AlphaGo Zero

Kharlamov Aleksey

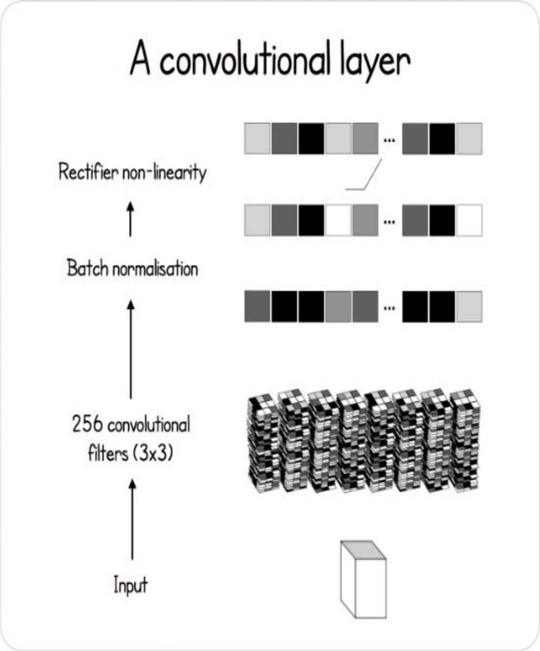
Intro

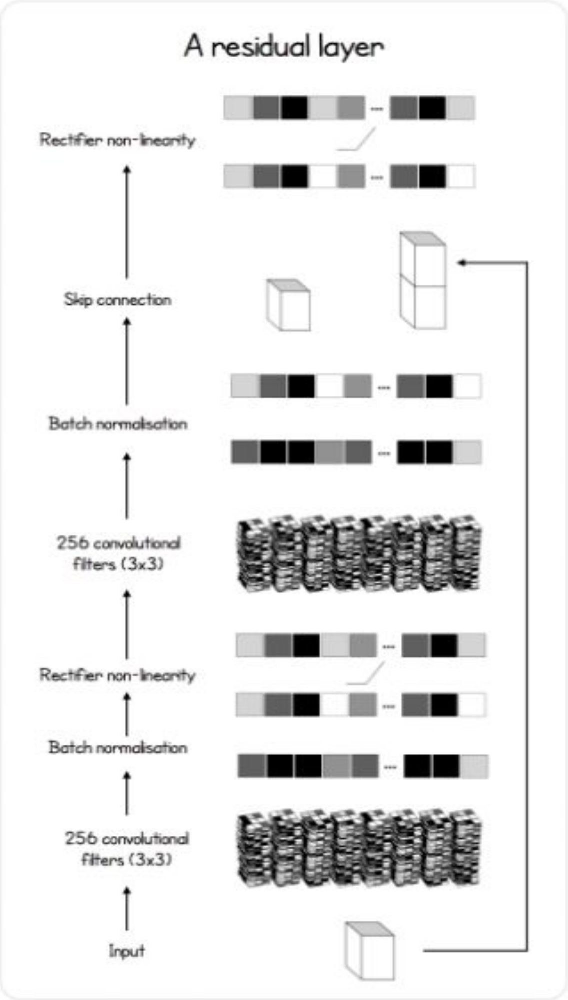
- Neural network description
- Main algorithm
- Self-play
- Performance

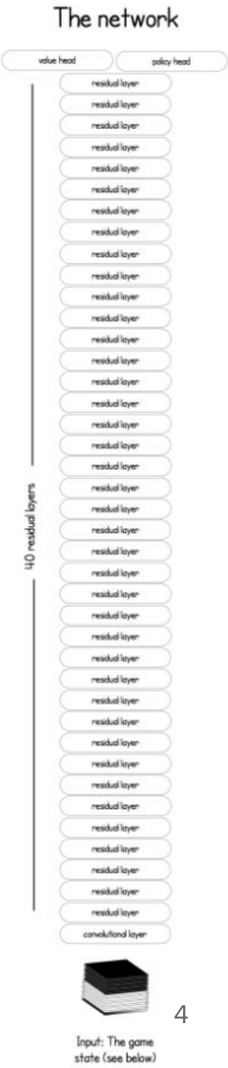
Neural network

Deep neural network f with parameters θ. This neural network takes as an input the raw board representation s of the position and its history, and outputs both move probabilities and a value, (p, v)=f. The vector of move probabilities p represents the probability of selecting each move a (including pass), p(a)=Pr(a|s).

