

1 개요

기존 MCTS의 문제점을 크게 두가지로 요약하면, 첫번째는 무작위 선택의 임의성으로 인한 노이즈/오버헤드가 크게 발생한다는 것이다. 즉,

- 양 플레이어가 최선을 다하여 결정된 경로가 아니어서 승패 결과의 신뢰성 부족
- 실험할 가치가 적은 경로를 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

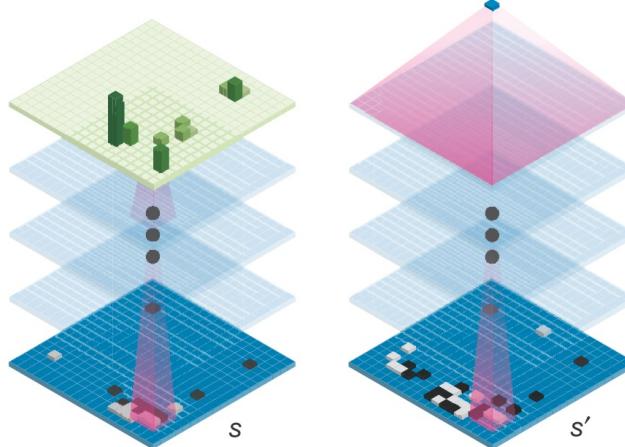
두번째 문제는, 초반 몇 번의 시뮬레이션 결과로 최종 결과가 편향되는 경향이 있는 것인데, 이는 selection/expansion 스텝에서 오차가 큰 판세분석 결과에 기반하여 탐색/확장하기 때문이다.

AlphaGo는 이 문제를 해결하기 위해 세개의 policy network와 하나의 value network를 도입하여 수많은 대국을 통해 얻은 정보를 네개의 CNN에 요약하고, 실제 경기를 위한 MCTS에서 이 CNN들을 이용하여 짧은 시간내에 훨씬 더 정교하게 게임 트리를 탐색하여 높은 성능을 보장한다.

경기전에 훈련되는 네개의 CNN은 다음과 같다.

- Policy network (정책망) p_σ, p_π, p_ρ : 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_σ : (아마추어) 인간의 기보로 학습 (supervised learning)
 - p_π : p_σ 를 단순화한 빠른 버전 (supervised)
 - p_ρ : p_σ 간의 자가대결 기보로 학습 (reinforced)
- Value network (가치망) $v_\theta(s) = "s\text{로 시작하면 이길 확률}"$

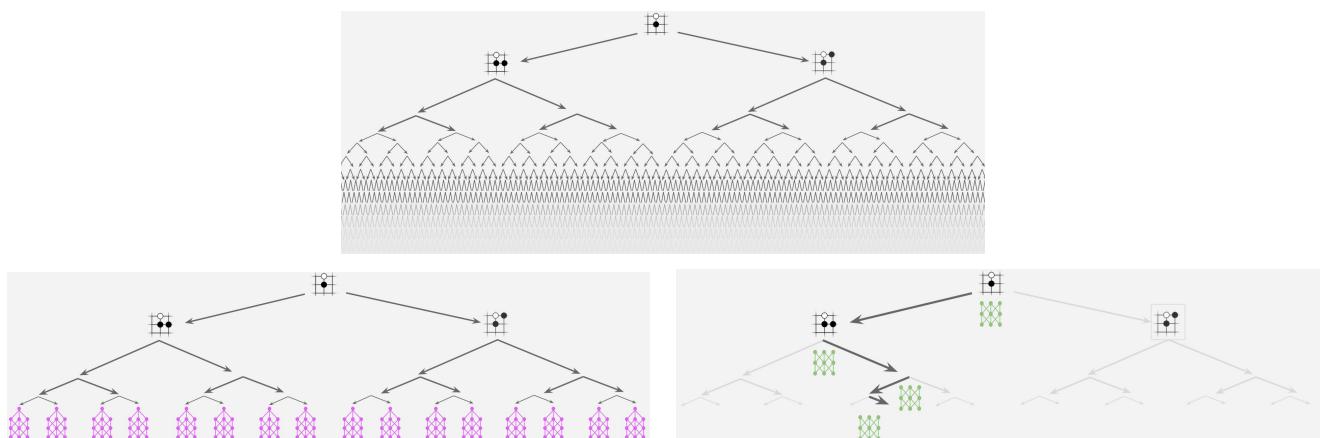
- p_ρ 간의 자가대결 결과를 입력 데이터로 학습하여 근사화
- Match 도중에는 CNN을 변경하지 않고 훈련된 그대로 사용
 - policy network
 - value network



이들 CNN을 이용하여 크게 강화된 MCTS는 다음과 같이 특성을 가진다.

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

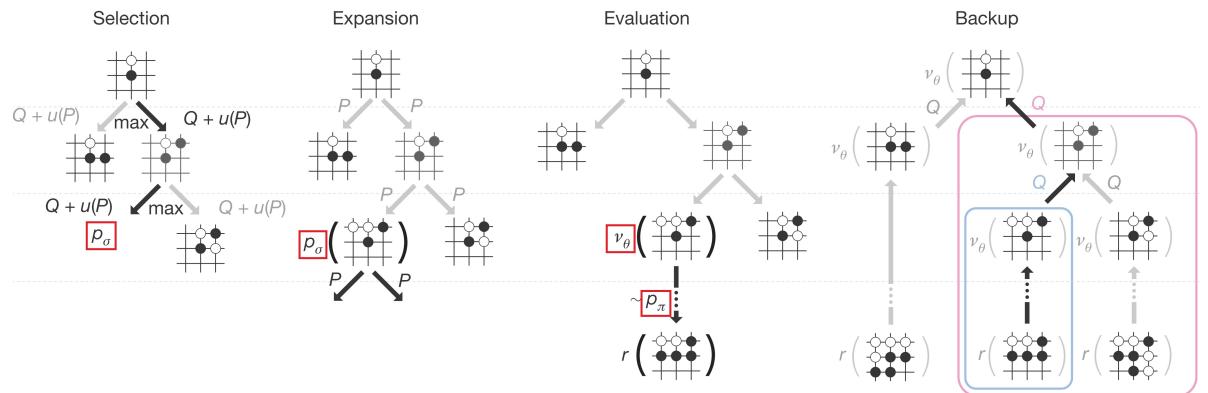
아래 그림은 full search를 할 경우의 게임 트리와 value network과 policy network을 적용한 MCTS를 이용하여 탐색 깊이와 너비를 줄인 게임 트리를 각각 보여준다.



