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

DeepMind

Summary

AlphaGo

“MCTS” 로 찾고
+
“value network” 로 바둑판 평가
+
“policy network” 로 착수점 결정

Summary

알파고?

1. 전례없던 바둑 기사 수준의 실력을 가진 인공지능
2. DNN + Search Tree Combination
3. SL(supervised learning) + RL(reinforcement learning)의 조합을 통해서
   학습된 DNN을 기반으로 바둑에서의 효율적인 착수점 선택(action selection)과 바둑판 위치 평가(state evaluation) 함수를 개발함.

Motivation

perfect information의 게임들은 모두 승패를 결정하는 optimal value function v*(s)를 가지고 있다.

이런 게임들은 약 b^d = 게임폭^{게임깊이}개의 가능한 이동 순서가 포함된 search tree에서 재귀적으로 optimal value function을 계산함으로써 풀 수 있다.

그러나 바둑은? NOPE

"바둑은 AI 가 클래식게임 play하는 것 중에 최고 난이도에 속함"

“ 바둑 경기의 경우의 수는 10의 170제곱에 이른다.

이를 숫자로 풀면
1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,000,000이다.”

https://www.bloter.net/archives/280664

바둑의 경우 b^d가 약 250^{150}
Search Tree 너무 커져서 탐색이 불가함.

2가지 원칙을 사용해서 이렇게 큰 Search Tree를 효율적으로 줄일 수 있다.

1. **Depth Reduction**
   state s 에 있을 때, search tree를 잘라내고 그 자리의 subtree를 approximate value function v(s)로 대체한다.
   전부 다 가보지 않고 value function으로 예측하는 방법

2. **Breadth Reduction**
   탐색의 폭을 policy p(a|s)로부터 action을 sampling해서 줄인다.
   찾아볼 후보군 자체를 줄이는 방법

Methods

Monte Carlo Search Tree

Search Tree에 MC 기법이 적용된 것이라고 생각하면 된다.
모든 state를 다 탐색하지 않아도 최선의 action을 구할 수 있게 하는 알고리즘.

시뮬레이션을 많이 할 수록 search tree의 규모가 커지게 되고
그만큼 value 도 더 정확해진다. value가 정확해지면 policy도 개선된다.

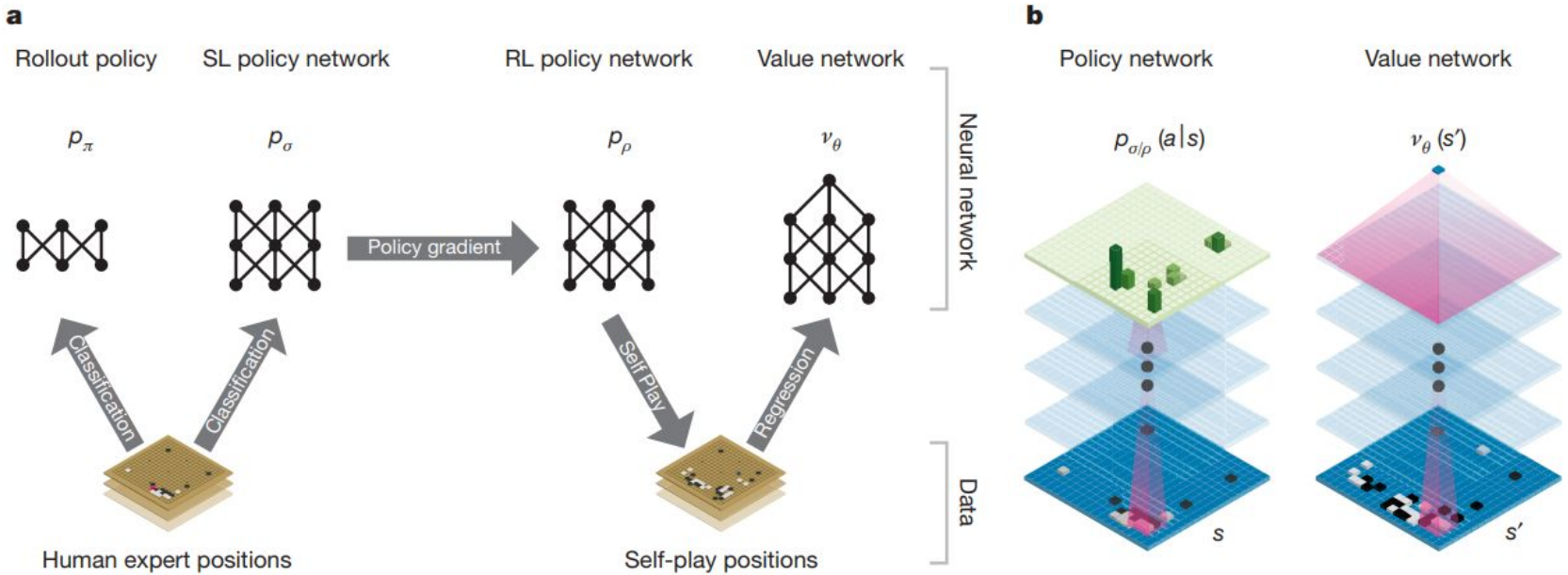
점차 policy는 optimal policy가 되고 evaluation의 경우에도 optimal value function으로 수렴하게 된다.