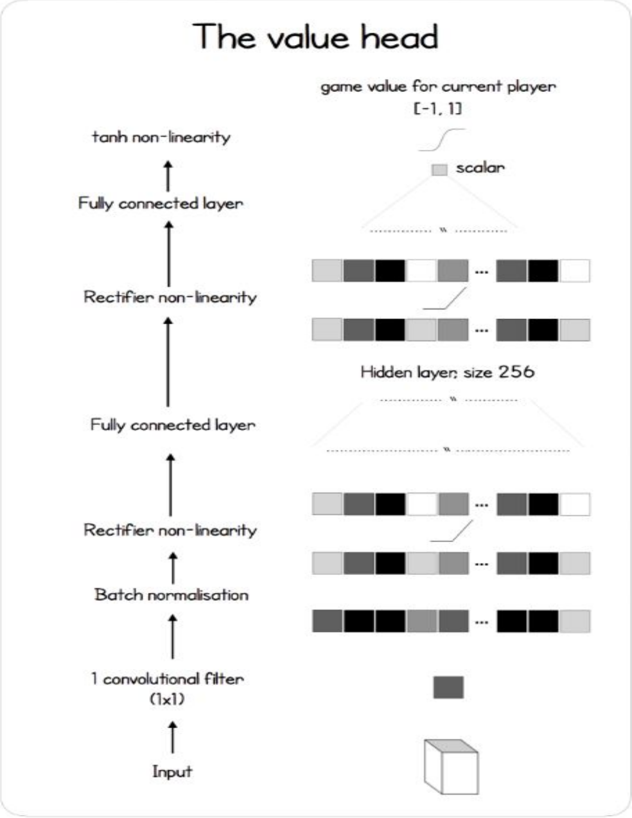
Architecture Part One

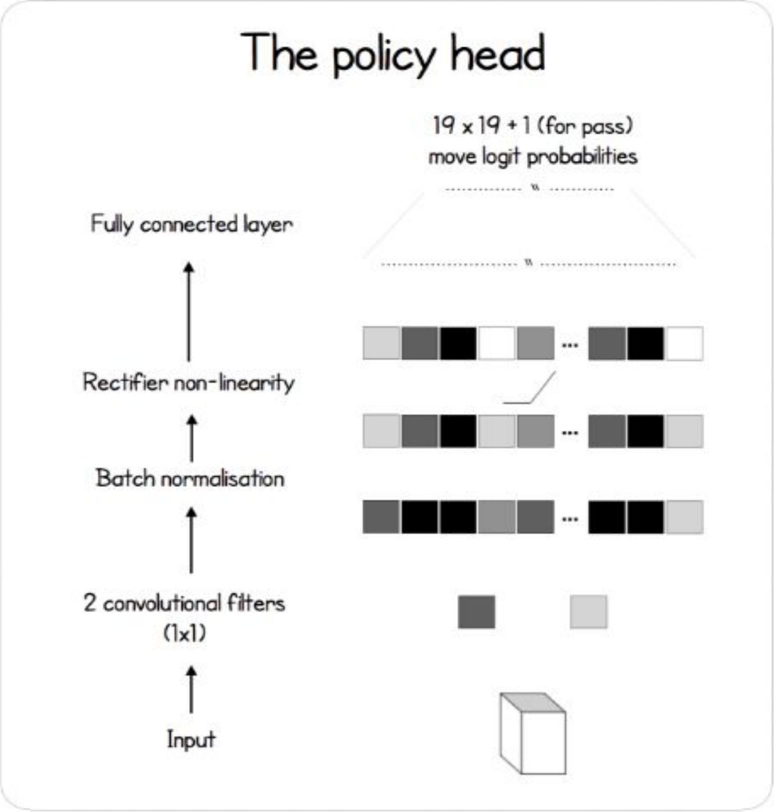


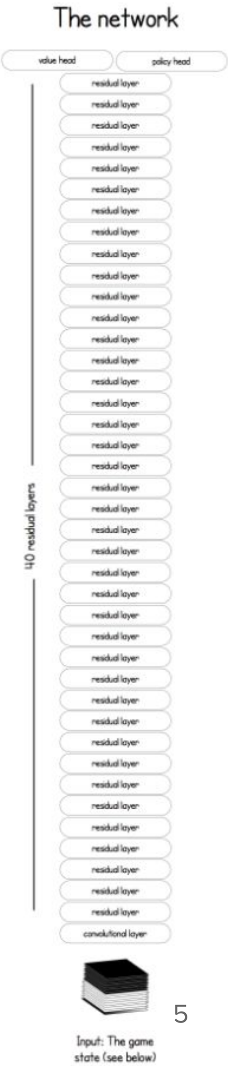




Architecture Part Two



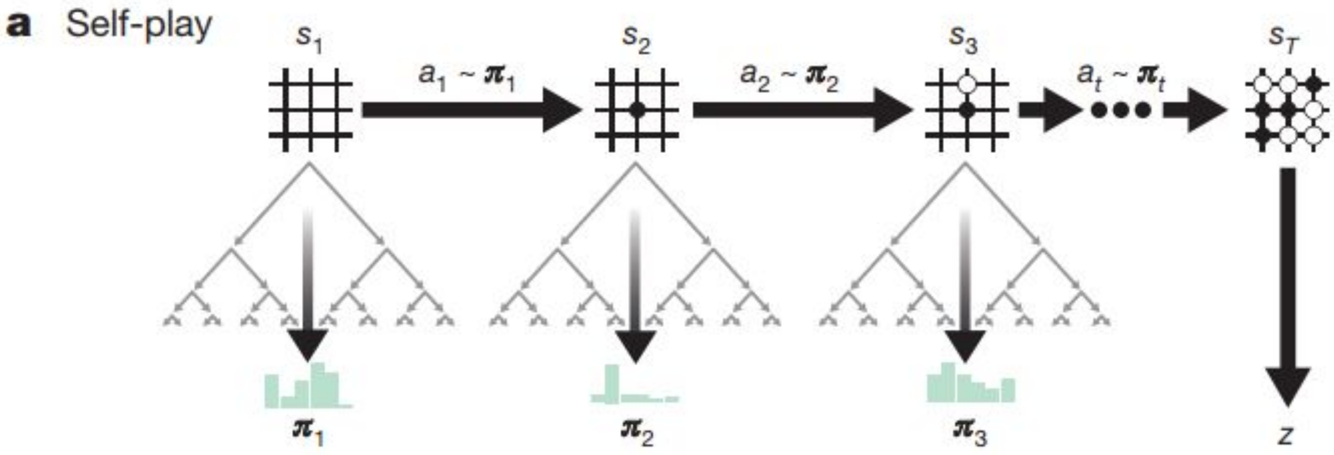