MCTS가 비록 full search를 하지 않아도 된다지만, 결국 바둑에 적용하기 위해서는 breadth와 depth를 줄이는 과정이 필요하다. AlphaGo에서 이 둘을 줄이기 위하여 사용한 것이 바로 CNN으로, 먼저 착수하는 지점을 평가하기 위한 value network, 그리고 샘플링을 하는 distribution을 만들기 위한 policy network 두 가지 network를 사용하게 된다.

결국 AlphaGo가 한 것을 요약하자면 Monte-Carlo Tree search에 tree의 search space를 줄이기 위하여 value network와 policy network를 한 번에 learning할 수 있는 architecture를 만들고 이를 사용해 MCTS의 성능을 끌어올린 것이다. 따라서 이 방법은 바둑에서만 사용할 수 있는 방법이 아니라, MCTS를 사용할 수 있는 거의 모든 방법론에 적용하는 것이 가능하다. (그리고 실제로 AlphaZero에서 체스와 장기로도 확장시켰다.) 지금 AlphaGo가 input으로 흰 돌과 검은 돌들이 놓여져 있는 바둑판 그림을 사용하고 있기에 바둑을 학습하는 것이고, 그 이외에 search space가 너무 넓어서 exact tree search가 불가능한 model에서 전부 AlphaGo의 방법론을 사용할 수 있는 것이다.

본론으로 돌아와서, 더 자세한 설명을 하기 이전에 general MCTS에서 사용하는 네 가지 step (selection, expansion, simulation, backpropagation)을 AlphaGo는 어떻게 적용했는지 살펴보자.



1. Selection: 현재 상태에서 $Q + u$ 가 가장 큰 지점을 고른다.
 - Q : MCTS의 action-value function, 클수록 승리 확률이 높아짐
 - $u(P)$: Policy network p_σ 와 node 방문 횟수 등에 의해 결정되는 값
2. Expansion: 방문 횟수가 40회가 넘는 경우 child를 하나 expand한다.
3. Simulation: Value network v_θ 와 policy network p_π 를 이용한 fast rollout을 사용해 reward를 계산한다.
 - Value network v_θ 는 policy network p_ρ 을 사용해서 learning 한다.
4. Backpropagation: 시작 지점부터 마지막 leaf node까지 모든 edge의 parameter를 갱신한다.
5. 1-4를 (시간이 허락하는 한도 내에서) 계속 반복하다가, best child selection으로는 robust child, 즉 가장 많이 방문한 node를 선택한다.

AlphaGo는 위에서 언급한 policy network를 supervised learning (SL), reinforcement learning (RL) 두 가지로 나눠서 학습한다. SL policy network p_σ 와 p_π 는 그 동안 실제

프로기사가 둔 기보를 바탕으로 특정 기보에 대한 다음 수를 classification하는 방식으로 learning하고, RL policy network p_ρ 는 SL network p_σ 로 initialize한 후, reinforcement learning 방식 (AlphaGo의 자가대국이라고 부르는 방식)으로 주어진 기보에 대한 다음 수의 distribution을 학습한다.

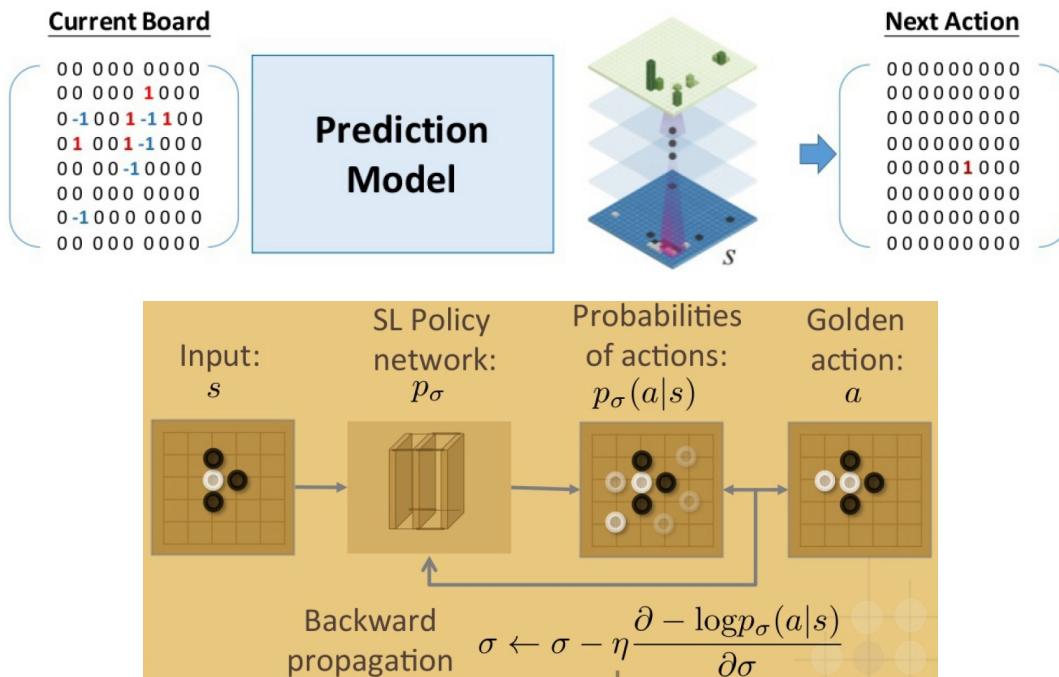
AlphaGo는 앞에서 설명한 fast rollout policy network p_π , SL policy network p_σ , RL policy network p_ρ 그리고 value network v_θ 를 한 번에 pipeline 방식으로 learning하는 architecture를 디자인했다.