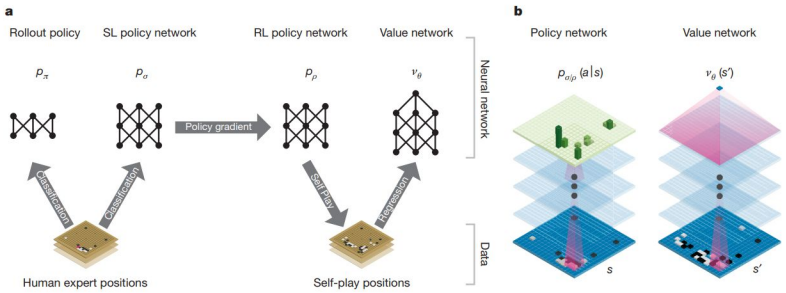
문제점 : shallow policy, value function은 input feature들의 linear combination에 기반된 구성

search tree의 breadth, depth를 효율적으로 줄이기 위해서 neural net 사용

Value network : board position evaluation

Policy network : sampling actions


a
Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Classification
Classification
Self Play
Regression
Neural network
Data
Human expert positions
Self-play positions
b
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s′



1. Rollout과 SL policy network로 프로 바둑기사들의 착수점들을 학습함.
2. RL policy network는 SL Policy로 초기화된다.
3. 새로운 데이터는 RL Policy network에서 self-play로 생성된다.
4. 이전보다 결과를 최대화하는 학습 방향을 가진 policy gradient로 improvement
5. Value network는 결과값을 예상하는 regression으로 학습.