Main algorithm

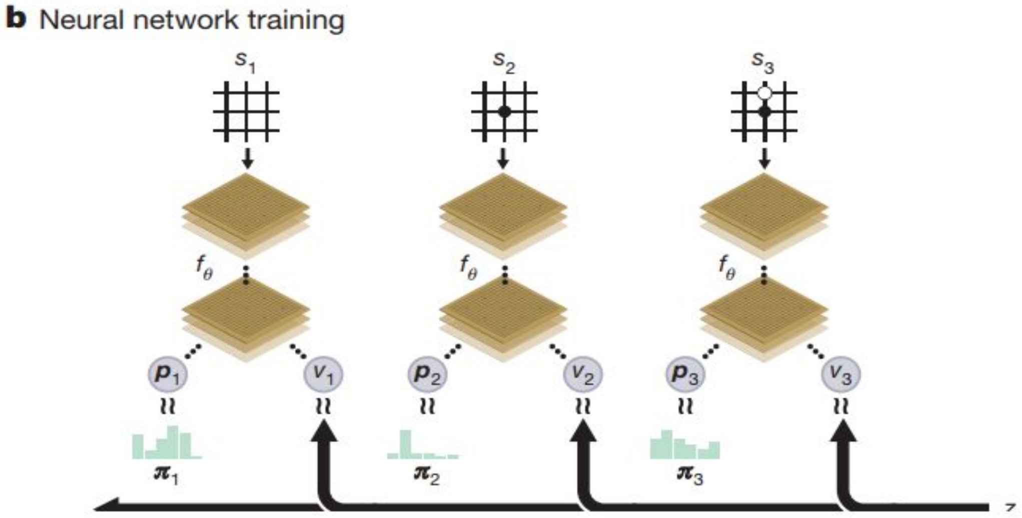
The general scheme of training:

1) Self-playing several times.
2) Update weights and go to step 1.



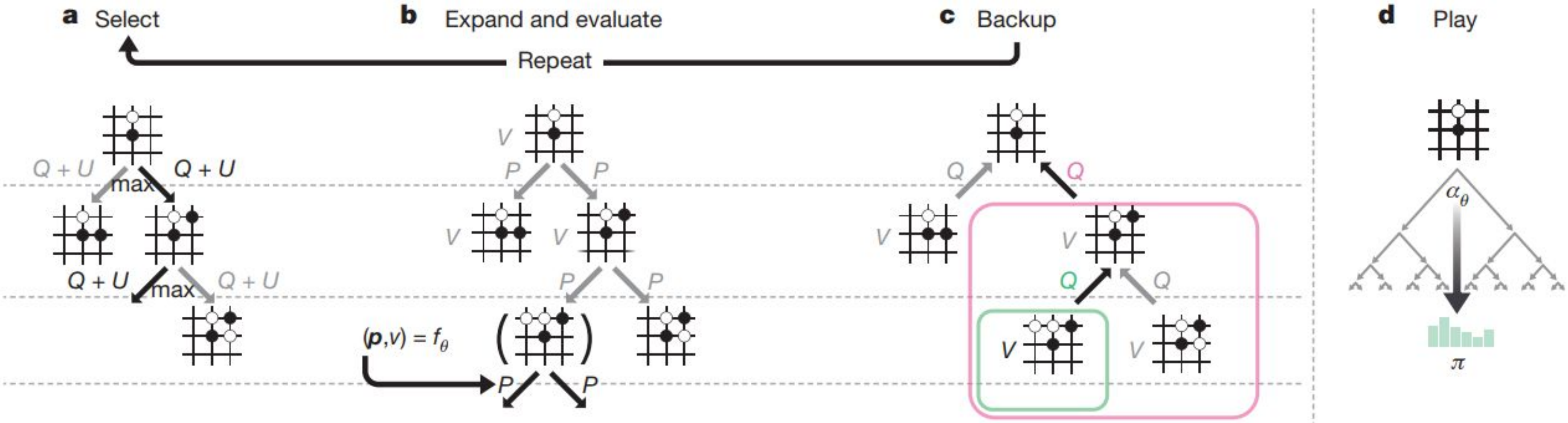
Weights update

The neural network parameters θ are updated to maximize the similarity of the policy vector p to the search probabilities π, and to minimize the error between the predicted winner v and the game winner z.