2 Supervised Learning of Policy Network (p_σ, p_π)

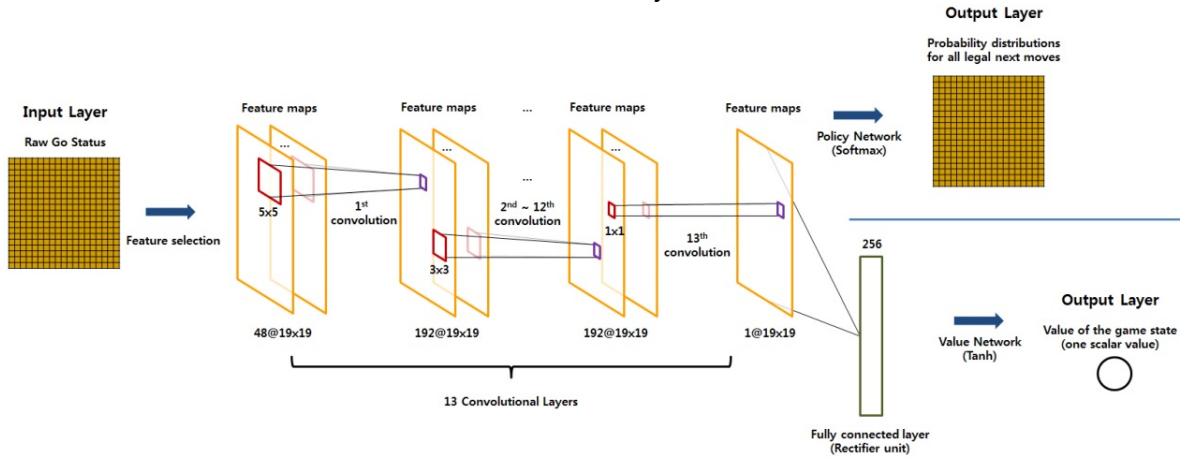
먼저 AlphaGo의 SL Policy Network $p_\sigma(a|s)$ 에 대해 살펴보자. 이 네트워크는 단순한 CNN으로, input은 시간 t 일 때의 기보(s)이고, output은 시간 $t + 1$ 일 때의 기보(a)가 된다. 따라서 이 네트워크는 classification network가 된다. 문제라면 output layer의 dimension이 너무 거대하다는 것이다. (학교에서 보유중인 머신러닝 서버를 총동원해도 이 정도의 거대한 네트워크를 learning시킬 수는 없다.) 참고로 이 네트워크는 단순 classification task만 하기 때문에 sequential할 필요는 없다. 때문에 그냥 모든 (s, a) pair에서 랜덤하게 데이터를 샘플해서 SGD로 learning하게 된다. Learning은 단순히 backward propagation

$$\sigma \leftarrow \sigma - \eta \frac{\partial - \log p_\sigma(a|s)}{\partial \sigma}$$

을 적용한다.



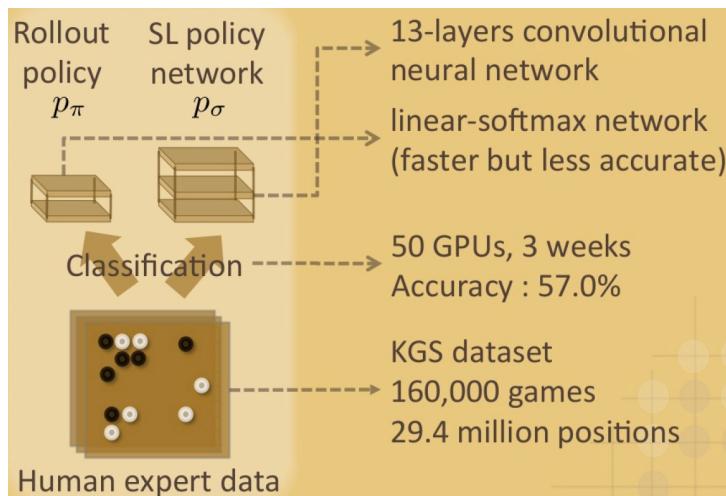
네트워크는 아래 그림과 같이 총 13 layer CNN을 사용했으며 KGS라는 곳에서 3천만 건의 기보 데이터를 가져와서 학습했다고 한다. 네트워크 구조는 inner product layer는 하나도 없이 처음부터 끝까지 convolution layer만으로 구성되어 있다.



AlphaGo는 이 부분에서 기존 state-of-art였던 44.4%보다 훨씬 좋은 classification accuracy인 57%까지 성능개선을 보였다고 한다. 또한 이 accuracy가 좋을수록 AlphaGo의 최종 winning rate가 상승한다는 사실까지 실험적으로 보이고 있다.

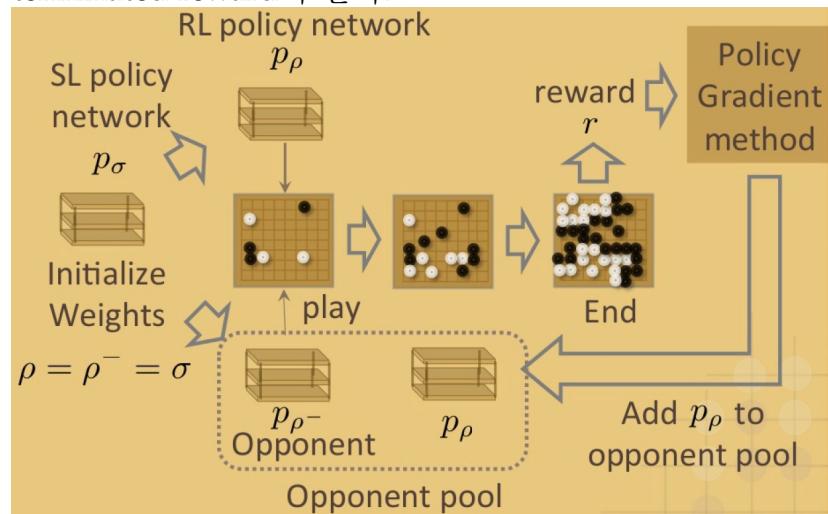
SL policy network를 training하면서 이보다 단순한 구조의 CNN인 p_π 도 함께 training 시킨다. p_σ 와 p_π 의 용도는 다음과 같다.

