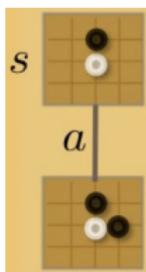


## Information for Each Edge

상태  $s$ 에서 다음 수를  $a$ 로 뒀을 때의 새로운 상태를  $s'$ 로 표시

각 edge  $(s, a)$ 마다 다음 정보들이 저장/update 된다:

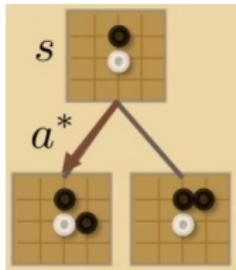
- $N_v(s, a)$ :  $v_\theta(s')$ 를 계산해본 총 횟수
  - $W_v(s, a)$ :  $v_\theta(s')$ 를 누적한 값
  - $N_r(s, a)$ :  $s'$ 를 지나간 ( $p_\pi$ 를 이용한) roll-out의 총 횟수
  - $W_r(s, a)$ :  $N_r(s, a)$ 회 실험에서 승리 횟수의 누적합
  - $P(s, a) = p_\sigma(a|s)$ 
    - ▶ 인간의 기준으로 훈련한  $p_\sigma$ 상에서 상태  $s$ 에서 다음 수로  $a$ 를 두는 확률 값
  - $Q(s, a) = (1 - \lambda) \frac{W_v(s, a)}{N_v(s, a)} + \lambda \frac{W_r(s, a)}{N_r(s, a)}$ 
    - ▶  $\lambda = 0.5$ 로 할 때 실험적으로 성능이 가장 좋음



## Step 1: Selection

## Recall: MTCS with UCT

$$a^* = \underset{a}{\operatorname{argmax}} \left( Q(s, a) + c \sqrt{\frac{2 \ln \sum_b N(s, b)}{1 + N(s, a)}} \right) \text{인 } a^* \text{를 선택}$$

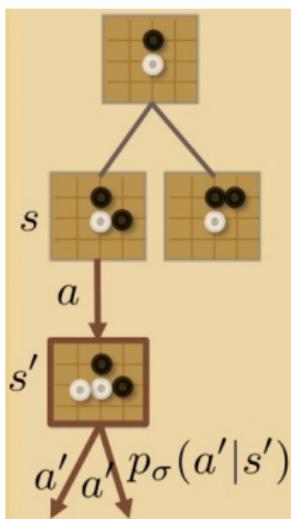


- $$u(s, a) = c \cdot \underbrace{P(s, a)}_{p_\sigma(a|s)} \cdot \frac{\sum_b N_r(s, b)}{1 + N_r(s, a)}$$
    - ▶  $c$ 는 5정도를 사용하나 시간 분배에 따라 조정됨
    - ▶  $c$  값을 키우면 더 많은 노드/경로를 탐색하게 됨
  - $$a^* = \operatorname{argmax}_a (Q(s, a) + u(s, a))$$
인  $a^*$ 를 선택

▶ 할: 승률  $Q$ 가 높은 쪽으로 실험을 더 자주 하도록  
 실험이 충분히 되지 않은 쪽으로도 실험을 유도

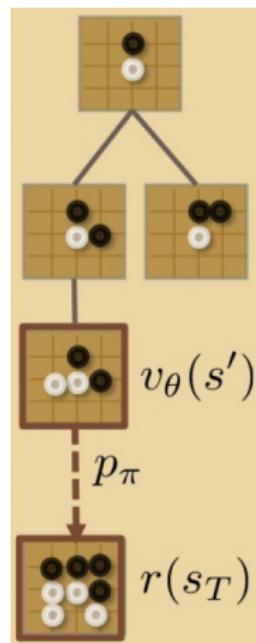
방식과의 차이:  $P$ 를 반영하여 사람이 많이 둔  
 쪽을 더 선호하도록 함

## Step 2: Expansion



- ❶  $s$ 에서  $s'$ 로 가는 경로의 횟수가 threshold  $n_{th}$ 를 넘어서게 되면  $s'$ 에 대응되는 노드를 tree에 추가
  - ▶  $n_{th}$ 는 40정도를 사용하고 시간 분배에 따라 조정됨
  - ▶ 즉, expansion을 신중하게 해서 tree가 너무 커지지 않도록
- ❷  $s'$ 의 다음에 둘 각 수  $a'$ 에 대해 edge  $(s', a')$ 의 정보를 초기화
  - ▶  $N_v(s', a') = N_r(s', a') = 0$
  - ▶  $W_v(s', a') = W_r(s', a') = 0$
  - ▶  $P(s', a') = p_\sigma(a'|s')$

## Step 3: Evaluation



### ❶ Value network(CNN)을 사용하여 $v_\theta(s')$ 를 계산

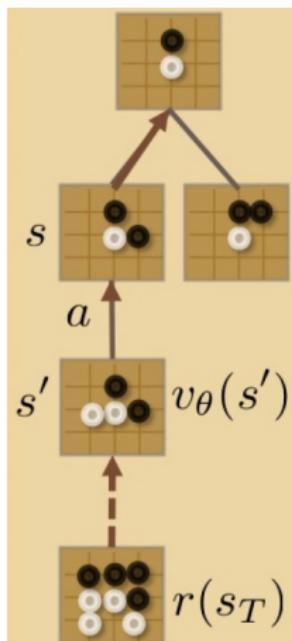
- ▶  $s'$ 에서 시작할 때 이길 확률 (Value network  $v_\theta$ 는 RL policy network  $p_\rho$ 를 이용하여 경기전 미리 training)
- ▶ 계산을 바로 하는 것이 아니고 GPU 계산 queue에 넣어두고 비동기 분산 계산 ( $s'$  생성하면서 시작)
- ▶ 한번 계산된  $v_\theta(s')$ 는 memoize (자주 등장될테니)
- ▶ 계산 시간: 평균 3ms

### ❷ Fast roll-out 시뮬레이션을 반복하여 승률 계산

- ▶  $s'$ 에서 시작하여 빠른( $0.2\mu s$ ) policy network인  $p_\pi$ 를 이용하여 승패 판단 (c.f. 일반 MCTS는 랜덤하게)
- ▶ 이기면  $r$ 값은 1, 지면 -1
- ▶ 시간을 단축하기 위해 끝까지 가지 않고 적절히 간 후 value network 값을 사용할 수도 있음