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

Self-play: MCTS



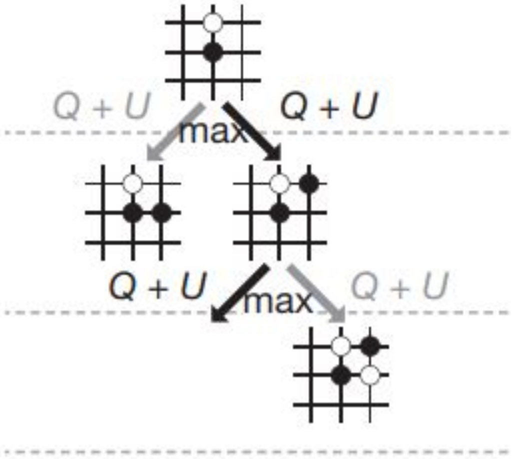
$$\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$$

where N(s, a) is the visit count, W(s, a) is the total action value, Q(s, a) is the mean action value and P(s, a) is the prior probability of selecting that edge.

MCTS: Select

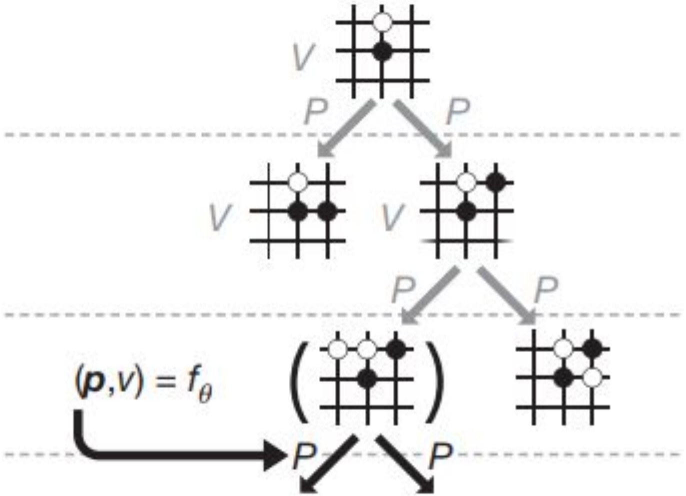
Each simulation starts from the root state and iteratively selects moves that maximize an upper confidence bound Q(s, a)+U(s, a), where U(s, a)∝P(s, a)/ (1+N(s, a))

$$U(s,a) = c_{\mathrm{puct}} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$



MCTS: Expand and evaluate

The leaf node is expanded and each edge (s, a) is initialized to {N(s, a) = 0, W(s, a) = 0, Q(s, a) = 0, P(s, a) = p(a)}; the value v is then backed up.



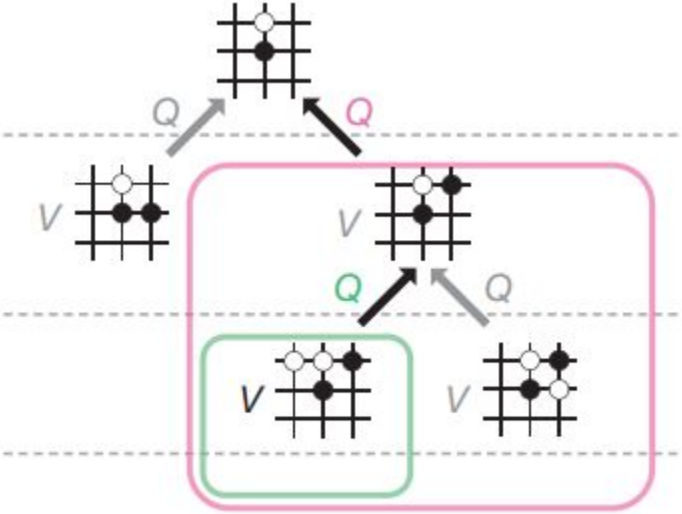
MCTS: Backup

The edge statistics are updated in a backward pass through each step t≤L. The visit counts are incremented, and the action value is updated to the mean value.