- p_σ : MCTS의 selection/expansion step에서 탐색할 다음 수 결정 (그리고 RL policy network의 초기화)
- p_π : MCTS의 fast rollout



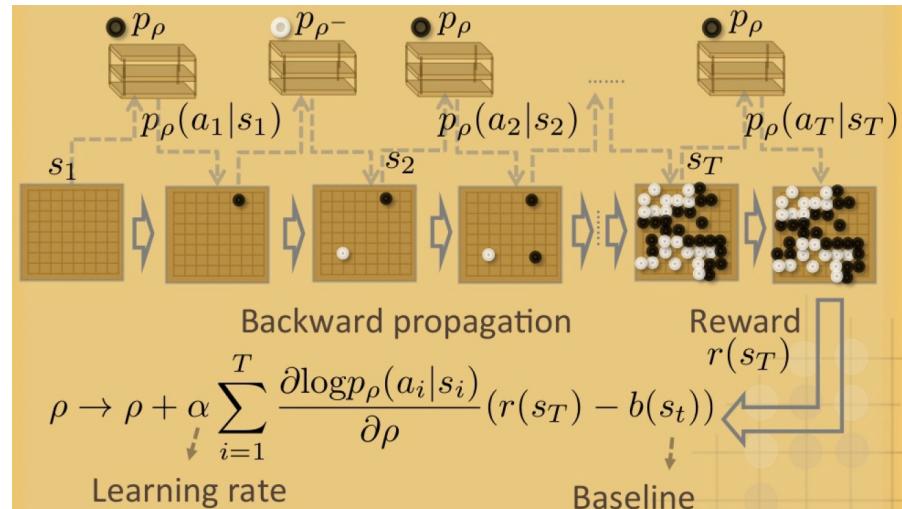
3 Reinforcement Learning of Policy Networks (p_ρ)

RL policy network p_ρ 는 SL policy network p_σ 와 동일한 구조를 가지고 있으며, 초기 값 ρ 역시 SL policy network의 parameter value σ 로 초기화된다. 현재 RL policy network p_ρ 와 이전 iteration에서 사용했던 RL policy network 중에서 랜덤하게 하나를 뽑은 다음 이 둘끼리 서로 대국을 하게 한 후, 둘 중에서 현재 네트워크가 최종적으로 이기면 reward를 +1, 지면 -1을 주도록 설정되어 있다. Reward는 대국이 끝난 시점의 T 에서의 reward이지 현재 시점 t 에서의 reward는 0이기 때문에, 대신 네트워크의 outcome을 $z_t = \pm r(s_T)$ 으로 정의한다. 즉, 이 네트워크의 outcome은 현재 player의 time t 에서의 terminated reward가 된다.



SL policy network와 마찬가지로 policy gradient 방식을 적용하여 training한다.

$$\rho \leftarrow \rho - \alpha \sum_{i=1}^T \frac{\partial -\log p_\rho(a_i|s_i)}{\partial \rho} (r(s_T) - b(s_t))$$



이 네트워크 역시 stochastic gradient method를 사용해 expected reward를 maximize하는 방식으로 학습이 된다. 여기에서 과거에 학습된 네트워크를 사용하는 이유는, 좀 더 generalize된 모델을 만들고, overfitting을 피하고 싶기 때문이라고 한다 (언론에서 말하는 ‘자기 자신이랑 계속 반복해서 대국을 진행하는 방식으로 더 똑똑해진다’라는 표현은 여기에서 나오는 자가 대국을 의미한다).

SL policy network와 RL policy network가 경쟁할 경우, 80% 이상의 게임을 RL network가 승리했다고 한다. 또한 다른 state-of-art 프로그램들과 붙었을 때도 훨씬 좋은 성능을 발휘했다고 한다.

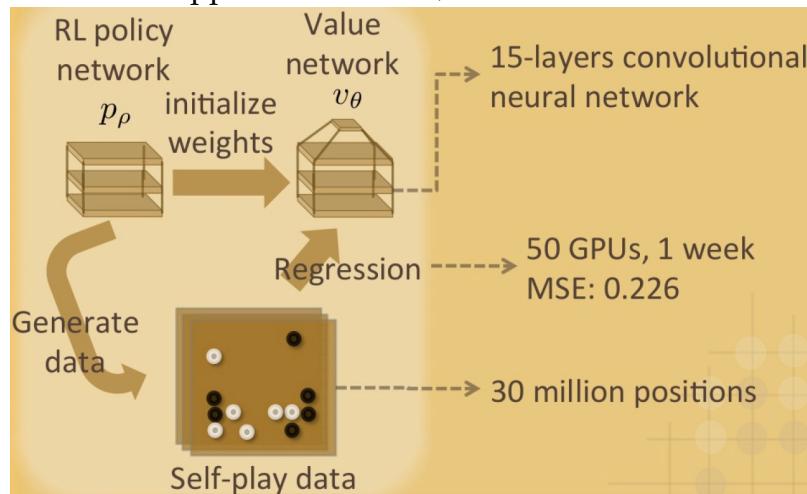
RL policy network p_ρ 는 SL policy network p_σ, p_π 와 달리 그 자체로 바로 경기를 위한 MCTS에 사용되지 않고 value network v_θ 의 training에만 사용된다. 직관에 부합하지는 않지만 SL policy network는 뒤를 생각한 좀 더 global한 결정에 적합하고 RL policy network는 그 순간의 최고의 move 선택에 적합하기 때문이라고 하며, 실제 실험을 해도 더 낫다고 한다.