### ❸ 두 값을 backup 단계에서 가중합해서 Q에 반영

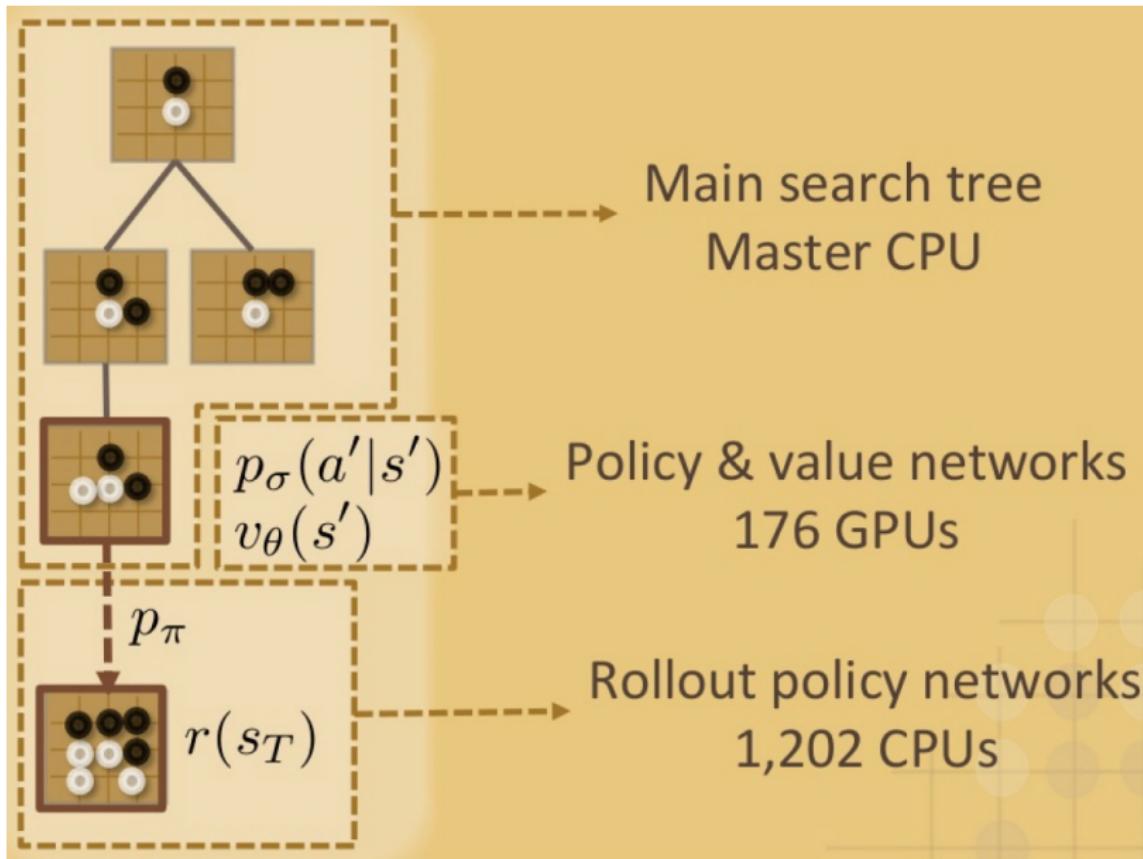
## Step 4: Backup



$s'$ 에서 루트까지 거꾸로 올라가면서 각 edge의 정보들을 update:

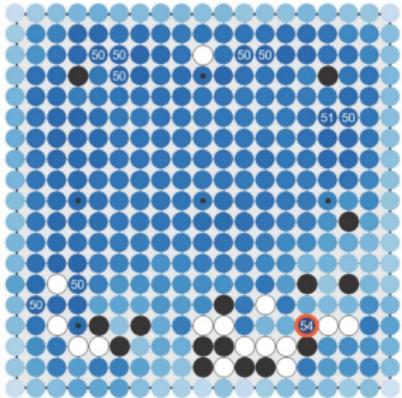
- $N_r(s, a) += 1$
- $N_v(s, a) += 1$
- $W_r(s, a) += r(s_T)$  (보통은 ±1)
- $W_v(s, a) += v_\theta(s')$
- $Q(s, a) = (1 - \lambda) \frac{W_v(s, a)}{N_v(s, a)} + \lambda \frac{W_r(s, a)}{N_r(s, a)}$ 
  - ▶  $\lambda = 0.5$ 로 할 때 실험적으로 성능이 가장 좋음

