Mastering the game of Go with deep neural networks

David Silver, Aja Huang, Christopher Maddison, 2016, Nature

Yakshimamedov Vladimir, 192

What is Go?

Two-player, zero-sum, complete information.

States(positions) ≈

2.081681994 * 10^170

Chess positions upper bound -

10^123

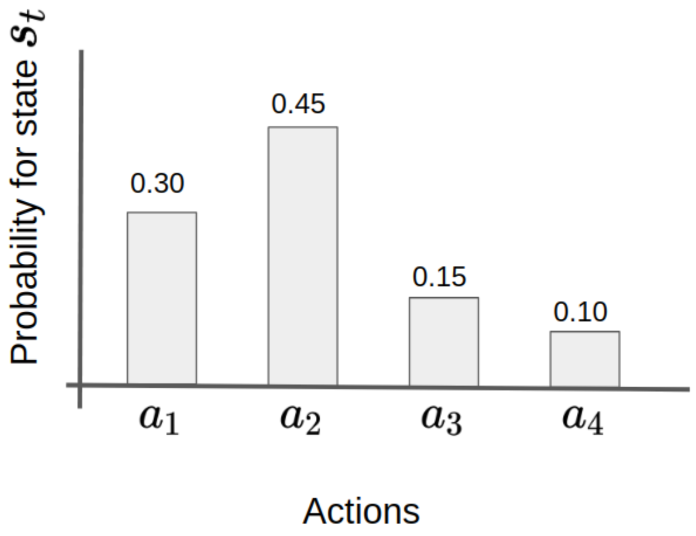
Policy and value functions

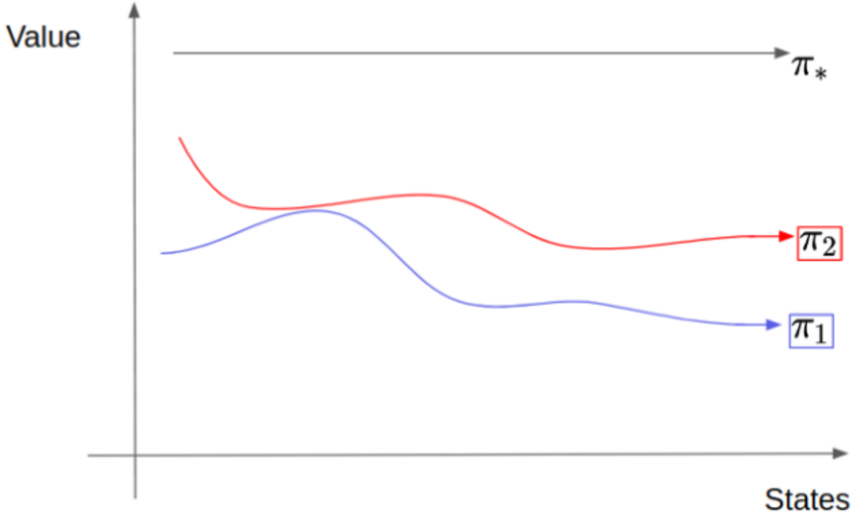
Policy - f: state -> probabilities to choose each action.

Softmax can be added to choose one action instead.

