AlphaGo Zero Paper Review

[쉽게 읽는 강화학습 논문 2화]

팡요랩

2018.11.18

ARTICLE

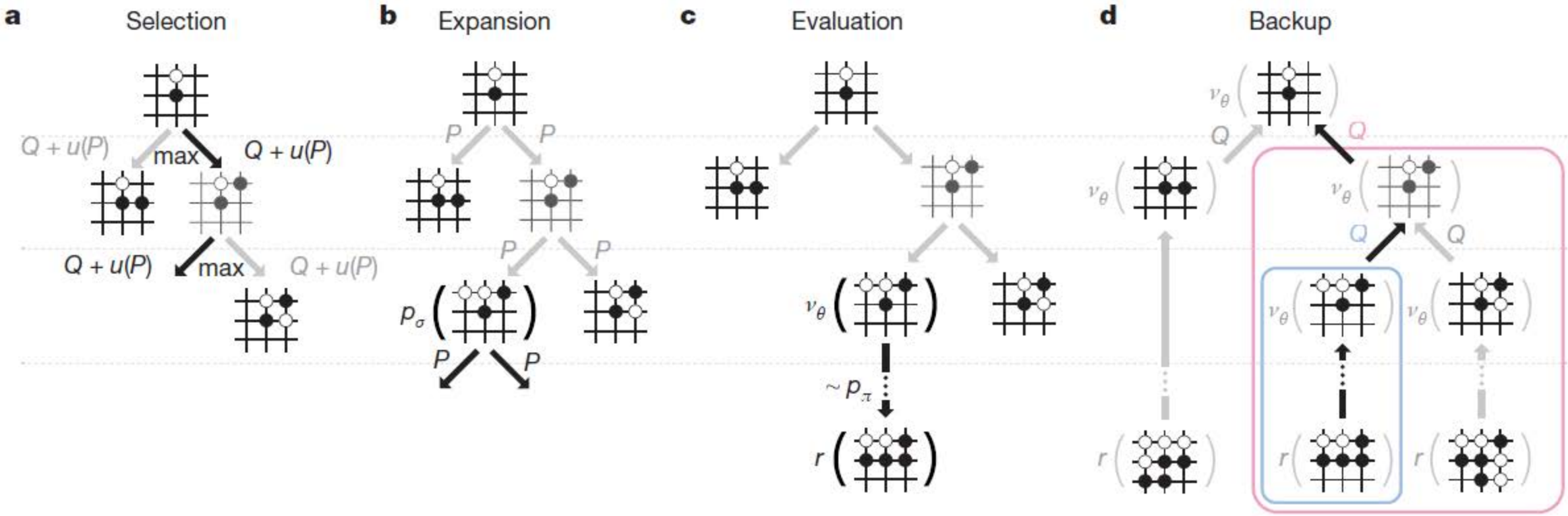
doi:10.1038/nature24270

Mastering the game of Go without human knowledge

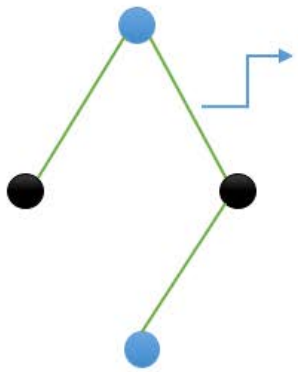
David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

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.

MCTS 복습


a
Selection
Q + u(P)
max
Q + u(P)
Q + u(P)
max
Q + u(P)
b
Expansion
P
P
P
P
p_σ
P
P
c
Evaluation
v_θ
$\sim p_\pi$
r
d
Backup
v_θ
Q
Q
v_θ
v_θ
Q
Q
v_θ
v_θ
r
r
r

MCTS 복습



For each Edge...
Q(S,A) : Action Value
N(S,A) : Visit Count
P(S,A) : Prior probability

$$a_t = \underset{a}{\operatorname{argmax}} \left(Q(s_t, a) + u(s_t, a)\right)$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} \mathbf{1}(s,a,i) V(s_L^i)$$

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$$V(s_L) = (1-\lambda) v_\theta(s_L) + \lambda z_L$$

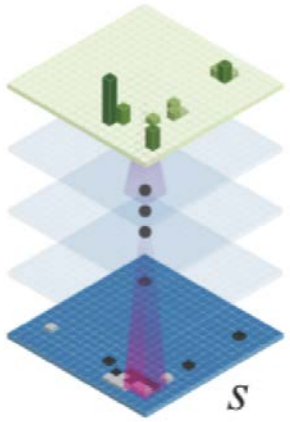
Value Network

롤아웃으로 게임을 끝까지 플레이 했을 때의 결과

AlphaGo

1. Rollout policy

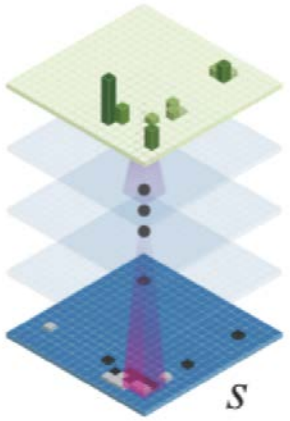
$p_{\sigma/\rho}(a|s)$



- Trained from 8 million positions (Data from tygem game server)
- Execute 1,000 simulations per second