$$N(s_t, a_t) = N(s_t, a_t) + 1$$

$$W(s_t, a_t) = W(s_t, a_t) + v$$

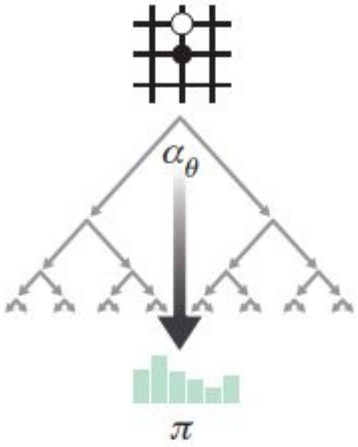
$$Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$$



MCTS: Play

At the end of the search AlphaGo Zero selects a move a to play in the root position s0, proportional to its exponentiated visit count. Where τ is a temperature parameter that controls the level of exploration.

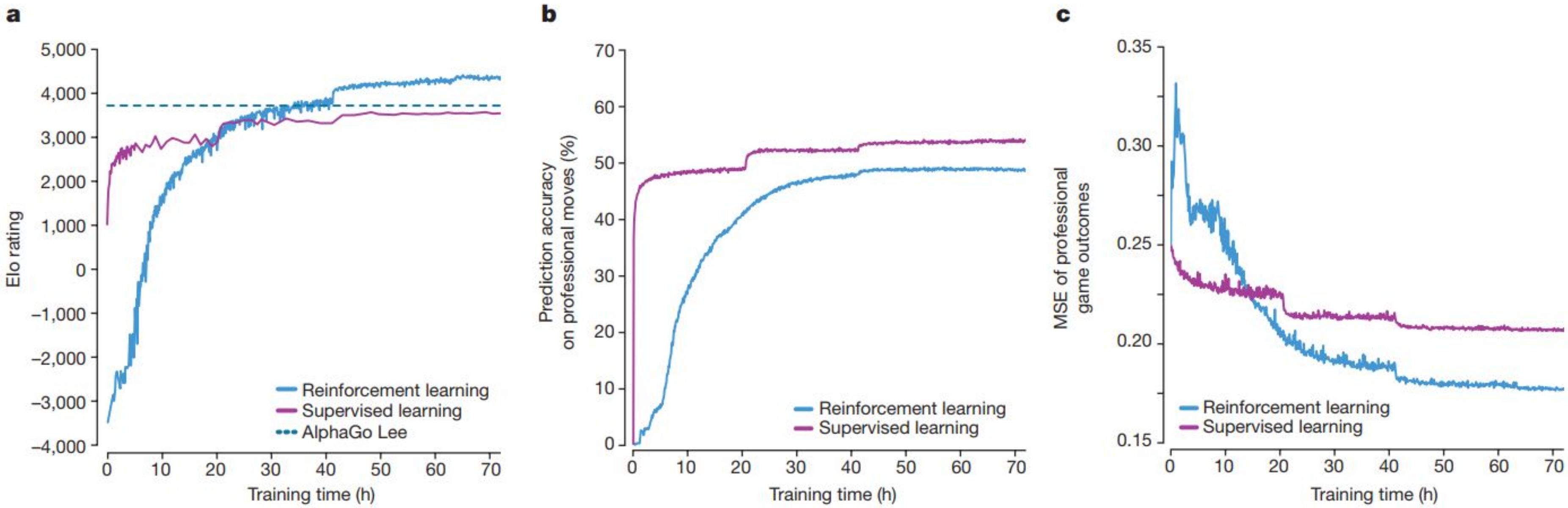
$$\pi(a|s_0) = N(s_0, a)^{1/\tau} / \sum_b N(s_0, b)^{1/\tau}$$



Summary

- Algorithm that learns, tabula rasa, superhuman proficiency.
- It uses only the black and white stones from the board as input features.
- Single neural network, rather than separate policy and value networks.
- Simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts.