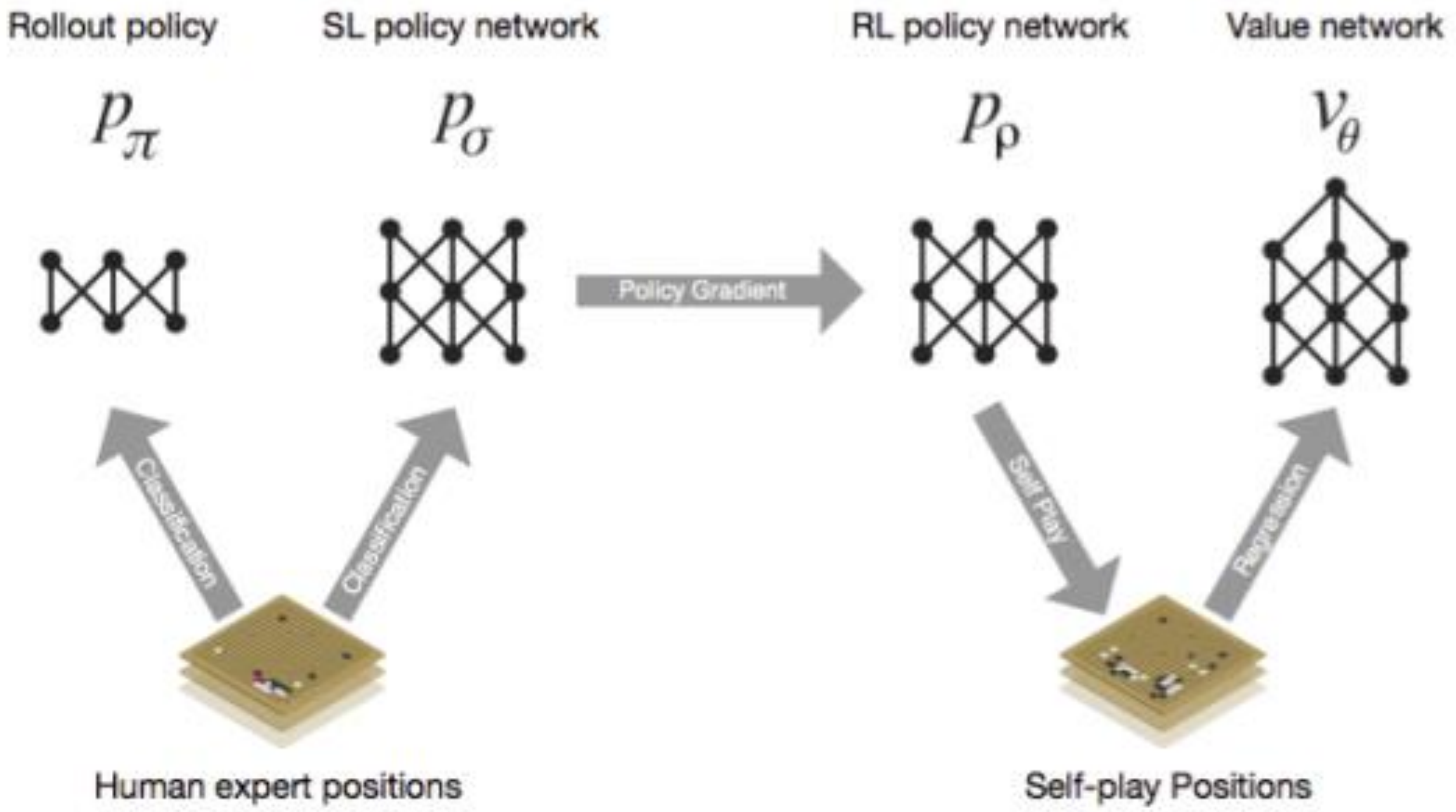
Value function - $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$

Action-value function - $q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$

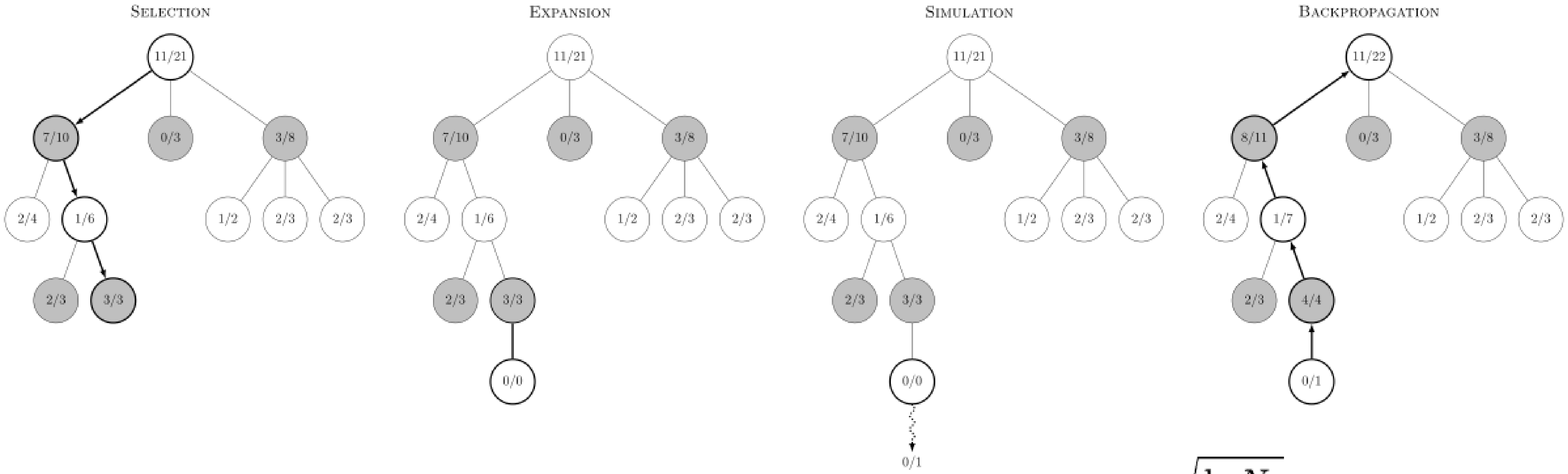




Full AlphaGo structure



Monte-Carlo Tree Search (MCTS)