2. SL policy

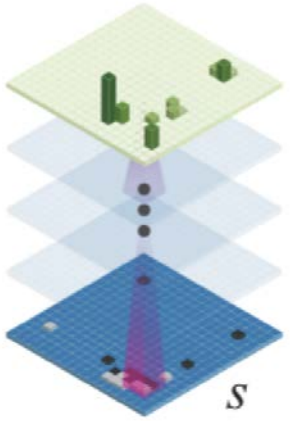
$p_{\sigma/\rho}(a|s)$



- Input :19*19*48
- 13 Layer Conv Network
- Trained from 30 million positions (Data from KGS Go server)

3. RL policy

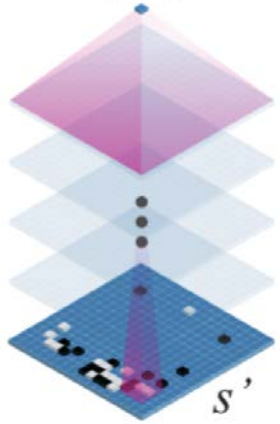
$p_{\sigma/\rho}(a|s)$



- Initialize with SL policy
- RL VS SL : RL 80% WIN

4. Value Network

$v_\theta(s')$

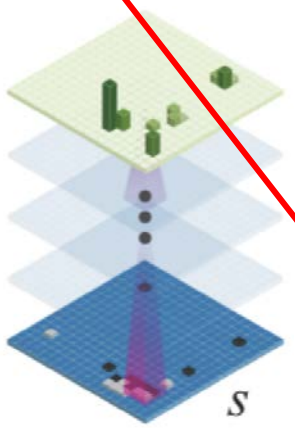


- Trained with RL policy

AlphaGo -> AlphaGo Zero

1. Rollout policy

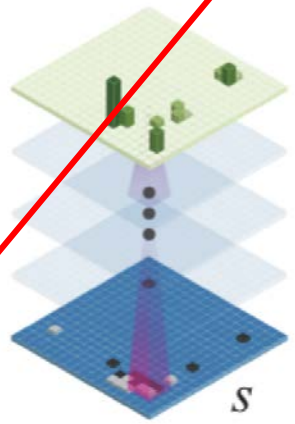
$p_{\sigma/\rho}(a|s)$



- Trained from 8 million positions (Data from tygem game server)
- Execute 1,000 simulations per second

2. SL policy

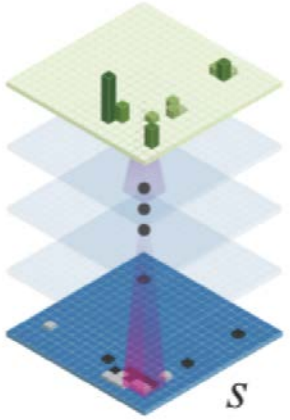
$p_{\sigma/\rho}(a|s)$



- Input :19*19*48
- 13 Layer Conv Network
- Trained from 30 million positions (Data from KGS Go server)