## Asynchronous Distributed Search

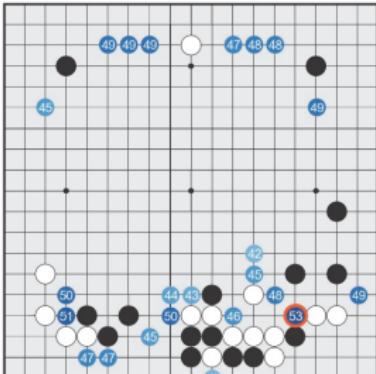


# Search Example

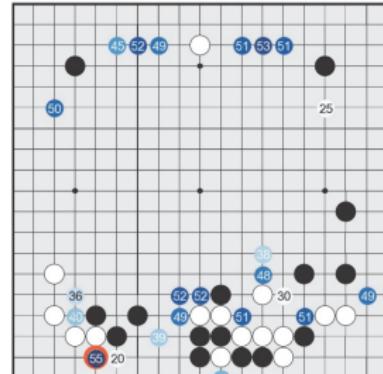
**a** Value network



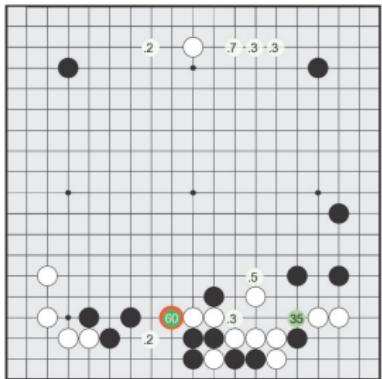
**b** Tree evaluation from value net



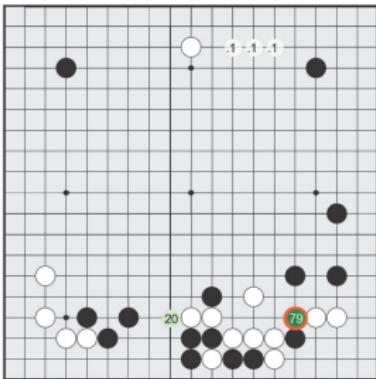
**c** Tree evaluation from rollouts



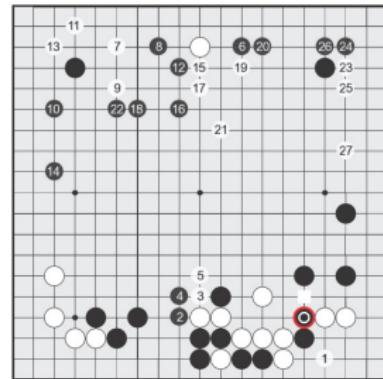
**d** Policy network



**e** Percentage of simulations



**f** Principal variation



MCTS for AlphaGo  
oooooooooooo

Policy/Value Networks  
oooooooooooooooooooo

AlphaGo Zero  
ooooooo

# Outline

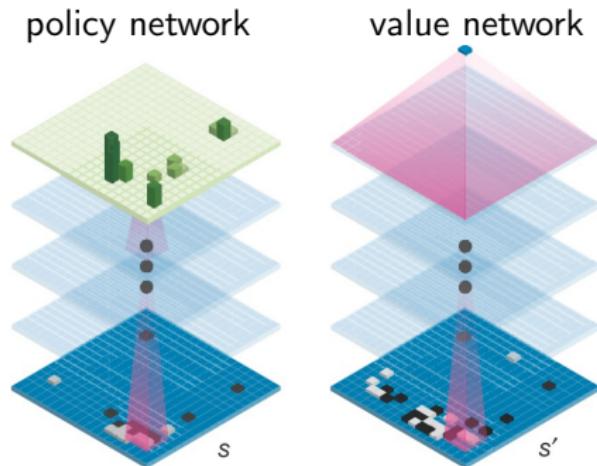
1 MCTS for AlphaGo

2 Policy/Value Networks

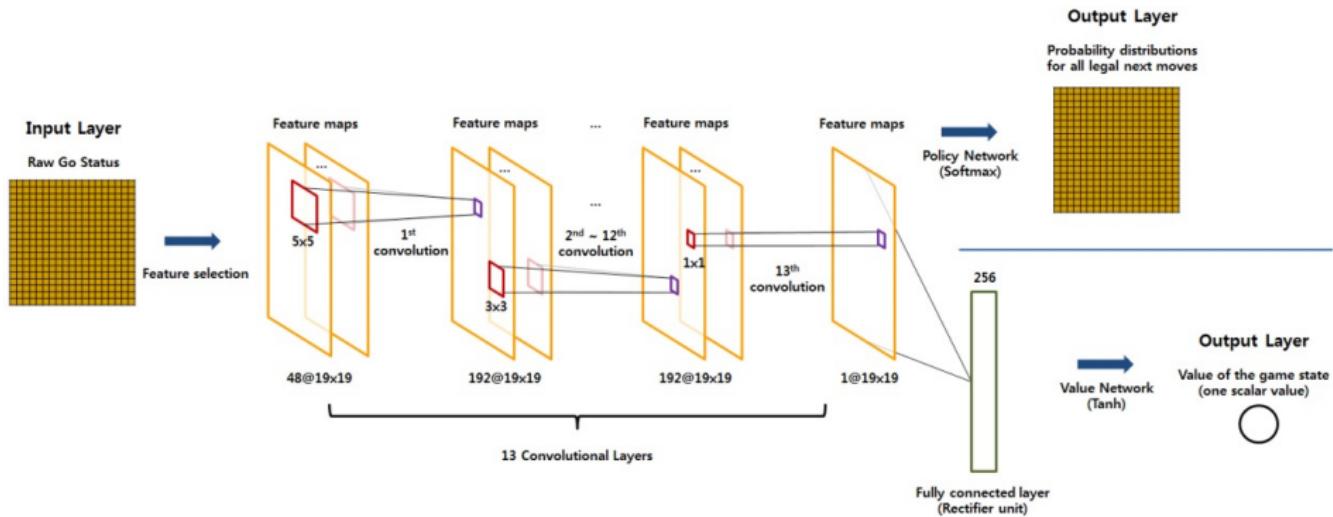
3 AlphaGo Zero

## Recall: Policy/Value Networks for MCTS

- **Policy network:**  $p(s) = "s\text{에서 최적의 다음 수}"$  (softmax)
  - ▶  $p_\sigma$ : (아마추어) 인간의 기보로 학습 (supervised)
  - ▶  $p_\pi$ :  $p_\sigma$ 를 단순화한 빠른 버전 (supervised)
  - ▶  $p_\rho$ :  $p_\sigma$  간의 자가대결 기보로 학습 (reinforced)
- **Value network:**  $v_\theta(s) = "s\text{로 시작하면 이길 확률}"$ 
  - ▶  $p_\rho$  간의 자가대결 결과를 입력 데이터로 학습 (supervised)



# CNN Architecture for Policy/Value Networks



- 13개의 convolution-pooling layer
- 마지막 full-connection 층을 제외하고 policy network와 value network 구조가 거의 같음 (출력 값/차원이 틀리므로 당연)

## Input Data for CNNs

19×19 판 위의 11종류, 48개의 feature를 preprocessing해서  
입력으로

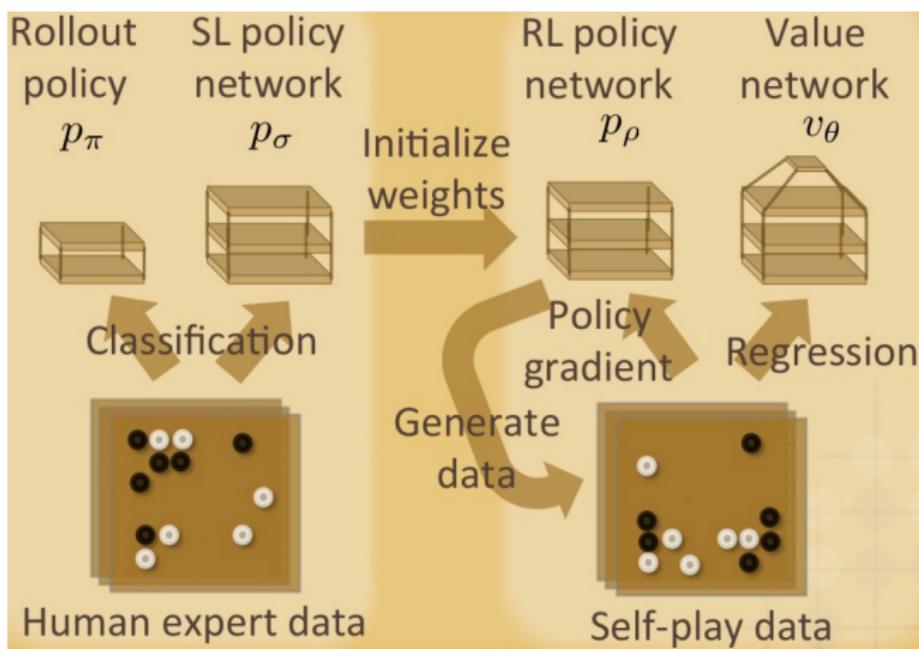
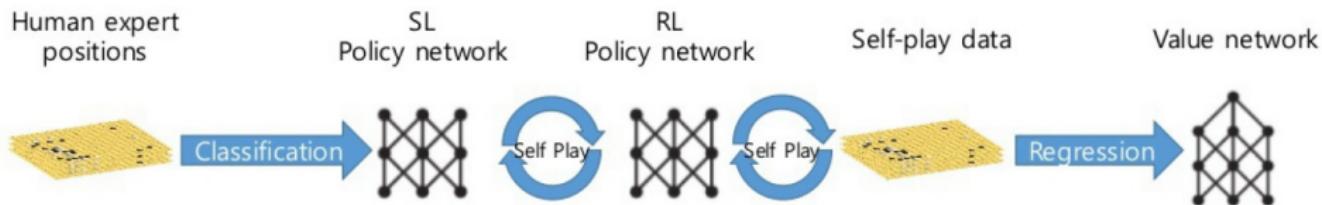
- 돌의 색깔 (player/opponent/empty) 3개
- 현재 턴 8개
- 주위 빈자리 정보 (liberties) 8개
- 내가 잡은 돌의 수 8개
- 상대가 잡은 돌의 수 8개
- 착수한 후 주위 빈자리 정보 (liberties after move) 8개
- 죽 성사 여부 1개
- 죽 탈출 여부 1개
- 착수 가능 여부 1개
- 내가 짠 돌의 색 1개
  - ▶ 흑돌인 경우 상수 1로 채워짐