Selection – Choose action: $a_t = \underset{a}{\operatorname{argmax}} \left(Q(s_t, a) + u(s_t, a)\right)$

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

Expansion – If not terminal, add children.

Simulation – Choose a child and play a complete (random or simple policy) playout.

Backpropagation – Update win count and play count on parents.

Chad Rollout Policy

Chooses an action before you ask. (2μs to choose an action)

Trained on like 6 features.

Only 32 filters of conv layers

What are symmetries?

Accuracy on test set: 24.2%

Virgin Supervised Learning Policy

Needs 3 ms to choose

Spends its life looking at 50 features.

256 filters of convolutional layers

Spins the board for no reason.

(Mean of 8 evaluations – flipping the colour and rotating the board)

Took 3 weeks to train.

Accuracy on test set: 57%

RL policy network

Initialization:

Structure and weights are the same as SL policy.

Step:

Play a batch of games with one of the previous iterations

Update weights using stochastic gradient.

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\rho} z_t \,.$$

(log likelyhood)

RL value network

Initialization:

Structure is the same as SL policy, but with 1 output.

Step:

Generate a lot of games.

Choose one position from every game and predict the outcome of that game.

Subsequent positions from fewer games are significantly worse.

Optimizing MSE

Monte-Carlo in AlphaZero

Selection:

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

P(s, a) - prior probability to choose this action from SL policy network

Exploration function - $u(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)}$

Expansion:

1. Reach the threshold for playout counts.
2. Creating node with all action edges:

For every edge:

$N_r(s,a)$ - count of playout simulations for every edge

$N_v(s,a)$ - count of value network evaluations for the s' = s + a.

$W_r(s,a)$ - wins over $N_r(s,a)$ playouts.

$W_v(s,a)$ - value sum over $N_v(s,a)$ evaluations.

Q(s, a) - state-action evaluation. Initialized as 0.

Monte-Carlo
in AlphaZero (Part 2)

Simulation (Evaluation):
Evaluate new state with value network if it wasn't already.

Playout this position using rollout policy.

Backup:
For each edge:
$N_r(s,a)$ += 1
$N_v(s,a)$ += 1
$W_r(s,a)$ += 0 or 1
$W_v(s,a)$ += value network evaluation of the new state
Q(s, a) = $(1-\lambda)\frac{W_v(s,a)}{N_v(s,a)} + \lambda\frac{W_r(s,a)}{N_r(s,a)}$