4 Supervised Learning of Value Networks (v_θ)

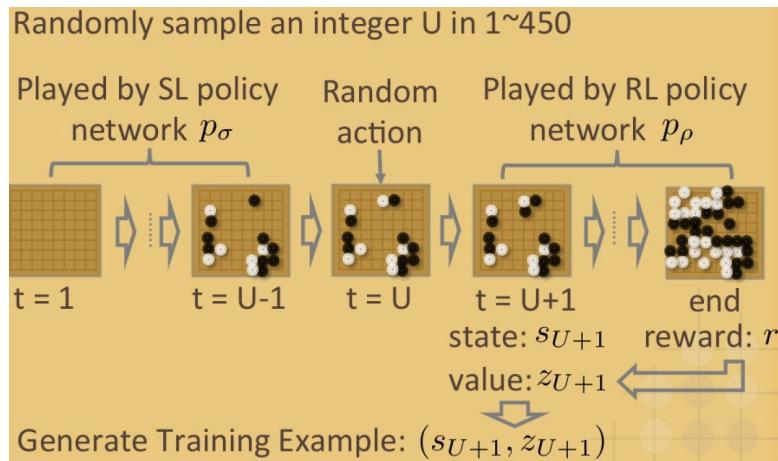
이제 AlphaGo의 training pipeline 중에서 마지막 단계인 value network $v_\theta(s)$ 를 살펴보자. Value network는 MCTS evaluation 단계에서 사용하는 네트워크로, position(현재 기보) s 와 policy p 가 주어졌을 때, value function $v^p(s)$ 를 predict하는 네트워크이다. 즉, 다음과 같은 식으로 표현할 수 있다.

$$v^p(s) = \mathbb{E}[z_t | s_t = s, a_{t \dots T} \sim p]$$

문제는 그 누구도 바둑에서 최적의 수를 계산할 수 없기 때문에(search space가 우주의 원자 개수보다 많다) optimal value function $v^*(s)$ 를 학습할 방법이 없다는 것이다. 그 대신, AlphaGo는 현재 세계에서 가장 우수한 policy인 RL policy network p_ρ 를 사용해 optimal value function을 approximation한다.

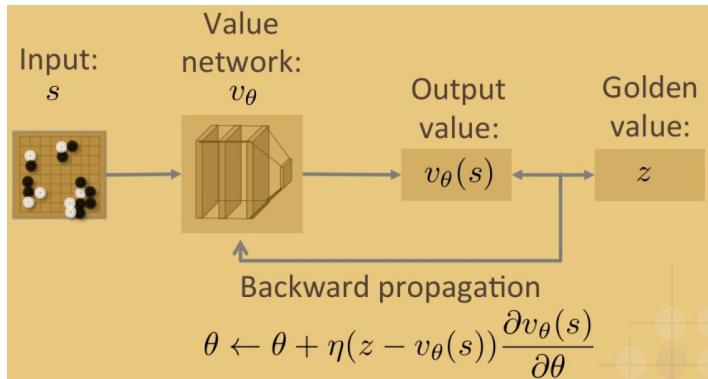


Value network는 앞에서 설명한 policy network와 비슷한 구조를 띠고 있지만, 마지막 output layer으로 모든 기보가 아닌, single probability distribution을 사용한다. 따라서 이제 문제는 classification이 아니라 regression이 된다. Value network는 현재 가장 state-outcome pair인 (s, z) 에 의해서 학습이 된다 (z 는 RL network에서 나왔던 최종 reward의 값으로 1 또는 -1이다).



따라서 value network는 s 에 대해 z 가 나오도록 하는 regression network를 학습하게 되며, error는 $z - v(\theta)$ 가 된다.

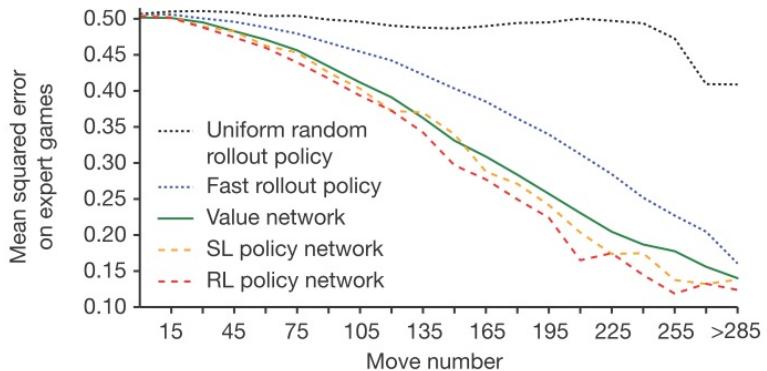
$$\theta \leftarrow \theta + \eta(z - v_\theta(s)) \frac{\partial v_\theta(s)}{\partial \theta}$$



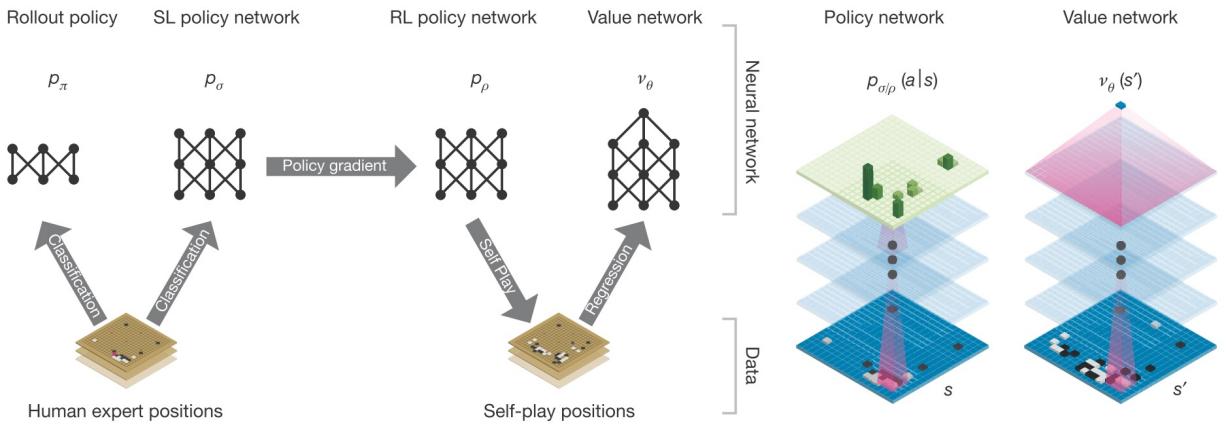
문제는, state s 는 한 개의 기보인데, reward target은 전체 game에 대해 정의되므로, successive position들끼리 서로 강하게 correlation이 생겨서 결국 overfitting이 발생한다는 것이다. 이 문제를 해결하기 위해 AlphaGo는 3천만개의 데이터를 RL policy network들끼리의 자가대국을 통해 만들어낸 다음 그 결과를 다시 또 value network를 learning하는 데에 사용한다. 그 결과 원래 training error 0.19, test error 0.37로 overfitted되었던 네트워크가, training error 0.226, test error 0.234로 훨씬 더 generalized된 네트워크로 학습되었다는 것을 알 수 있다.