Learning pipeline step 1

SL policy network

지도학습을 사용해서 바둑에서 전문가의 action을 예측하는 prior work 구성

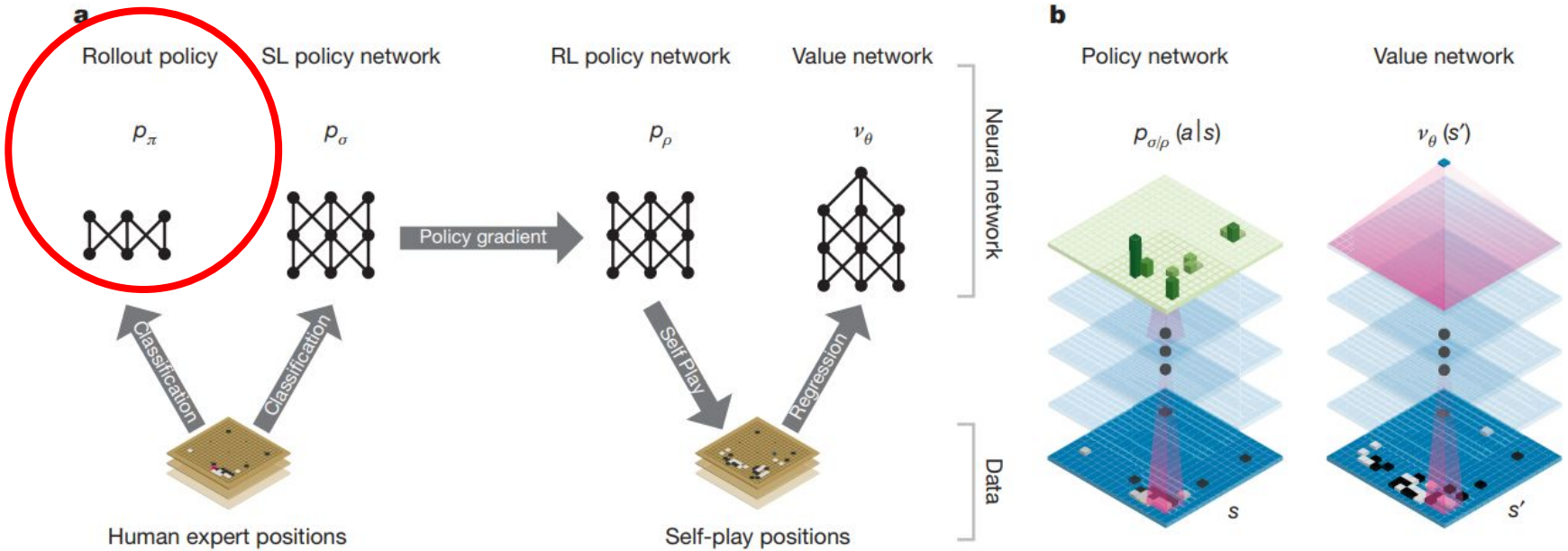
$P_\sigma(a|s)$ 는 conv layer와 non-linear layer가 반복되게 구성
마지막 softmax layer가 가능한 action a에 대한 확률 분포로 출력한다.
s는 바둑판의 상태를 간단하게 표현한 것

∴ $P_\sigma(a|s)$는 바둑기사의 기보에서 어떠한 state s와 움직일 a를 쌍으로 random sampling을 한다.

이때, 사람의 착수와 likelihood를 최대화하기 위해서 stochastic gradient ascent 사용

$$\Delta\sigma \propto (\partial \log P_\sigma(a|s) / \partial\sigma)$$

왜 있을까


a
Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Classification
Classification
Self Play
Regression
Human expert positions
Self-play positions
Neural network
Data
b
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s′

SL policy network 학습시켜서 적용해보면

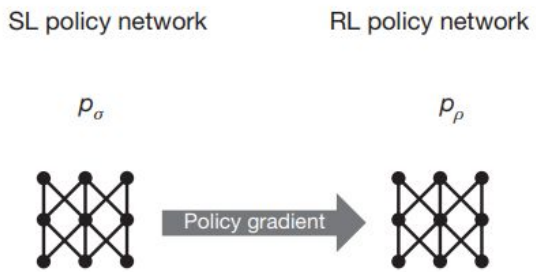
모든 input feature사용 시 57%로 바둑기사의 다음 action 예측함
position과 움직인 history, 이 두 가지만 입력으로 사용하면 55.7%로 예측함

정확도가 조금만 올라도 play 실력이 굉장히 좋아진다는 것이 중요하다.
그러나 정확도가 올라가려면 network가 커져야하고, 커지면 계산이 오래걸리게 됨
그래서 덜 정확해도 빠른 rollout policy network $P_\pi(a|s)$ 사용한다.

∴ SL policy network와 rollout policy network는 상호보완적이다. (제 생각..ㅎ)

Learning pipeline step 2

Policy gradient로 Policy network를 개선시킴



SL policy network와 RL policy network 구조는 같다.

$$\rho = \sigma$$

이전의 policy network의 iteration을 무작위로 골라서 현재 RL policy P_ρ와 서로 바둑대결을 하게 한다.

이전의 policy network pool에서의 랜덤 선택으로 현재 policy가 overfitting되지 않게 한다.

이때 reward function r(s)는 terminal time step이 아니면 전부 0이다.