즉, 19×19×48 0/1-텐서로 표현

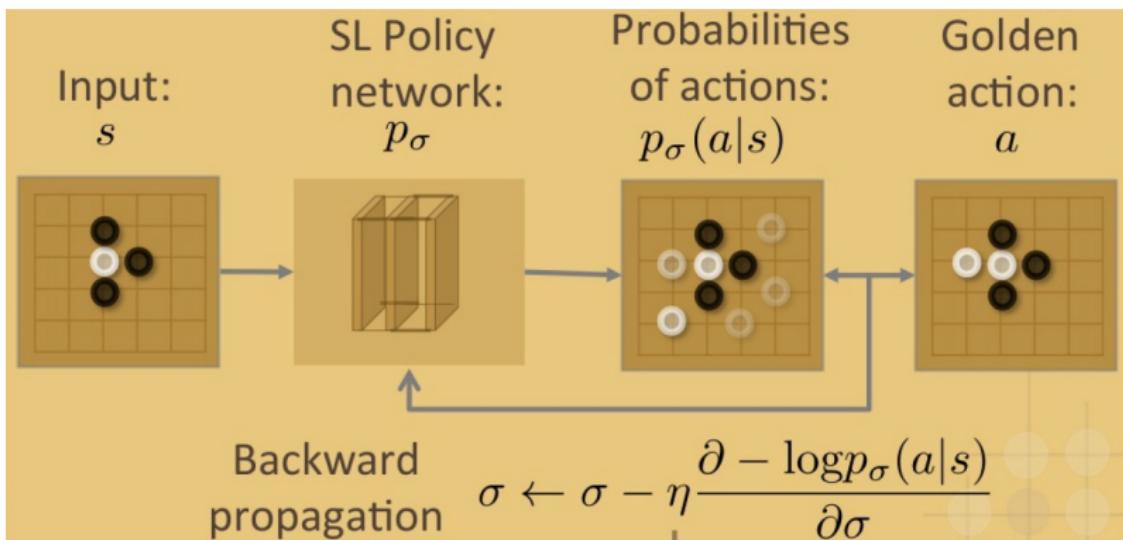
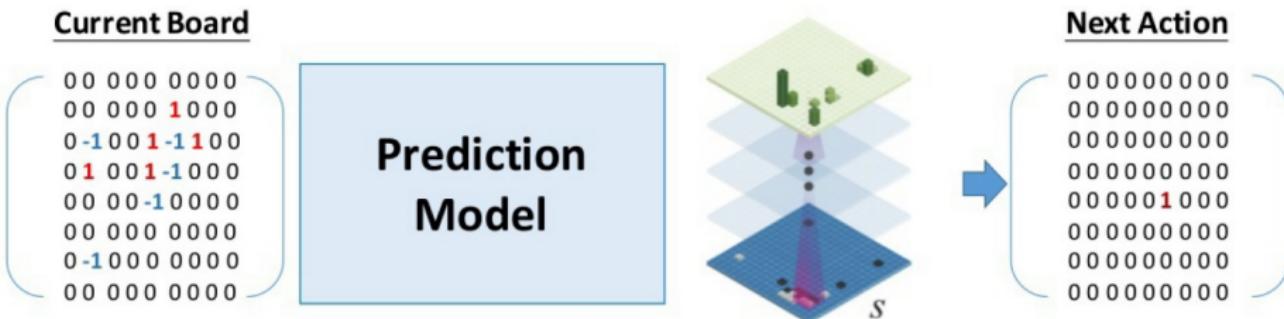
## Correlation bet'n Policy Network and Value Network

- Policy net: exact game solver의 **winning strategy**에 대응됨
- Value net: exact game solver의 **winning/losing position** 판단에 대응
- 따라서 policy net과 value net은 **독립이 아님**
  - ▶ 정확한 policy net이 있으면, 이걸 사용하면 정확한 value net을 구할 수 있음
  - ▶ 정확한 value net이 있으면, 이걸 사용하면 정확한 policy net을 구할 수 있음
- 알파고에서 두 종류의 network를 구분한 이유는 MCTS 휴리스틱에 더 맞추기 쉽고 계산하기 **편해서**
- 서로 상관관계가 있음을 이용하여, policy network 종  $p_p$ (강화학습 버전)으로 value network 훈련을 위한 **입력 데이터**를 생성