Performance



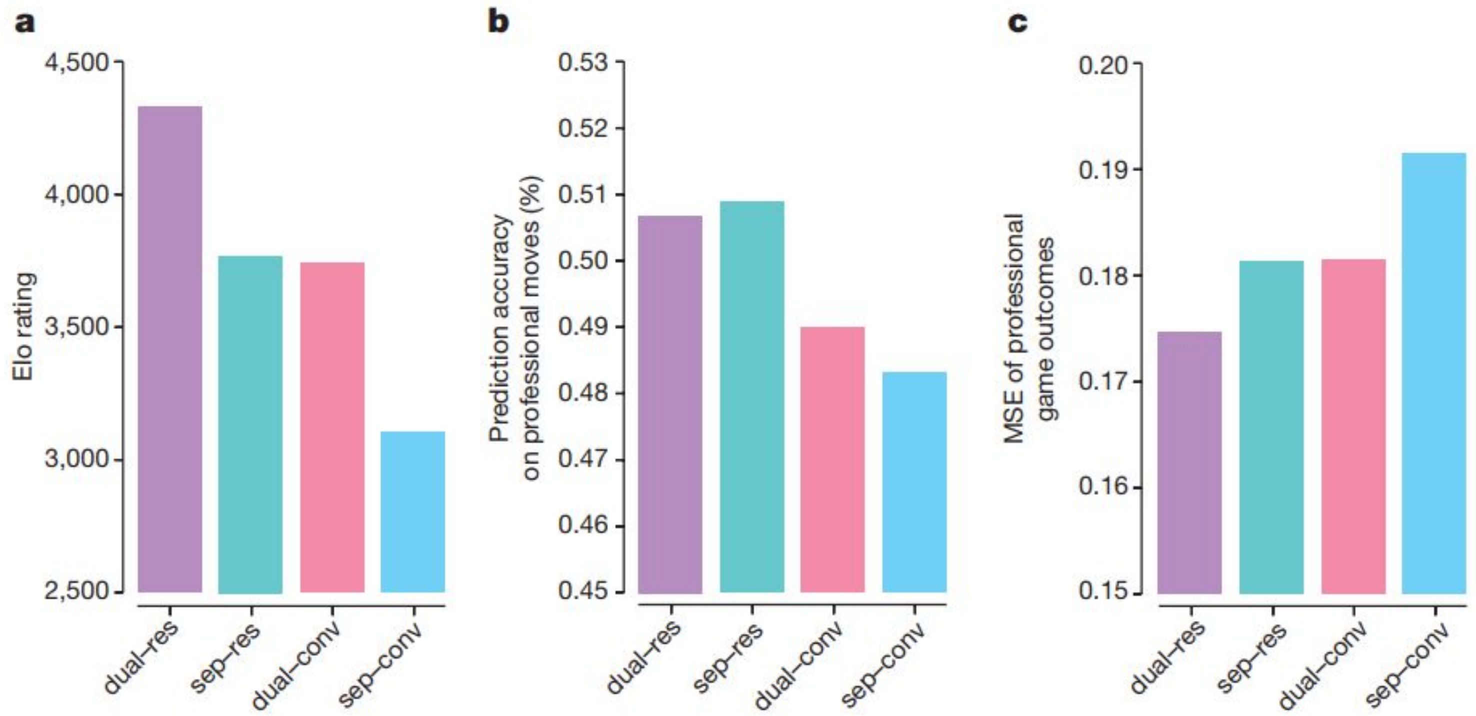
$$\mathbb{E}_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}$$