$z_t = \pm r(s_T)$ 는 현재 player가 t시점에서 게임의 마지막 단계를 예측했을 때의 최종 보상의 합

win 1, lose -1

ex) z_t 를 구할 때 현재 t 시점에서 terminal state가 3가지(win 1, lose 2) 이면

z_t = (-1-1+1) = -1 z가 양수면 형세가 유리하다고, 반대로 음수면 불리하다고 판단한다.

weight update는 stochastic gradient ascent 사용하여 예측 결과가 최대화 될 수 있게 update한다.

$$\Delta\rho \propto (\partial \log P_\rho(a|s) / \partial\rho)$$

RL policy network 성능 평가에서 각 확률분포 $P_\rho(\cdot|s_t)$에서의 a_t를 샘플링해서 비교함.

RL vs SL, 80%로 RL이 승리함.

RL policy network는 탐색과정없이 강화학습만으로도 이미 존재하는 오픈소스 바둑프로그램을 이겼다.

SL의 경우에는 11~12% 승리에 그침.

∴ SL은 예측, RL은 이기기 즉, 상대방 수를 예측만 한다고 좋은 것이 아님을 알 수 있음.

Learning pipeline step 3

position에 대한 evaluation에 주목한다.

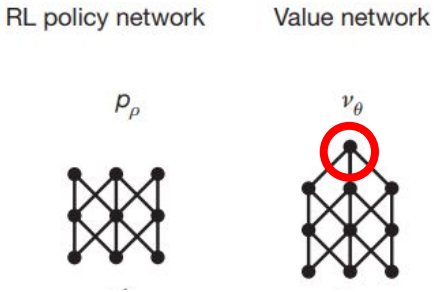
어떤 상태 s에서 두 player 모두 policy p를 따를 때, 최종 결과를 예측하는 value function $v^P(s)$를 추정하는 것이다.

$$v^P(s) = E[z_t \mid s_t=s, a_{t...T} \sim P]$$

가장 이상적인 것은 player가 둘다 완벽하게 바둑을 둬서 optimal value function v*(s) 를 아는 것인데, 알 수 없음.

그래서 P_ρ로 $V^{P\rho}(s)$를 추정한다.

$v_\theta(s) \approx v^{P\rho}(s) \approx v^*(s)$



policy network와 유사한 구조이지만 output이 하나의 예측값임 (확률분포x)

(s, z) 쌍에 대한 regression 구함

SGD, MSE 사용함

예측값 $v_\theta(s)$와 결과 z 간의 MSE를 최소화하도록 update

$$\Delta\theta \propto (\partial v_\theta(s)/\partial\theta)*(z-v_\theta(s))$$

$z-v_\theta(s)$ = 결과-승패예측값

당연히 오차가 작을수록 좋아지는것

바둑에서 좋은 돌의 위치라는 건 아주 강한 correlation을 갖는다.
그리고 하나의 돌 위치에 따라서 최종 승패가 변한다.

바둑 게임 한 판에서 s는 굉장히 많지만 z는 최종 보상 1개
s는 모두 z를 공유한다는 의미

이를 바탕으로 KGS dataset에서 학습시키면
value network v_θ는 새 position을 일반화하기보다는 승패를 기억하게 된다.

이 문제를 해결하기위해서 3천만개 게임에서 self-play data set을 만들고 각 RL network들끼리 승패가 날때까지 paly를 계속 한다.

∴ self-play로 data set 만들면 랜덤성이 더해져서 overfitting을 막을 수 있다.