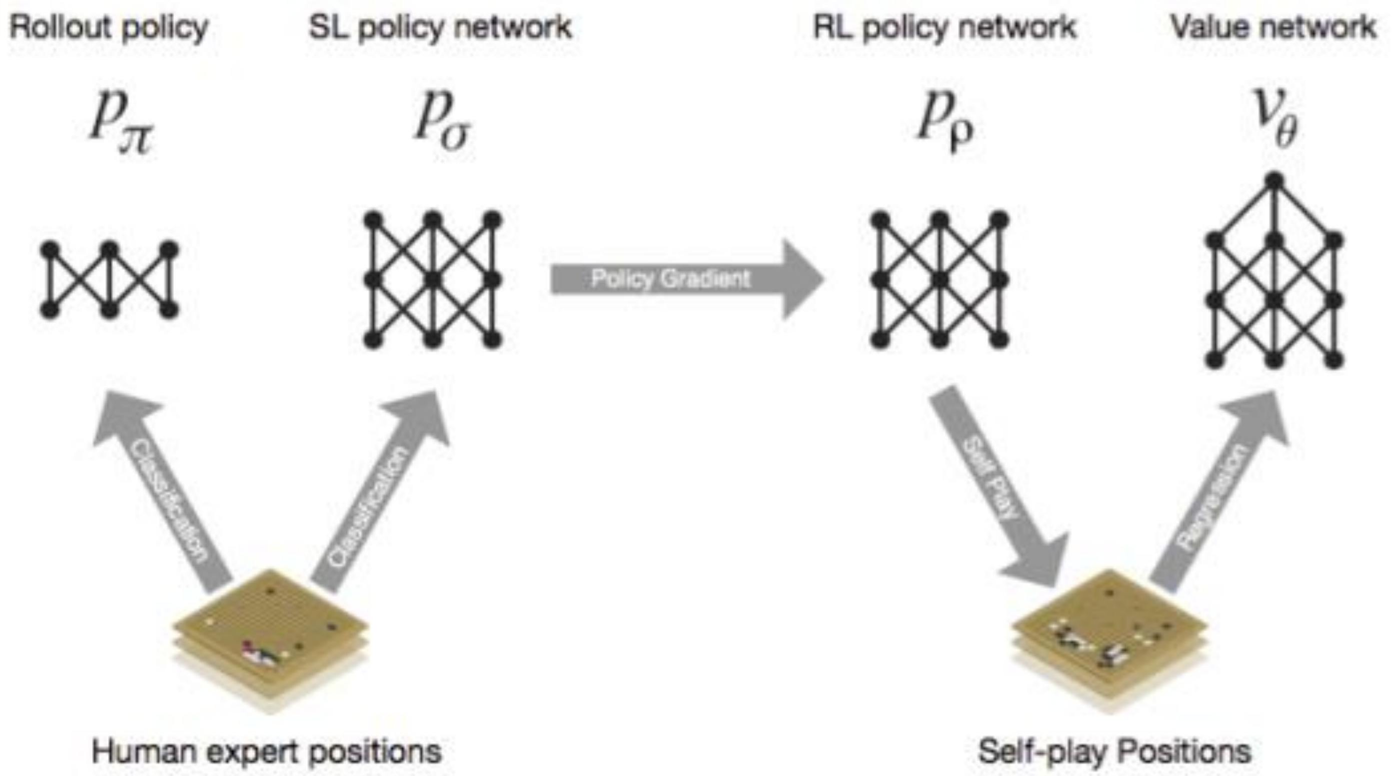
Full AlphaGo structure

Rollout – fast policy for playouts in MCTS.

SL policy – pre-train for RL policy + prior probabilities for MCTS.

RL policy – better SL for generating games

Value network – finally something evaluating positions.



Parameters

Lambda = 0.5 is the optimal parameter.

But even relying only on value network or only MCTS gives decent results.

Hell, even MCTS on rollouts is probably better than me.

Short name	Policy network	Value network	Rollouts	Mixing constant	Policy GPUs	Value GPUs	Elo rating
α_{rvp}	p_σ	v_θ	p_π	$\lambda = 0.5$	2	6	2890
α_{vp}	p_σ	v_θ	–	$\lambda = 0$	2	6	2177
α_{rp}	p_σ	–	p_π	$\lambda = 1$	8	0	2416
α_{rv}	$[p_\tau]$	v_θ	p_π	$\lambda = 0.5$	0	8	2077
α_{v}	$[p_\tau]$	v_θ	–	$\lambda = 0$	0	8	1655
α_{r}	$[p_\tau]$	–	p_π	$\lambda = 1$	0	0	1457
α_{p}	p_σ	–	–	–	0	0	1517

$[p_\tau]$ - Monte-Carlo Tree Search on only Rollout policy

p_σ - Supervised Learning policy

Distribution (or how to throw 176 GPUS at a model)

Monte Carlo Changes – add constant to rollout count at the start of an iteration to discourage other threads to follow the same path.

CPUS – rollouts.

GPU – value and policy networks.

Short name	Computer Player	Version	Time settings	CPUs	GPUs	KGS Rank	Elo
α_{rvp}^{d}	Distributed *AlphaGo*	See Methods	5 seconds	1202	176	–	3140
α_{rvp}	*AlphaGo*	See Methods	5 seconds	48	8	–	2890
CS	CrazyStone	2015	5 seconds	32	–	6d	1929
ZN	Zen	5	5 seconds	8	–	6d	1888
PC	Pachi	10.99	400,000 sims	16	–	2d	1298
FG	Fuego	svn1989	100,000 sims	16	–	–	1148
GG	GnuGo	3.8	level 10	1	–	5k	431
CS_4	CrazyStone	4 handicap stones	5 seconds	32	–	–	2526
ZN_4	Zen	4 handicap stones	5 seconds	8	–	–	2413
PC_4	Pachi	4 handicap stones	400,000 sims	16	–	–	1756