3. RL policy

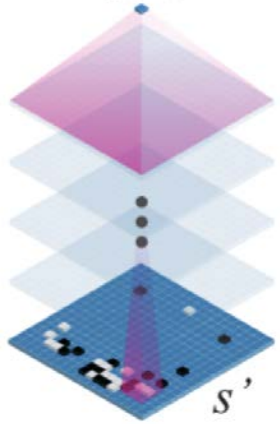
$p_{\sigma/\rho}(a|s)$



- Initialize with SL policy
- RL VS SL : RL 80% WIN

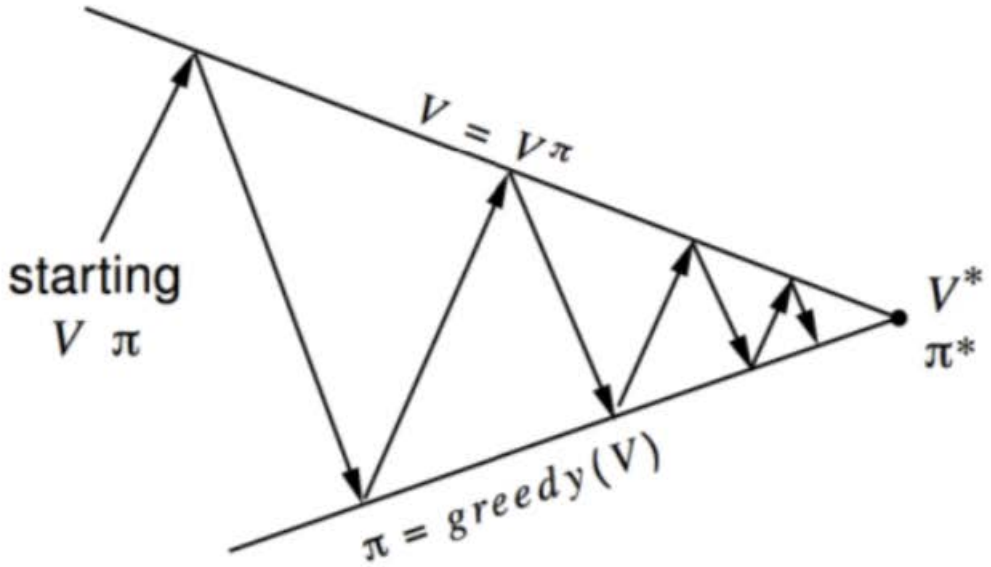
4. Value Network

$v_\theta(s')$



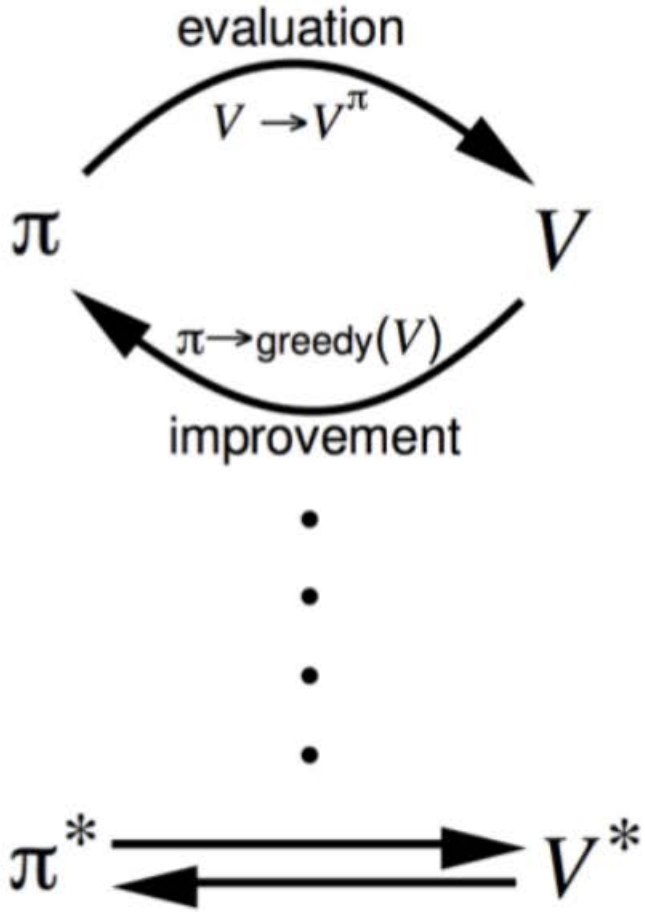