$$R'_A = R_A + K \cdot (S_A - \mathbb{E}_A)$$

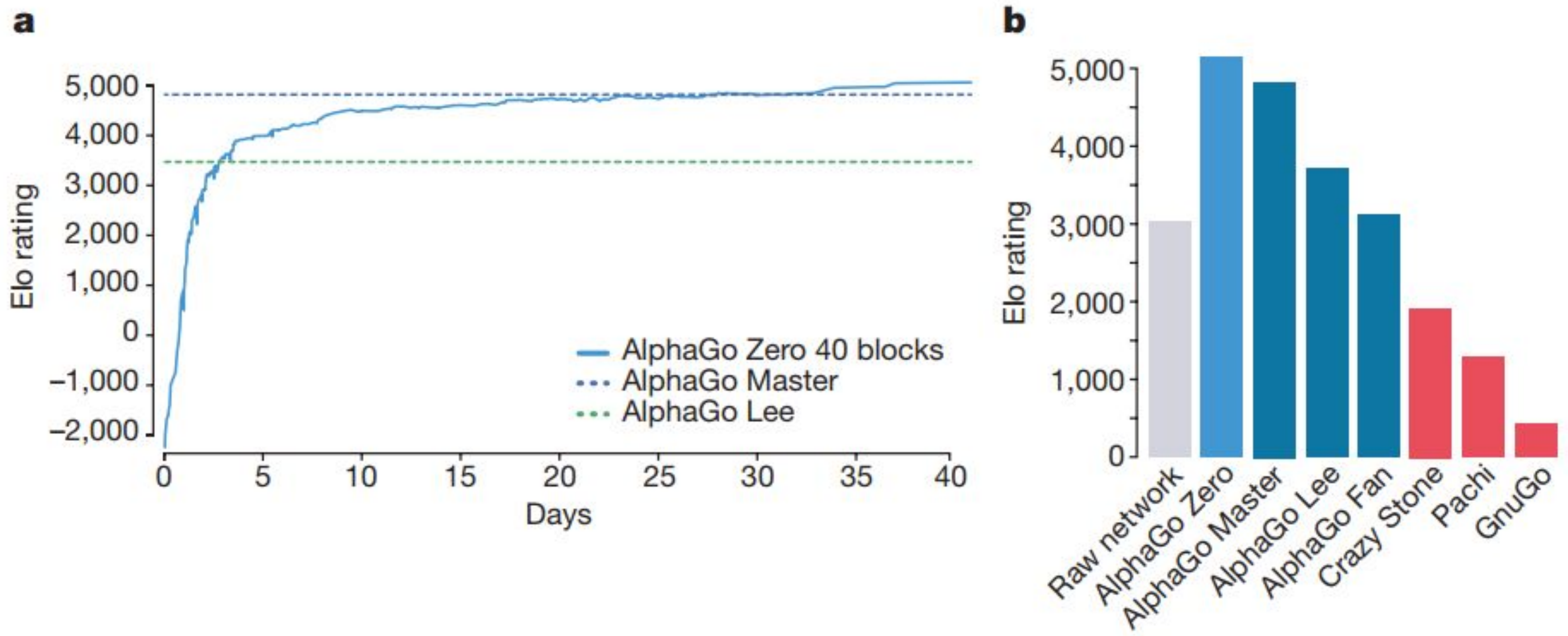
Neural networks versions

- dual–res: the network contains a 20-block residual tower, followed by both a policy head and a value head. This is the architecture used in AlphaGo Zero.
- sep–res: the network contains two 20-block residual towers. The first tower is followed by a policy head and the second tower is followed by a value head.
- dual–conv: the network contains a non-residual tower of 12 convolutional blocks, followed by both a policy head and a value head.
- sep–conv: the network contains two non-residual towers of 12 convolutional blocks. The first tower is followed by a policy head and the second tower is followed by a value head. This is the architecture used in AlphaGo Lee.

Neural networks comparison



AlphaGo versions comparison



Why is it better?

- Avoid noisy human data
- Uses only one network
- Better hardware usage

References

- Mastering the game of Go without human knowledge (https://deepmind.com/research/publications/mastering-game-gowithout-human-knowledge)
- AlphaGo Zero explained in one diagram (https://medium.com/applied-data-science/alphago-zero-explained-in-one-diagram-365f5abf67e0)