아래 그림은 랜덤 policy, fast rollout policy, value network, SL network 그리고 RL

network를 사용했을 때 각각의 value network의 expected loss가 plot되어 있다. Loss는 실제 프로기사가 둔 수와, 각 policy로 둔 수와의 mean square loss이다. 결국, RL policy를 쓰는 것이 그렇지 않은 것보다 훨씬 우수한 결과를 낸다는 것을 알 수 있다.



지금까지 살펴본 네트워크들을 요약하면 다음과 같다.



왼쪽 그림은 어떻게 AlphaGo에서 세 네트워크를 training을 위한 pipeline 형태로 둘었는지를 보여준다. 사람이 실제로 둈 기보를 바탕으로 rollout policy p_π , SL policy p_σ 를 learning하고, SL policy p_σ 를 초기값으로 사용해 RL policy p_ρ 를 learning한다. 그 후 RL policy p_ρ 를 사용해 value network v_θ 를 learning하는 것이다.

지금까지 제대로 설명하지 않은 fast rollout policy p_π 는 전체 바둑 상태가 아닌 local 한 3-by-3 판에서 다음 수를 빠르게 예측해서 terminate state까지 게임을 play한 후 simulation하는 policy로, policy network p_σ 를 사용한 방법보다 성능은 떨어질지 몰라도 약 1500배 정도 빠르다고 한다.

오른쪽 그림에서는 policy network와 value network의 차이를 보여주고 있다. Policy network들은 전부 input, output이 기보로 나타나고 (input이 지금 기보, output이 다음 기보) value network는 board 전체에 대한 probability를 학습한다는 점이 다르다. 즉, policy network는 주어진 기보에서 가장 확률이 높은 action을 고르는 방식으로 MCTS의 selection을 하는 역할을 하고, value network는 simulation 결과를 통해 실제로 둘 수 있는 점들 중에서 가장 이길 확률이 높은 (reward가 승리이므로) 곳을 찾아내는 역할을

하게 되는 것이다.

5 MCTS with Policy and Value Networks

학습된 policy와 value network를 어떻게 MCTS에서 사용하는지 살펴보자. MCTS의 각각의 edge (s, a) 는 action value $Q(s, a)$, visit count $N(s, a)$, prior probability $P(s, a)$ 를 저장한다. Tree는 simulation을 사용해서 traversal을 root node에서부터 진행하게 된다. Simulation의 각 step마다, action a_t 는 state s_t 에 대해 다음과 같이 정의된다.

$$a_t = \underset{a}{\operatorname{argmax}} (Q(s_t, a) + u(s_t, a)) \text{ where } u(s, a) = c \cdot \underbrace{P(s, a)}_{p_\sigma(a|s)} \cdot \frac{\sqrt{\sum_b N_r(s, b)}}{1 + N_r(s, a)}$$

참고로 UCT 방식의 MCTS에서는

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

를 선택하였는데 이와의 가장 큰 차이는 SL policy network p_σ 를 반영하여 사람이 많이 둔 경로쪽을 더 선호하도록 했다는 점이다.

Traversal을 지속하다 leaf node L 에 도달하게 되면, expand 여부를 결정하게 된다 (방문 횟수로 결정). 그 후 leaf node에서의 position s_L 을 사용해서 SL policy network p_σ 를 prior P 에 저장한다. 즉, $P(s, a) = p_\sigma(s, a)$ 가 된다. 이때 leaf node는 두 가지 방법으로 evaluate된다. 먼저 value network $v_\theta(s_L)$, 그리고 fast rollout policy p_π 를 사용해 terminal step T 까지 도달했을 때 random rollout play로 얻어진 outcome z_L 이 둘은 parameter λ 를 사용해 다음과 같이 combine된다. ($\lambda = 0.5$ 에서 가장 좋은 성능을 보였다고 한다.)