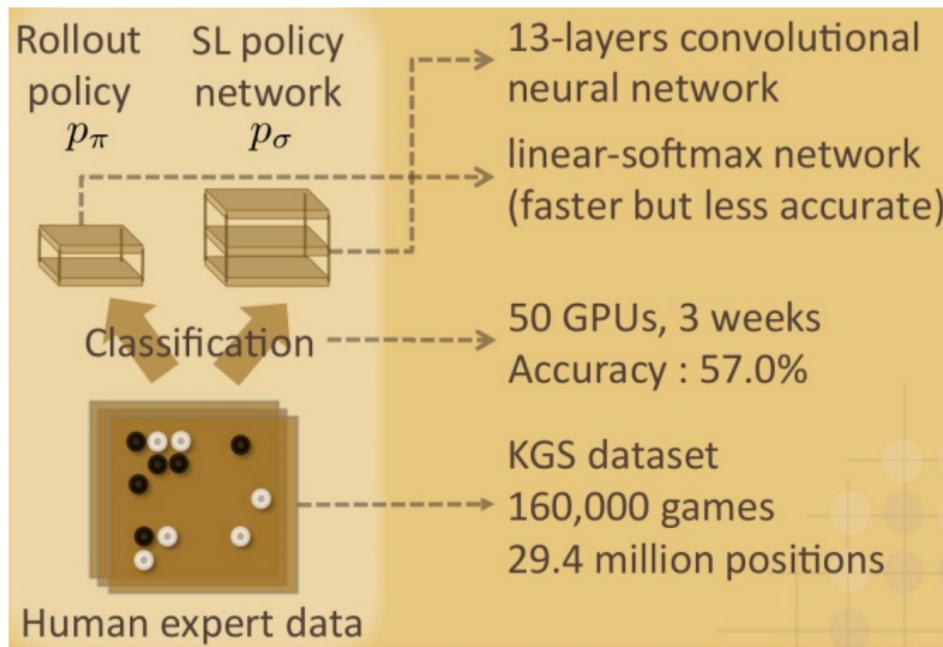
# Training Pipeline



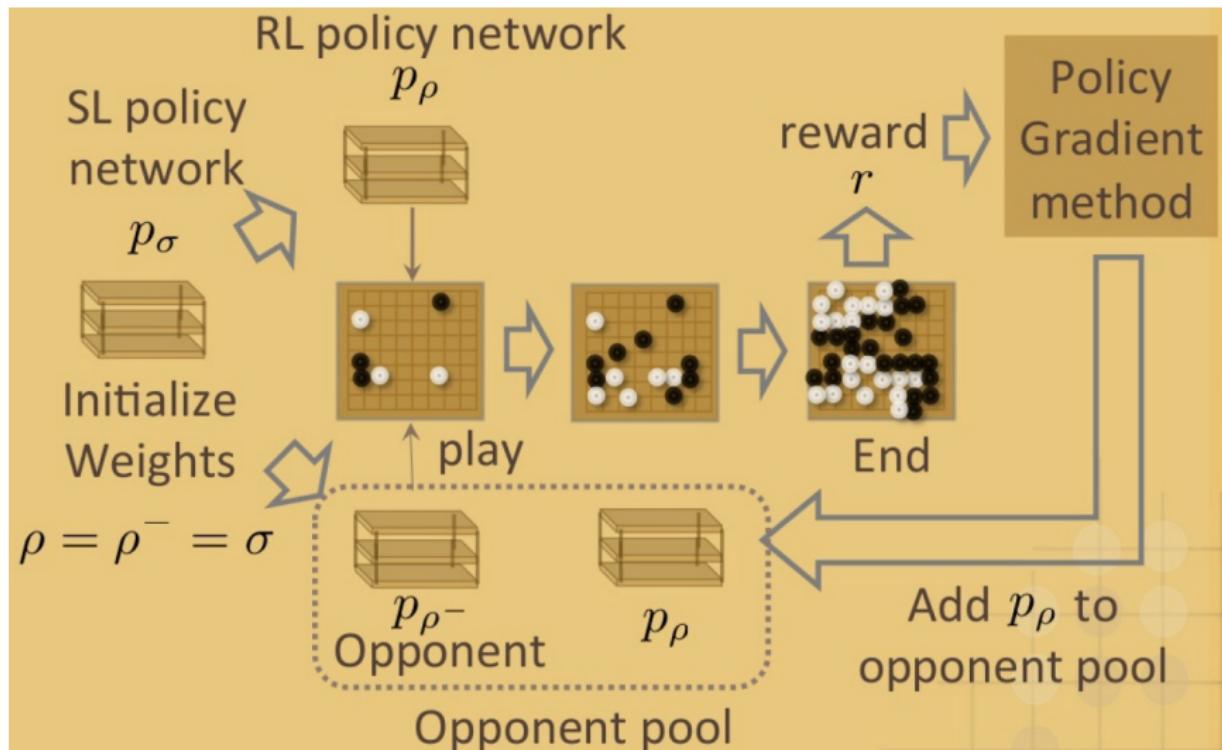
# Training SL Policy Network $p_\sigma$ with Human Record



## SL Policy Network $p_\sigma$ vs. Fast Version $p_\pi$



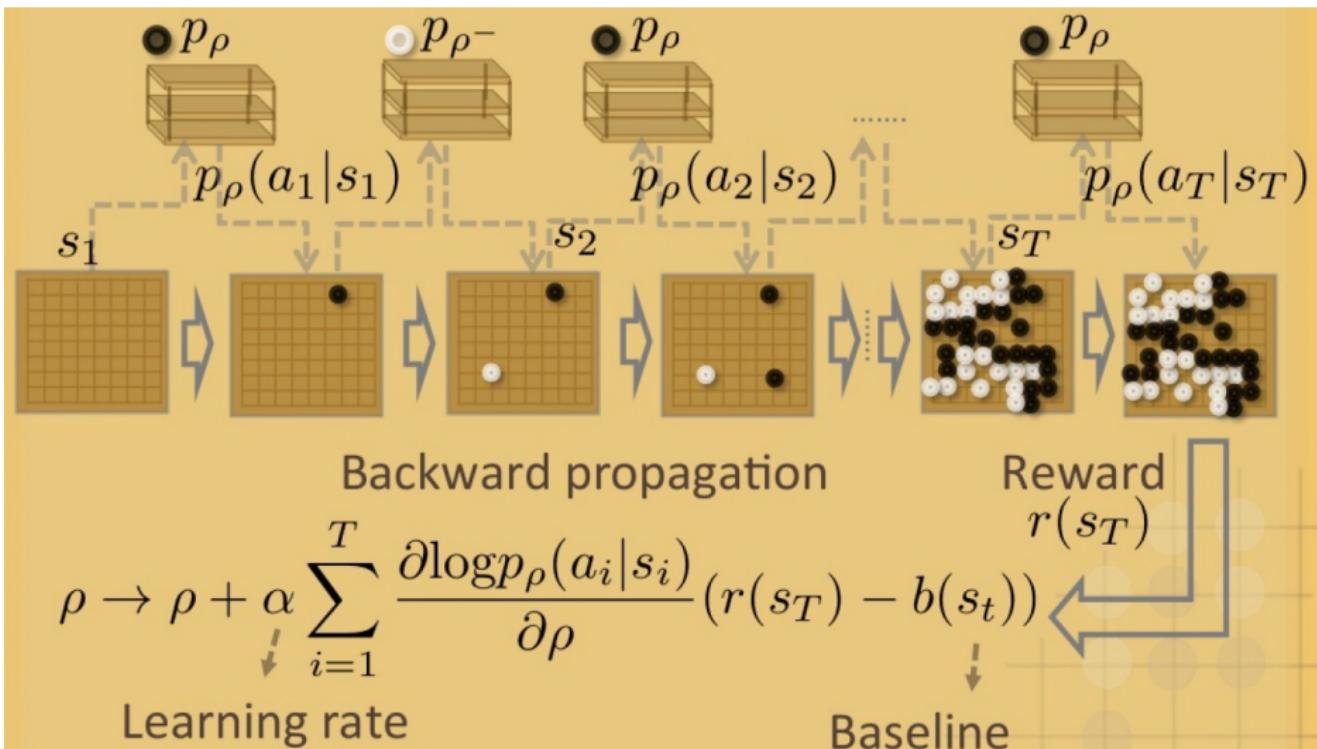
- $p_\pi$  : MCTS의 evaluation 스텝에서 fast roll-out에 사용
- $p_\sigma$  : MCTS의 selection/expansion step에서 탐색해볼 다음 수를 결정하기 위해 사용
  - ▶ 후술된 RL policy network  $p_\rho$ 의 초기화에도 사용

Training RL Policy Network  $p_\rho$  with  $p_\sigma$  and Reinforcement Learning