- Trained with RL policy

아이디어 – Policy Iteration !



Policy evaluation Estimate v_π
Iterative policy evaluation

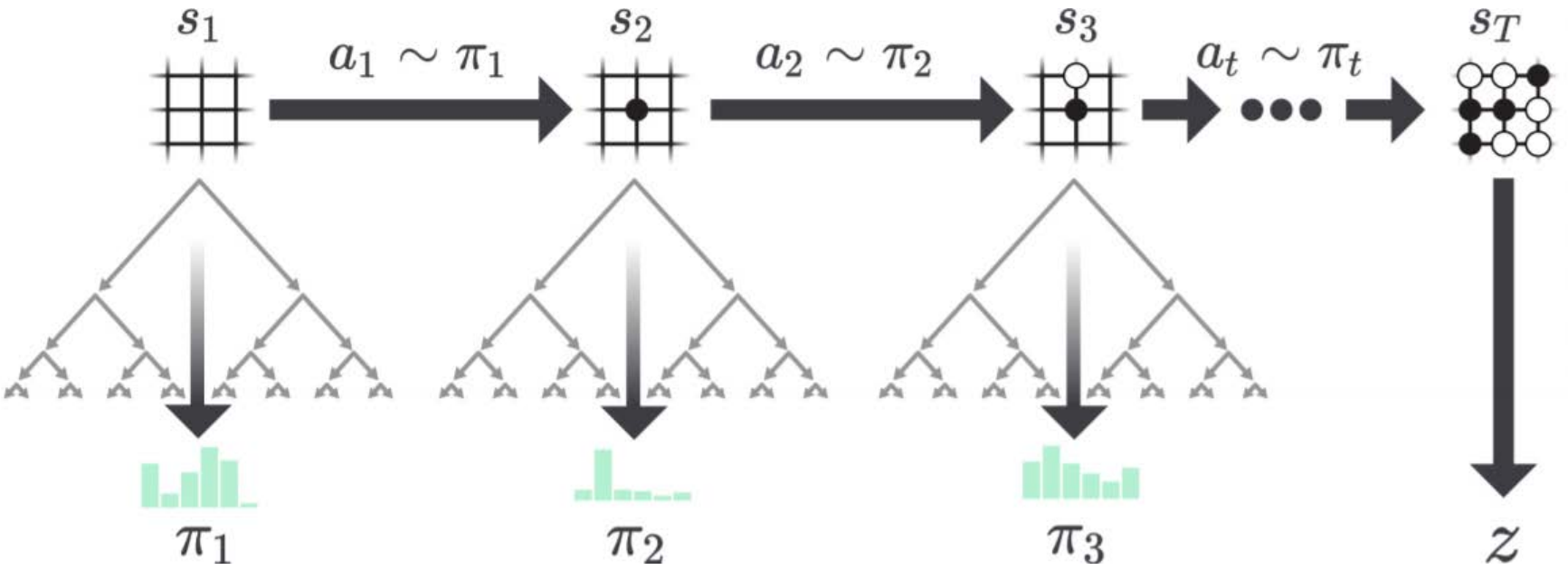
Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement



아이디어 – MCTS as an operator

In each position s, an MCTS search is executed, guided by the neural network f_θ. The MCTS search outputs probabilities $\boldsymbol{\pi}$ of playing each move. These search probabilities usually select much stronger moves than the raw move probabilities $\mathbf{p}$ of the neural network $f_\theta(s)$; MCTS may therefore be viewed as a powerful *policy improvement* operator[20,21]. Self-play with search – using the improved MCTS-based policy to select each move, then using the game winner z as a sample of the value – may be viewed as a powerful *policy evaluation* operator.

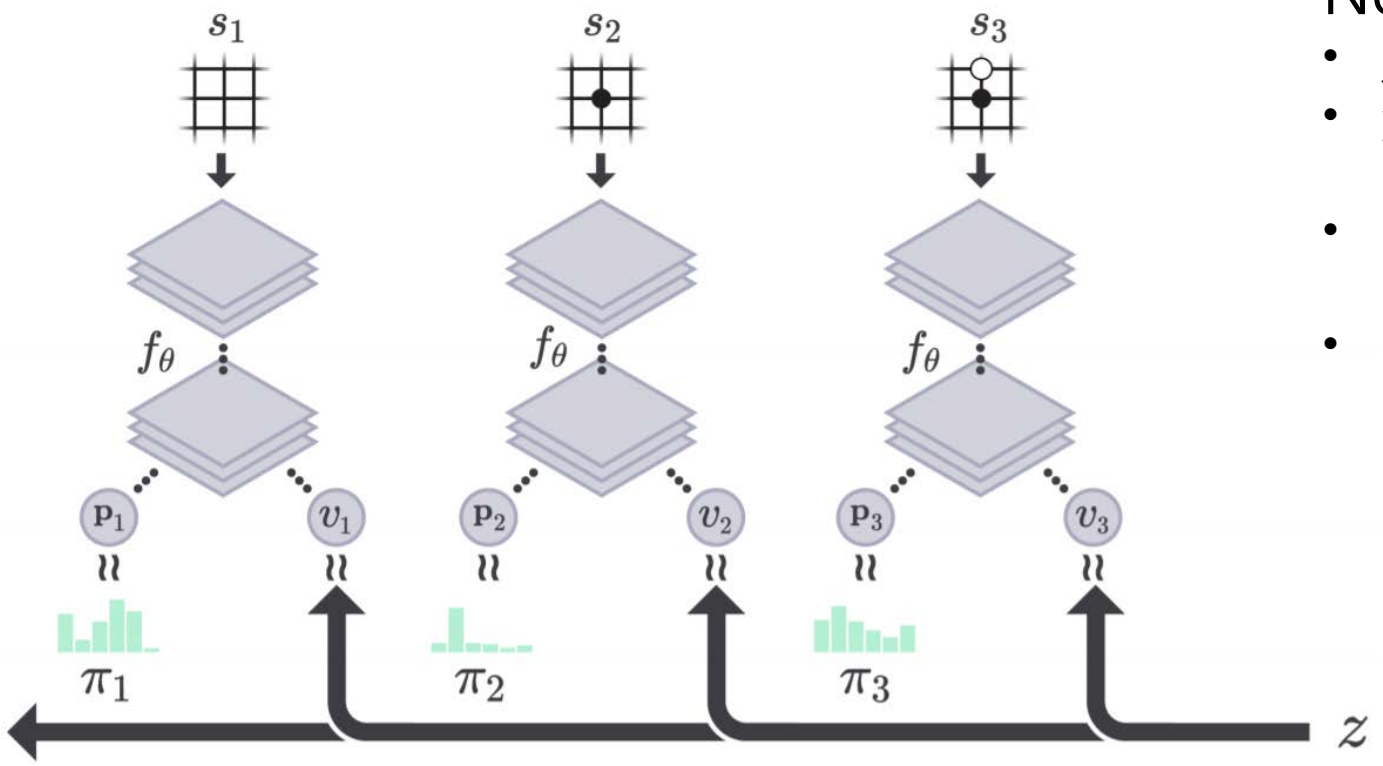
Methods



Self-Play

- 혼자서 번갈아가며 게임을 진행한다. 이때의 state는 $s_1, s_2, s_3, \ldots, s_T$
- 매 state 마다 MCTS를 통해$\boldsymbol{\pi}_t$를 계산하고, 샘플링 하여 action을 정한다.
- 게임이 끝나면 winner를 통해 z값을 정한다. 이기면 1, 지면 -1
- 결국 각 state 마다 $(s_t, \boldsymbol{\pi}_t, z_t)$ 가 남는다.

Methods

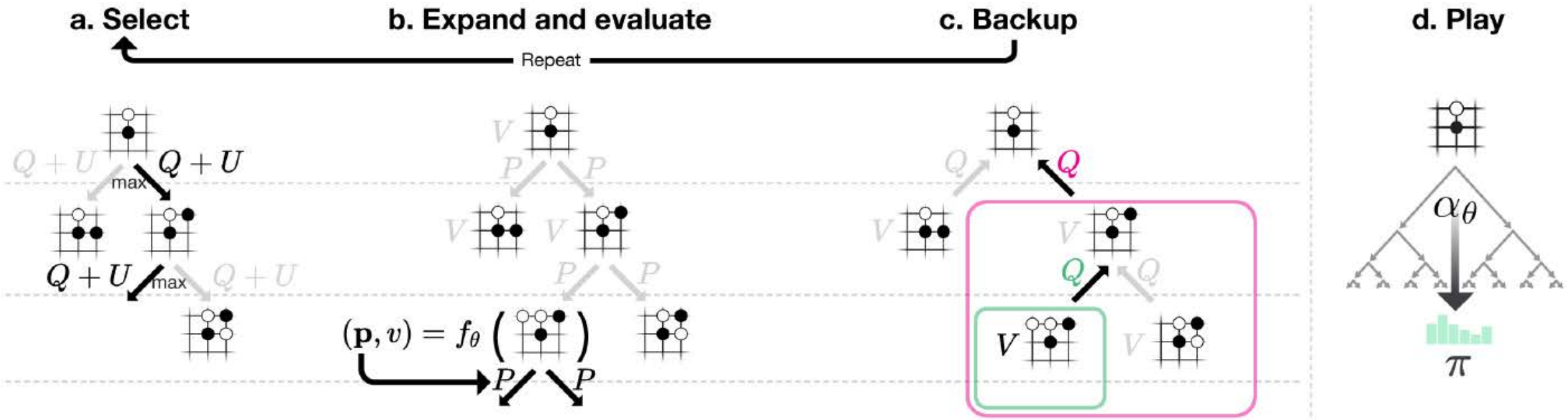


Neural Network Training

- $f_\theta(s) = (\boldsymbol{p}, v)$
- $\boldsymbol{p}_t$는 $\boldsymbol{\pi}_t$ 와 같아지도록, v 는 z와 같아지도록 θ 를 update.
- 데이터는 과거 게임에서 쌓은 $(s_t, \boldsymbol{\pi}_t, z_t)$ 중 random sampling 하여 사용.
- Loss 함수는 다음과 같다.

$$l = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c||\theta||^2$$

Methods



1. Selection 은 기존 방법과 유사하나 Q값의 구성이 바뀌었다.

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} \mathbf{1}(s,a,i) V(s_L^i)$$