self play 전)
training, MSE 0.19 test, MSE 0.37

self play 후)
testset에서 MSE 0.234로 줄어들게됨

$v_\theta(s)$를 이용해서 한번만 평가(single evaluation)하는 것의 정확도가

RL policy network Pρ를 이용한 MC rollout의 정확도에 근접함

연산은 15000배 적게 실행

Policy and Value network 이용한 Searching

AlphaGo

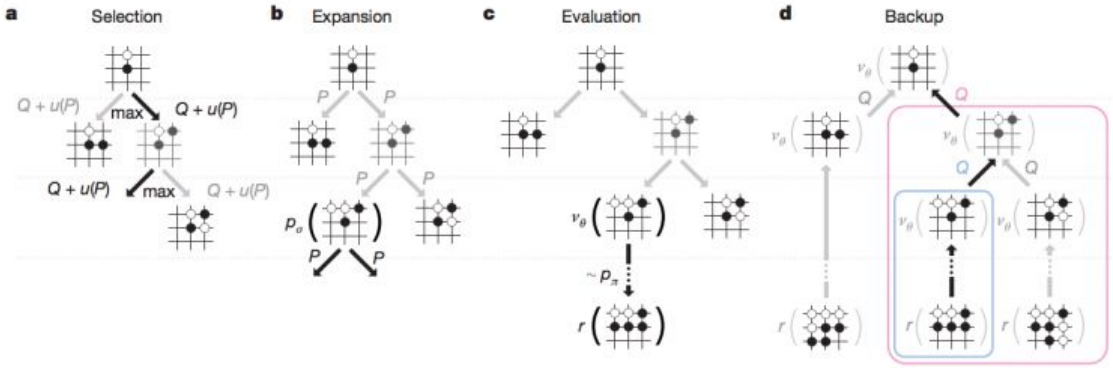
“MCTS” 로 찾고

+

“value network” 로 바둑판 평가

+

“policy network” 로 착수점 결정



a . 각 simulation은 action value Q + bonus u(p) 가 가장 큰 엣지를 선택해서 이동함.

b. 선택해서 나아가면 leaf node는 확장될 것이다. expansion으로 생성된 new node들은 SL policy network에 의해 딱 한 번 계산된다. 그리고 그 결과의 확률값이 각 action의 사전확률 p로 저장된다.

c. simulation이 끝나면 leaf node는 2가지 방법으로 evaluation 된다.
1. v_θ를 사용하는 것과 2. P_π를 사용해서 게임 끝까지 시뮬레이션 해보는 것

d. 어떤 action의 subtree에서 구한 모든 evaluation r(·)과 v_θ(·)의 평균값을 반영하기 위한 Q가 update된다

<Selection>

MCTS는 전방탐색으로 action을 selection

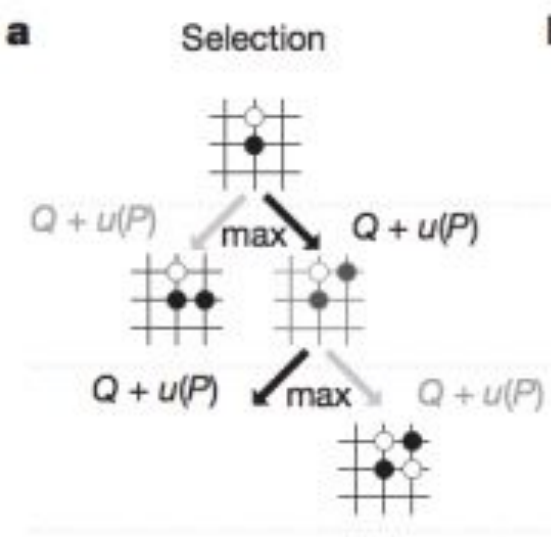
edge(s,a) : 행동 가치Q(s,a), 방문횟수 N(s,a), 사전확률 p(s,a)

tree는 root부터 backup없이 혼자 2인 self-play를 해서 완전히 내려감.

$a_t = \text{argmax}(Q(s_t,a)+U(s_t,a))$

U(s,a)는 사전확률에 비례하지만 exploration 권유를 위해 반복되는 방문횟수에 대해서는 반비례하다.

$$U(s,a) \propto p(s,a)/1+N(s,a)$$



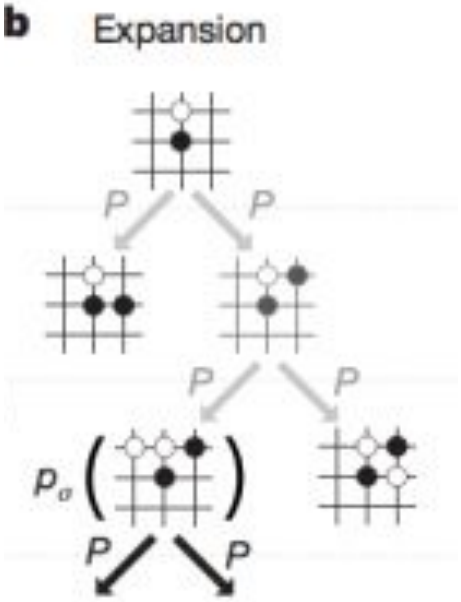
<Expansion>

탐색하면서 L step에 있는 leaf node s_L에 도달하면 leaf node는 확장한다.

expansion으로 생성된 new node들은 SL policy network에 의해 딱 한 번 계산된다. 그 결과의 확률값이 각 action의 사전확률 p로 저장된다.