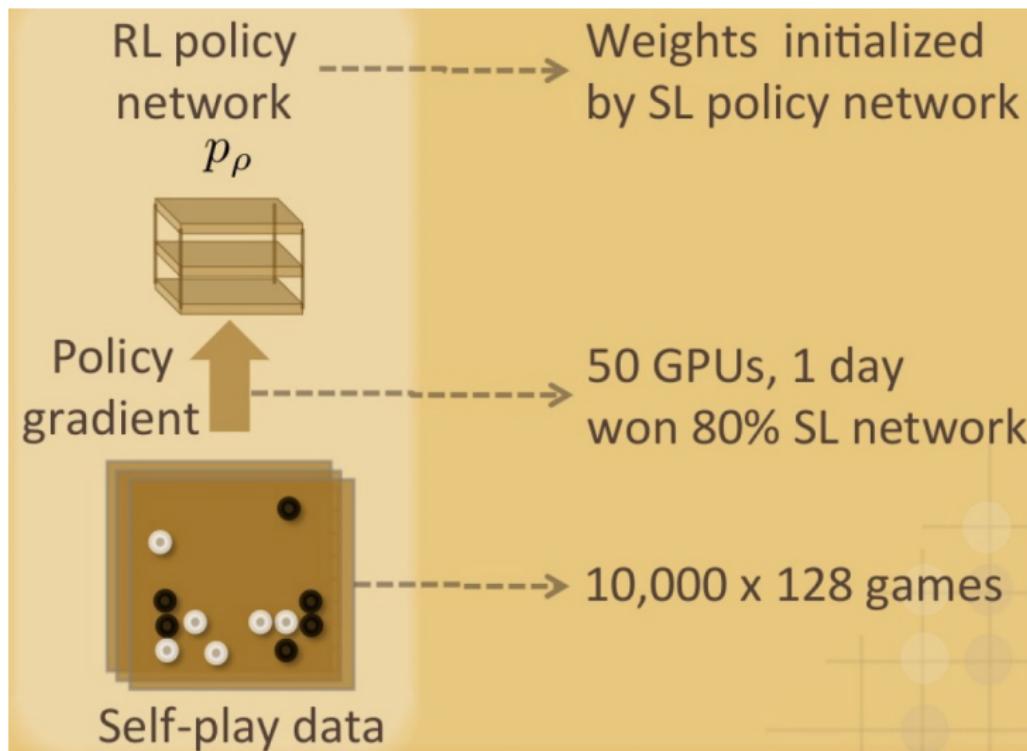
- SL policy network  $p_\sigma$ 의 weight로 초기화 후 자가대국을 거치면서 승패결과를 이용해 weight 조정 (진화?)

Training RL Policy Network  $p_\rho$  with  $p_\sigma$  and Reinforcement Learning