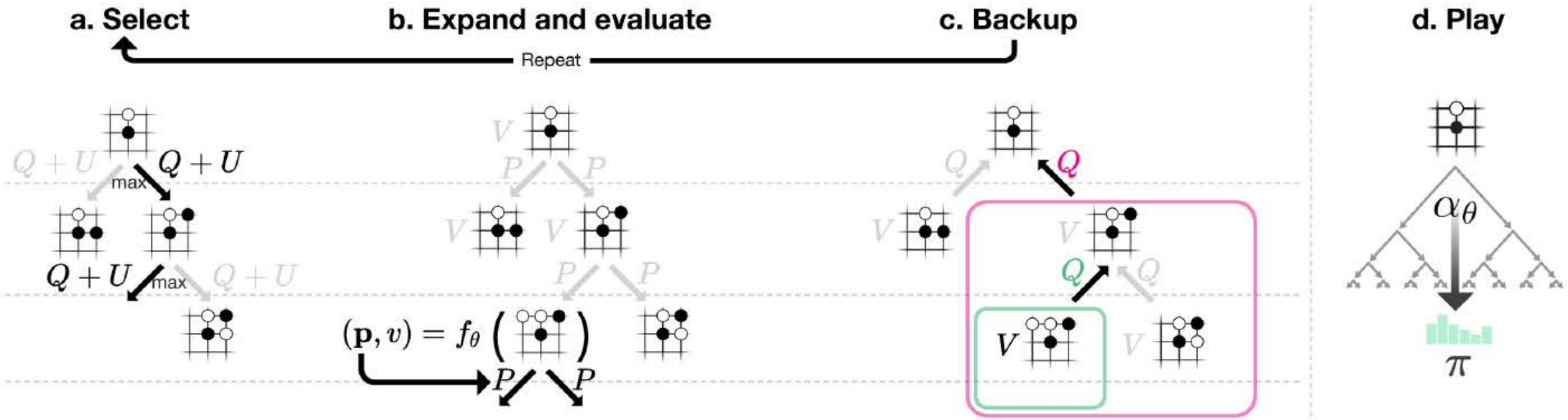
$$V(s_L) = (1-\lambda) v_\theta(s_L) + \lambda z_L$$

Value Network

롤아웃으로 게임을 끝까지 플레이 했을 때의 결과

$$\frac{\Sigma_{s\prime|s,a\to s\prime} V(s')}{N(s,a)}$$

Methods



2. Leaf Node에 도달하면 $f_\theta(s) = (\boldsymbol{p}, v)$ 를 evaluate 하여 prior와 Q를 계산
3. 이 계산된 값을 이용해 Q값을 update 해준다.
4. 1~3을 여러 번 반복하여 search 가 종료되면, 최종 확률 분포 $\boldsymbol{\pi}$ 가 리턴된다.
5. 여기서, $\boldsymbol{\pi}_a \propto N(s,a)^{1/\tau}$ (τ : temperature parameter)

Details – Self Play Reinforcement Learning

1. Optimisation
   - 64개의 GPU Worker가 쓰임
   - 16개의 CPU Parmater Server가 쓰임
   - mini-batch size는 2048 (worker 별 32)
   - 각 mini-batch는 최신 500,000 게임에서 uniform random 하게 sampling
   - 매 1000번의 update 마다 checkpoint 저장.
2. Evaluator
   - 새로 생긴 checkpoint는 기존 최고 네트워크와 붙어서, 55%이상의 승률을 보이면 기존 최고를 대체하여 data 생성 네트워크가 된다.
3. 각 Action 을 선택하기 위하여 1,600번의 simulation을 수행한다. (각 search 마다 0.4초 소요됨)
4. 각 게임에서, 초기 30번의 move 때는 τ 를 1로 설정한다. 이는 visit count에 proportional 하도록 move를 선택하게 해줘서 다양한 상황을 맞닥뜨리게 해준다.
5. 이후의 move에서는 τ -> 0으로 설정한다.
6. 한 action 때 생성된 subtree는 다음 action에서 재사용 된다.

Details - Neural Network Architecture

1. Input : 19 * 19 * 17

$$s_t = [X_t, Y_t, X_{t-1}, Y_{t-1}, ..., X_{t-7}, Y_{t-7}, C]$$

- X_t : t 시점에서 현재 플레이어의 돌의 위치정보
- Y_t : t 시점에서 상대방 플레이어의 돌의 위치정보
- C : 현재 플레이어가 흑인지 백인지 정보