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

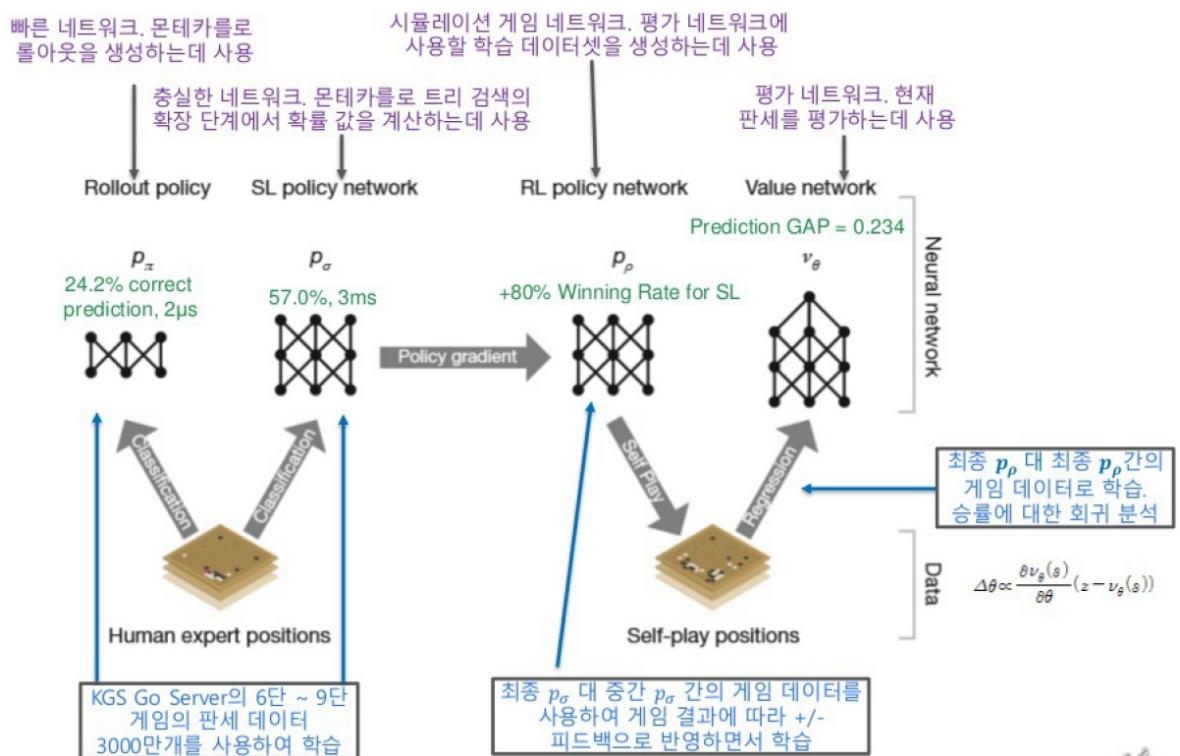
앞에서 진행한 simulation이 끝나고 나면, 이제 각 edge들이 가지고 있는 parameter들을 update할 차례다. 앞에서 언급했듯, AlphaGo의 MCTS는 각각의 edge (s, a) 에 action value $Q(s, a)$, visit count $N(s, a)$, prior probability $P(s, a)$ 를 저장한다. 여기에서 $P(s, a)$ 는 SL network p_σ 로 update가 되고, 남은건 Q 와 N 이다. 이 값들은 다음과 같은 과정으로 업데이트 된다.

$$\begin{aligned} N(s, a) &= \sum_i \mathbf{1}(s, a, i) \\ Q(s, a) &= \sum_i \frac{1}{N(s, a)} \mathbf{1}(s, a, i) V(s_L^i) \end{aligned}$$

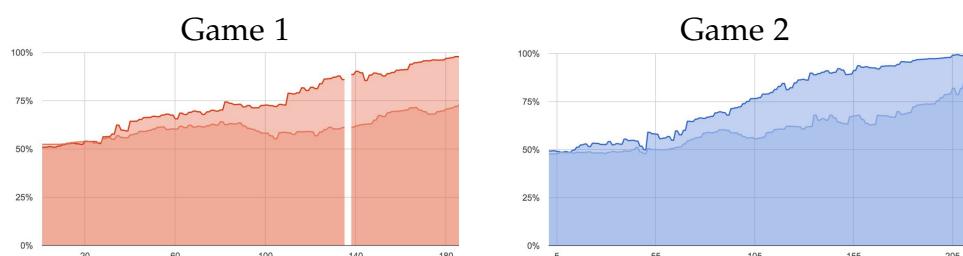
s_L^i 는 i 번째 simulation에서 leaf node를 의미하며, $\mathbf{1}(s, a, i)$ 는 i 번째 simulation에서 edge (s, a) 가 관측되었는지에 대한 indicator function이다. 이런 방식을 통해 서치가 다 끝나고나면 AlphaGo는 root에서부터 가장 많이 선택된 node를 선택하는 방식으로 한 수를 둔다.

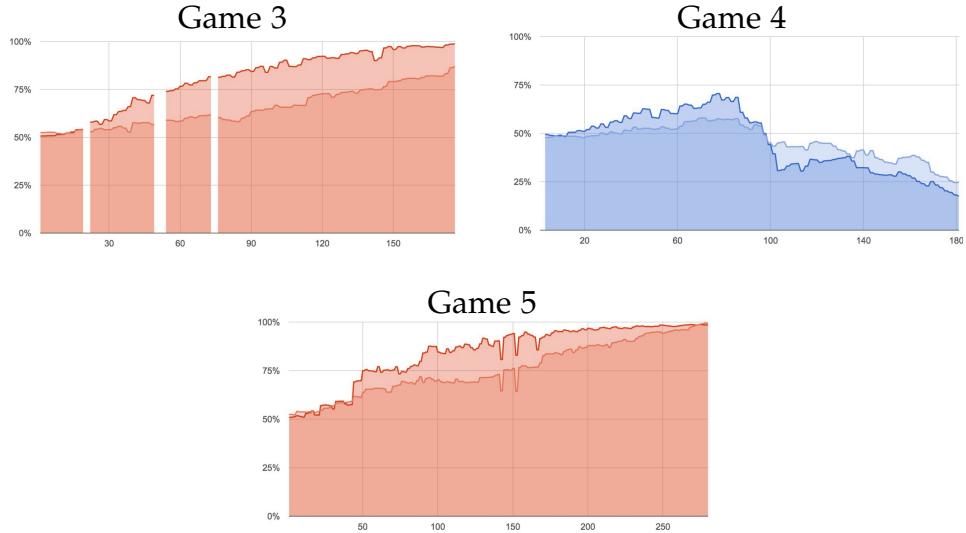
직관에 부합하지 않는 사실로, MCTS의 policy function으로 SL policy를 쓰는 것이 RL policy보다 낫다고 논문에 언급된 점이다. 이유는 (SL policy를 learning할 때 사용한) 사람이 두는 수는 뒤를 생각한 좀 더 global한 수를 두는 반면, RL policy는 그 순간의 가장 최고의 move를 하기 때문에, SL policy가 더 낫다는 것이다. 반면, value network는 SL network가 아닌 RL network를 사용하는 편이 훨씬 성능이 좋다고 한다.

MCTS에서의 용도를 포함한 네트워크에 대한 설명을 요약하면 다음 그림과 같다.



2016년도의 AlphaGo-Lee 간의 게임에서 AlphaGo의 value network가 예측한 각 게임의 시간 흐름에 따른 AlphaGo 승리 확률은 아래 그림과 같다.