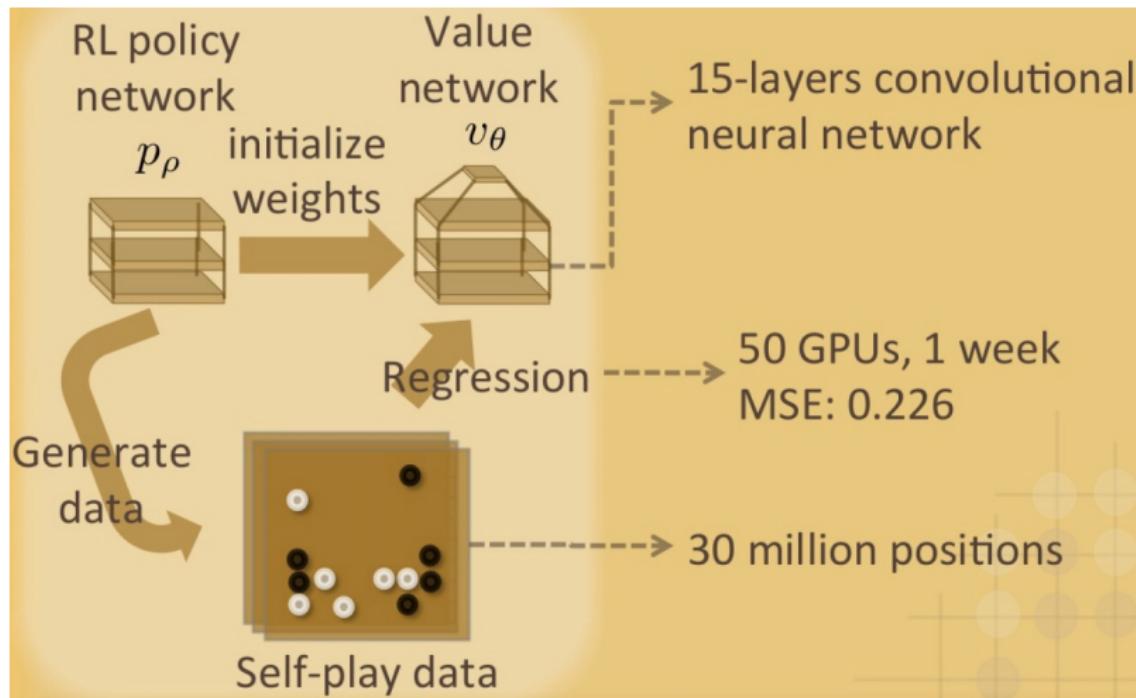
policy gradient method



## Training RL Policy Network $p_\rho$ with $p_\sigma$ and Reinforcement Learning

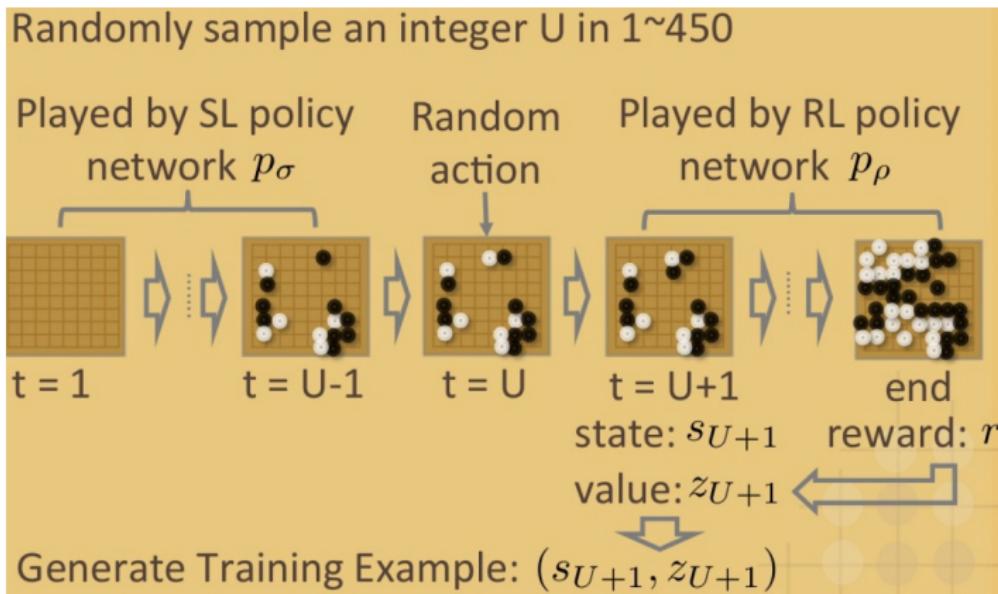


## Training Value Network $v_\theta$ with $p_\rho$



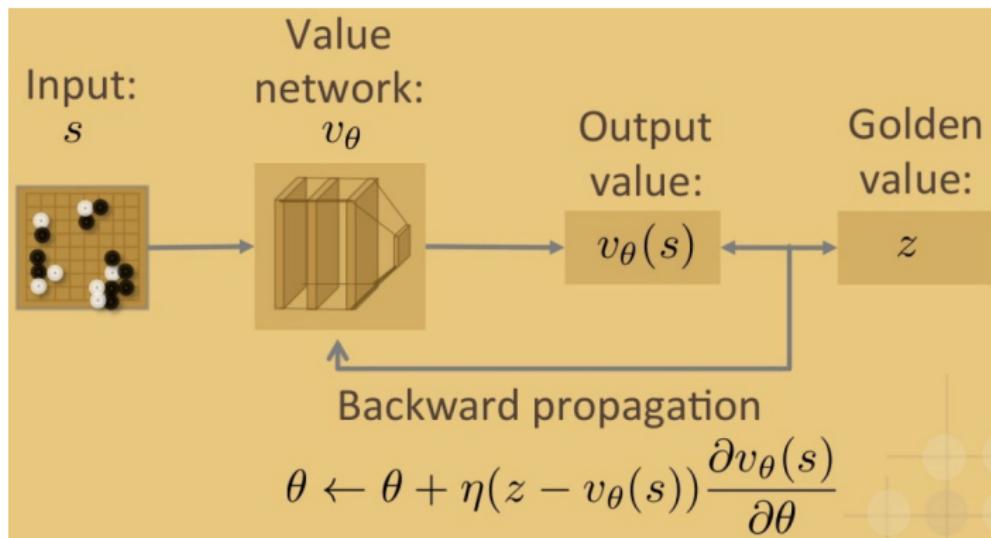
- RL policy network  $p_\rho$ 의 자가대국 결과를 입력 데이터로 이용해 value network  $v_\theta$ 를 supervised learning

## Training Value Network $v_\theta$ with $p_\rho$



- RL policy network  $p_\rho$ 의 자가대국 결과를 생성하는 과정
  - ▶ 인간의 경기 시작 방식 흉내는 SL policy network  $p_\sigma$ 가 더 잘나므로 초기  $U-1$ 수는  $p_\sigma$ 를 사용
  - ▶  $p_\sigma$ 을 MCTS의 selection/expansion에  $p_\rho$  대신 사용한 것도 유사한 이유 (SL policy: 뒤를 생각한 좀 더 global / RL policy: 그 순간의 최고의 move)

## Training Value Network $v_\theta$ with $p_\rho$



# Policy/Value Networks: Summary

빠른 네트워크. 몬테카를로  
를 아웃을 생성하는데 사용

시뮬레이션 게임 네트워크. 평가 네트워크에  
사용할 학습 데이터셋을 생성하는데 사용

평가 네트워크. 현재  
판세를 평가하는데 사용

충실향 네트워크. 몬테카를로 트리 검색의  
확장 단계에서 확률 값을 계산하는데 사용

RL policy network

Value network

$p_\pi$   
24.2% correct  
prediction, 2μs



$p_\sigma$   
57.0%, 3ms



+80% Winning Rate for SL

$p_\rho$

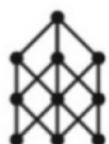
$v_\theta$

Prediction GAP = 0.234

Neural network

Policy gradient

Self Play



최종  $p_\rho$  대 최종  $p_\rho$  간의  
게임 데이터로 학습.  
승률에 대한 회귀 분석

$$\Delta\theta \approx \frac{\partial v_\theta(s)}{\partial \theta} (z - v_\theta(s))$$

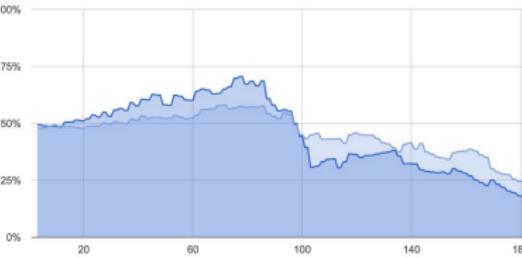
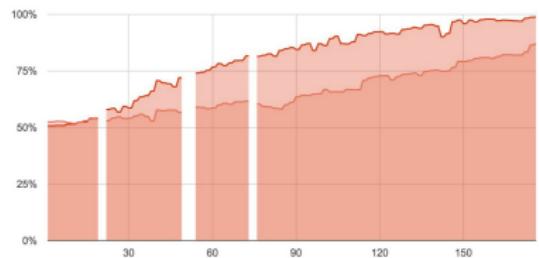
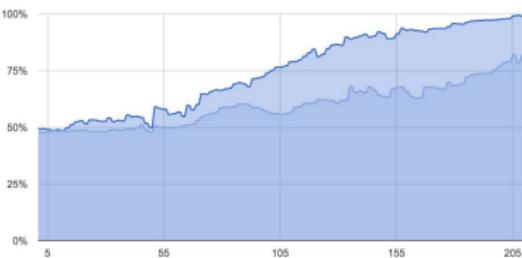
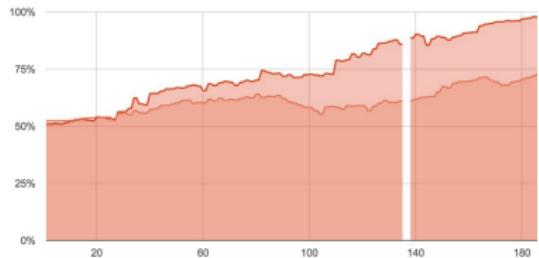
Human expert positions

KGS Go Server의 6단 ~ 9단  
게임의 판세 데이터  
3000만개를 사용하여 학습

Self-play positions

최종  $p_\sigma$  대 중간  $p_\sigma$  간의 게임 데이터를  
사용하여 게임 결과에 따라 +/-  
피드백으로 반영하면서 학습

## Prediction During 5 Games with Lee Sedol (value network)



MCTS for AlphaGo  
oooooooooooo

Policy/Value Networks  
oooooooooooooooooooo

AlphaGo Zero  
ooooooo

# Outline

1 MCTS for AlphaGo

2 Policy/Value Networks

3 AlphaGo Zero

## Recall: Simple-Go

Simple-Go 현재 버전

- 훈련 데이터를 뽑기 위한 경기: value net (with randomness)
  - ▶  $g-1$ 세대 value net로  $g$ 세대 value net 학습
- 실제 경기: value net (without randomness)

Simple-Go 속제 버전 (your job) ( $\approx$  [ALPHAGO 2015](#))

- 훈련 데이터를 뽑기 위한 경기: value net (위와 같음)
- 실제 경기: value net + [MCTS/α-β](#)

Simple-Go 궁극적 희망 버전 ( $\approx$  [ALPHAGO ZERO](#))

- 훈련 데이터를 뽑기 위한 경기: value net + [MCTS/α-β](#)
- 실제 경기: value net + [MCTS/α-β](#)

## ALPHAGO versions

### ALPHAGO FAN (판파고, Nature 2015 논문)

- 훈련 데이터를 뽑기 위한 경기: SL/RL policy net
  - ▶  $g-1$ 세대 RL policy  $\Rightarrow$   $g$ 세대 RL policy + value nets 학습
- 실제 경기: SL/rollout policy nets + value net + MCTS

### ALPHAGO LEE (돌파고)

- 훈련 pipeline은 거의 같음 (value net 학습만 약간 다름)
- CNN 규모가 커지고 ( $192 \Rightarrow 256$ ) GPU 대신 TPU 도입