2. Input은 한 개의 conv block 과 19 or 39개의 Residual block을 통과

Conv block

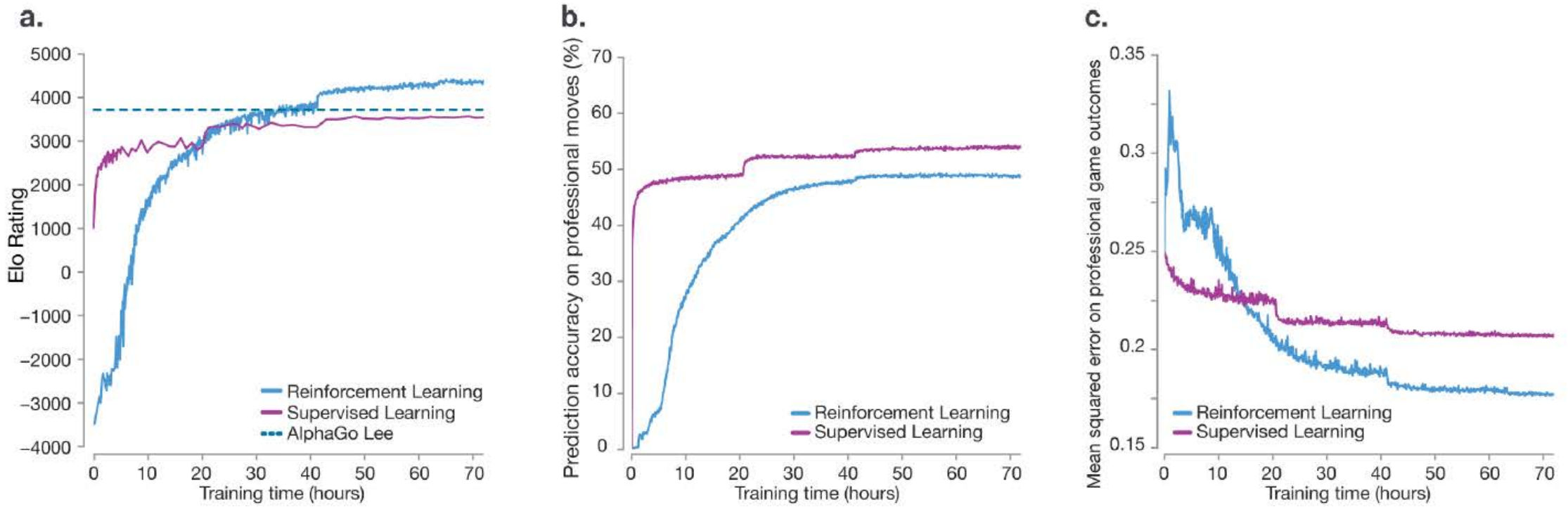
Conv (3*3 filter 256개)
Batch Normalization
Relu

Residual block

Conv (3*3 filter 256개)
Batch Normalization
Relu
Conv (3*3 filter 256개)
Batch Normalization
Skip Connection
Relu

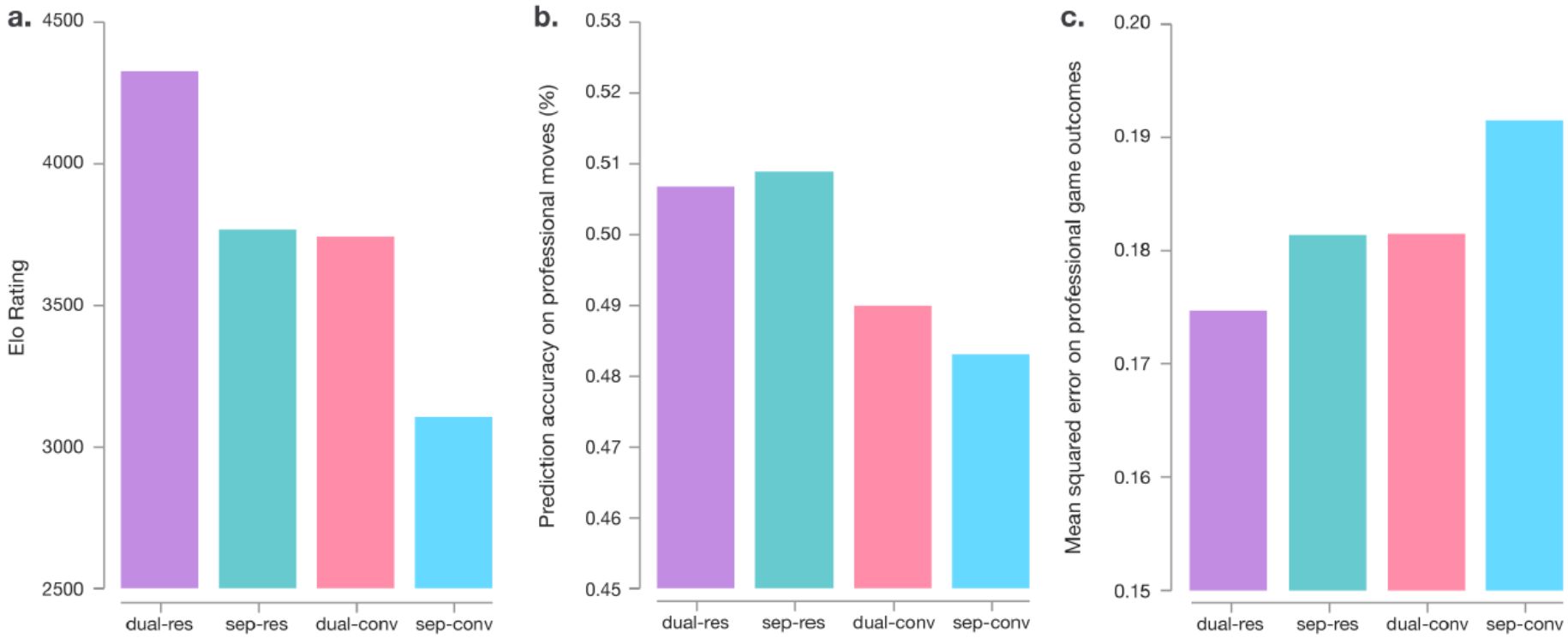
3. 최종적으로 value와 policy 는 각각 1차원 tanh와 362차원 softmax로 끝남