6 AlphaGo Zero

2017년에 Nature에 발표된 AlphaGo Zero는 인간기보를 사용하지 않고 자가대국 만으로 훈련된 시스템으로 기존의 AlphaGo에 비해 압도적으로 우월한 성능을 보여줬다. AlphaGo Zero를 포함해서 공개된 AlphaGo 버전은 모두 4개로 다음과 같다.

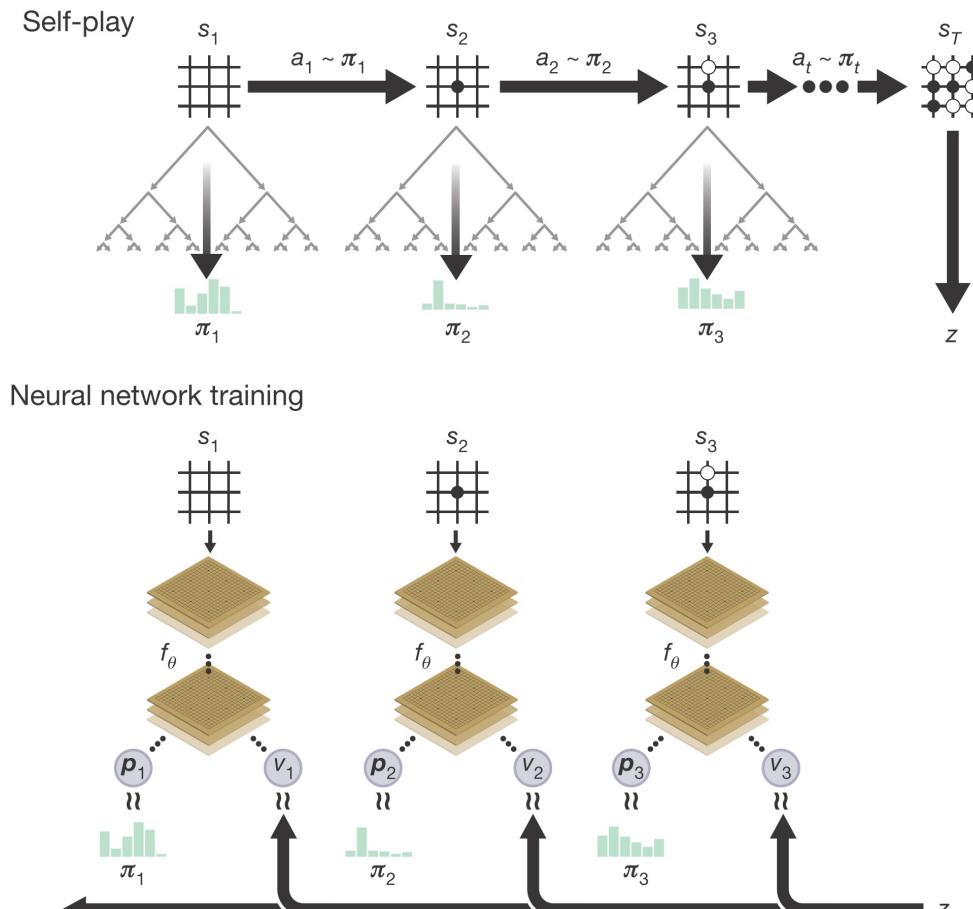
- ALPHAGo FAN (판파고, Nature 2015 논문)
 - 훈련 데이터를 뽑기 위한 경기: SL/RL policy net
- ALPHAGo LEE (돌파고, 2016)
 - 훈련 pipeline은 거의 같음 (value net 학습만 약간 다름)
 - CNN 규모가 커지고 ($192 \Rightarrow 256$) GPU 대신 TPU 도입
- ALPHAGo MASTER (마파고, 2017, 중국 기원과 대련)
 - ALPHAGo ZERO와 거의 같음. 인간 기보로 초기화 하는 것과 48개 feature를 뽑아내는 것은 돌파고와 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 Zero의 큰 특징은 다음과 같다.

- 인간 기보로 CNN 초기화 하지 않음 (*tabula rasa*)
- Policy network p 와 value network v 을 하나의 CNN f_θ 로
 - $p(s, a) = "s\text{에 } a\text{에 두는 것이 최적일 확률"$

- $v(s) = "s\text{로 시작하면 이길 확률}"$
- $(p, v) = f_\theta$ (θ 는 CNN의 weights)
- residual block 형태의 CNN으로 구현하여 학습 성능 향상
- CNN 층면: residual block, batch normalization, rectifier nonlinearities
- 훈련 데이터 생성용 경기에도 MCTS 사용
 - 훈련용 데이터: MCTS로 evaluation한 π + winner 정보 z
 - $(p, v) = f_\theta$ 가 (π, z) 에 최대한 가까워 지도록 θ 를 학습:
$$\text{loss} = (z - v)^2 + \pi^T \log p + c\|\theta\|^2$$
- 48개의 feature 대신 흑/백/empty 3개 feature만 사용 (no domain knowledge)
- MCTS에서 rollout을 사용 안하고 value net만 사용

AlphaGo Zero의 training pipeline은 아래와 같이 policy network와 value network가 통합된 형태의 네트워크를 policy iteration 형태의 강화학습을 통해 훈련시키며, 이전 버전들과 달리 훈련 데이터 생성을 위한 경기에도 MCTS (with CNN)를 사용하여 더욱 의미 있는 훈련 데이터를 생성하고 학습시켰다. 이를 위해 필요한 시간은 수천배 이상 많아졌지만, 그에 따른 매우 큰 폭으로 성능이 향상되었다.



아래 그림은 이전 버전과의 성능 비교를 보여주는데, 훈련 4일차 정도에 돌파고, 33일차 정도에 마파고 버전을 추월하였으며 ELO의 차이도 매우 커서 이전 버전들에게 100%에 가까운 승률을 보였다고 한다. 또한 같은 방식을 체스와 장기에도 적용하여 마찬가지로 매우 높은 성능을 보였다고 한다.