### ALPHAGO MASTER (마파고, 道場擊破 (온라인, 중국 기원))

- ALPHAGO ZERO와 거의 같음. 인간 기보로 초기화 하는 것과 48개 feature를 뽑아내는 것은 돌파고와 같음

### ALPHAGO ZERO (Nature 2017 논문)

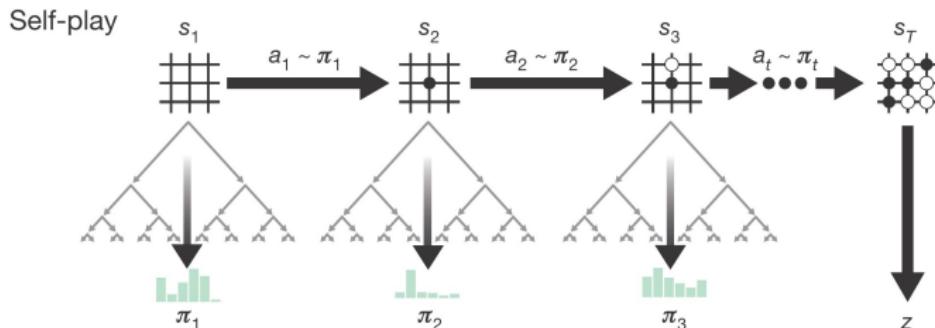
- 훈련 데이터용 경기: unified net + MCTS (w/o fast rollouts)
- 인간 기보로 초기화 안하고, 흑/백/empty feature만 사용

## ALPHAGO ZERO

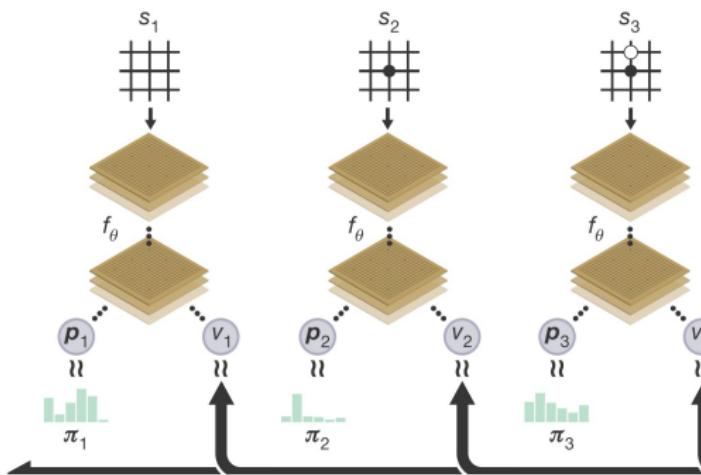
- 인간 기보로 CNN 초기화 하지 않음 (*tabula rasa*)
- Policy network  $p$ 와 value network  $v$ 을 하나의 CNN  $f_\theta$ 로
  - ▶  $p(s, a) = "s$ 에  $a$ 에 두는 것이 최적일 확률"
  - ▶  $v(s) = "s$ 로 시작하면 이길 확률"
  - ▶  $(p, v) = f_\theta$  ( $\theta$ 는 CNN의 weights)
  - ▶ residual block 형태의 CNN으로 구현하여 학습 성능 향상
- 훈련 데이터 생성용 경기에도 MCTS 사용 (next slide)
  - ▶ 훈련용 데이터: MCTS로 evaluation한  $\pi$  + winner 정보  $z$
  - ▶  $(p, v) = f_\theta$  가  $(\pi, z)$ 에 최대한 가까워 지도록  $\theta$ 를 학습:
$$\text{loss} = (z - v)^2 + \pi^\top \log p + c\|\theta\|^2$$
- 48개의 feature 대신 흑/백/empty 3개 feature만 사용
- MCTS에서 rollout을 사용 안하고 value net만 사용

# Training Pipeline for ALPHAGO ZERO

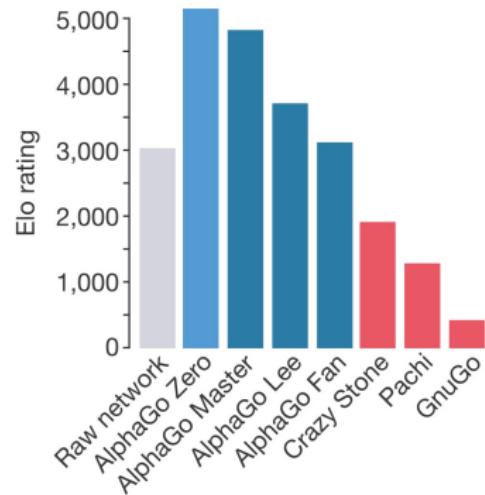
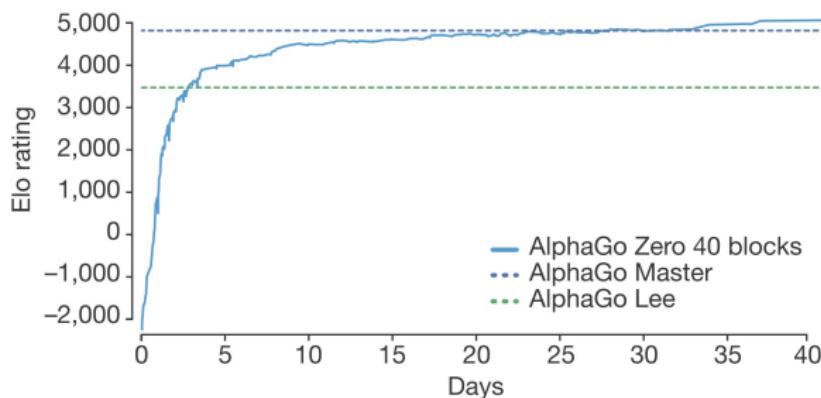
(c.f. policy iteration in RL)



## Neural network training



## Performance of ALPHAGO ZERO



- 40일간 총 2천9백만 게임으로 강화학습 (40 residual blocks)
- 4일차 정도에 돌파고, 33일차 정도에 마파고 주월
- ELO: Zero = 5,185, 마파고 = 4,858, 돌파고 = 3,739
  - ▶ ELO 2000| 높으면 승률0| 75% 정도

## ALPHAZERO for Chess/Shogi

- 체스에서는 MCTS가 매우 비효율적으로 알려져서  $\alpha$ - $\beta$ 를 사용해옴
  - ▶ 바둑과 달리 “trap state” 가 많아 “넓게” 탐색해야 함
  - ▶ 딥블루:  $\alpha$ - $\beta$  with depth 12
- 그럼에도 불구하고, MCTS에 기반한 ALPHAGO ZERO의 방식을 체스에 그대로 적용해도 챔피언을 추월
  - ▶ 2016 TEC 챔피언 STOCKFISH를 4시간 학습만으로 추월
- 쇼기(일본식 장기)에 적용해도 챔피언을 추월
  - ▶ CSA 챔피언 ELMO를 2시간 학습만으로 추월
- MCTS에  $\alpha$ - $\beta$ 를 이용하여 보완하면 더욱 성능이 향상될 것으로 예상