Empirical evaluation of AlphaGo Zero



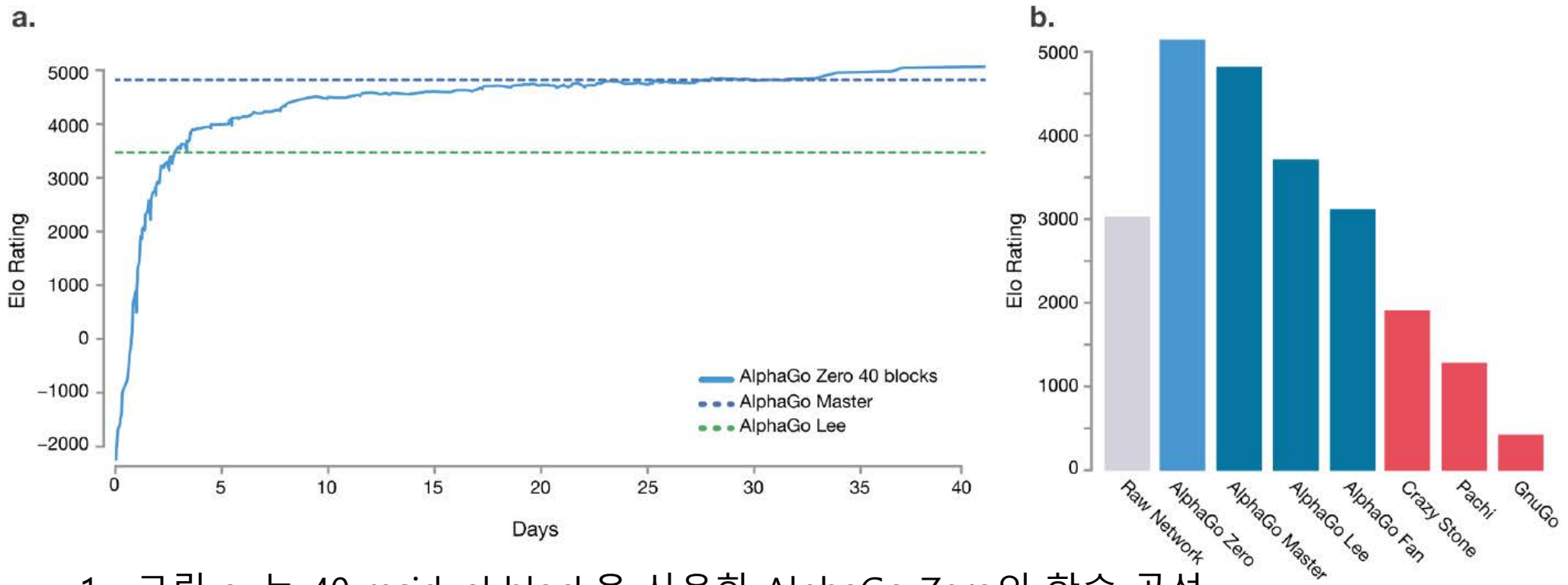
1. 비교를 위해 같은 Architecture 를 사용한 네트워크로 KGS data-set에 대해 Supervised Learning 을 함
2. ELO Rating 은 0.4초의 thinking time을 두고 실험함

Comparison of neural network architectures in AlphaGo Zero and AlphaGo Lee



1. "dual" : Value 와 Policy가 하나의 네트워크를 공유
   "sep" : Value 와 Policy가 각각의 네트워크를 갖고 있음
2. "res" : Residual network (Deep Residual Learning for Image Recognition, 2015)
   "conv" : Convolutional network
3. "dual-res" : Alphago- Zero
   "sep-conv" : Alphago - Lee

Performance of AlphaGo Zero



1. 그림 a. 는 40 residual block을 사용한 AlphaGo Zero의 학습 곡선
   - 여기서 ELO 점수는 0.4초의 thinking time을 두고 평가.
2. AlphaGo Zero의 최종 성능.
   - 기존의 AlphaGo들과, 다른 프로그램, 또 Raw Network가 포함된 토너먼트의 결과이다.
   - 5초의 thinking time을 두고 평가.
   - Raw Network는 확률이 가장 높은 선택지를 바로 선택하는 방식.
   - 200점 차이는 75%의 확률로 이긴다는 뜻

끝