-> SL policy network의 output은 가능한 actions에 대한 확률이고
new node에는 이 확률이 사전확률 p로 저장된다는 의미

$$p(s,a) = P_\sigma(a|s)$$



<Evaluation>

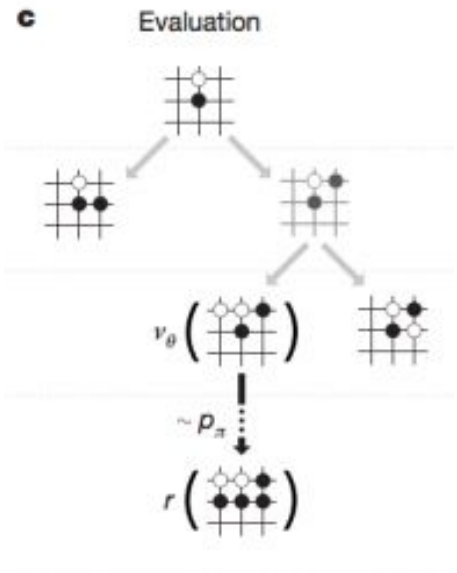
leaf node 평가 방법

1. $v_\theta(s_L)$에 의해서
2. P_π를 사용해 Terminal step T까지 랜덤 롤아웃 플레이하고 받은 reward z_L에 의해서

λ를 이용해서 두 평가를 조합하고 $v(s_L)$에 저장한다.

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

실제 결과로는 λ=0.5가 가장 좋았음



<Backup>

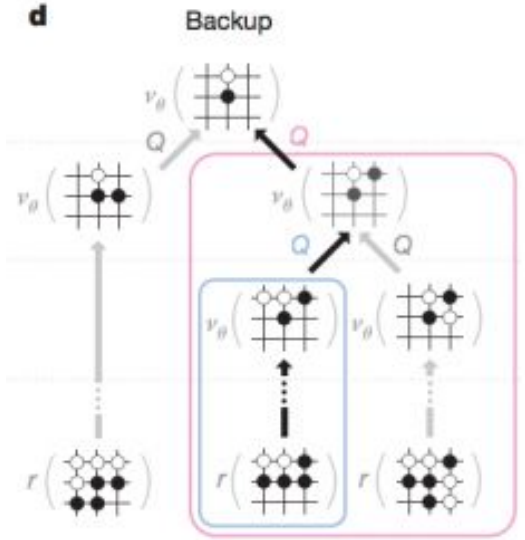
시뮬레이션의 끝에서는 action value Q와 visit count N이 update된다.

$$N(s,a) = \sum 1(s,a,i)$$

$$Q(s,a) = 1/N(s,a)\sum 1(s,a,i)v(s^i_L)$$

1(s,a,i) = edge(s,a)가 i번째 시뮬레이션에서 경유가 되는지아닌지

$v(s^i_L)$ = leaf node의 i번째 시뮬레이션



the search complete...

탐색 종료 후, root에서부터 가장 많이 visited된 node를 selection

알파고에서 P_σ가 P_ρ보다 성능이 더 좋은 이유?

사람은 자신에게 유리한 방향으로 다양하게 생각하고 행동함.

RL Policy network는 v*(s)로 최선의 선택을 하는 것에 최적화 되어 있음.

그러나 value function $v_\theta(s)$ 은 RL Policy network로 추정된 결과가 더 성능이 좋음.

MCTS와 DNN을 효율적으로 조합하기위해

CPU - simulation 수행
GPU - Policy, Value networks 수행

최종버전 알파고 - 40개 탐색스레드, 48개 CPU, 8개 GPU
분산버전 알파고 - 40개 탐색스레드, 2,202개 CPU, 176개 GPU

알파고 실력 평가는 다른버전의 알파고 및 외부 바둑프로그램들과 토너먼트 진행했다.

single 보다 분산된 경우가 훨씬 강력했음.

full size 19*19 에서 핸디캡 없이 이긴 것은 최초이다.

Things to Consider?