solving go: 2019

brettkoonce.com/talks

feburary 24th, 2019

go: overview

- discuss game, rules
- uct + random rollouts —> MCTS
- MCTS + policy + value —> Alpha Go
- policy + value + self-play —> Alpha Go Zero
- Alpha Go Zero - Go —> Alpha Zero, demo

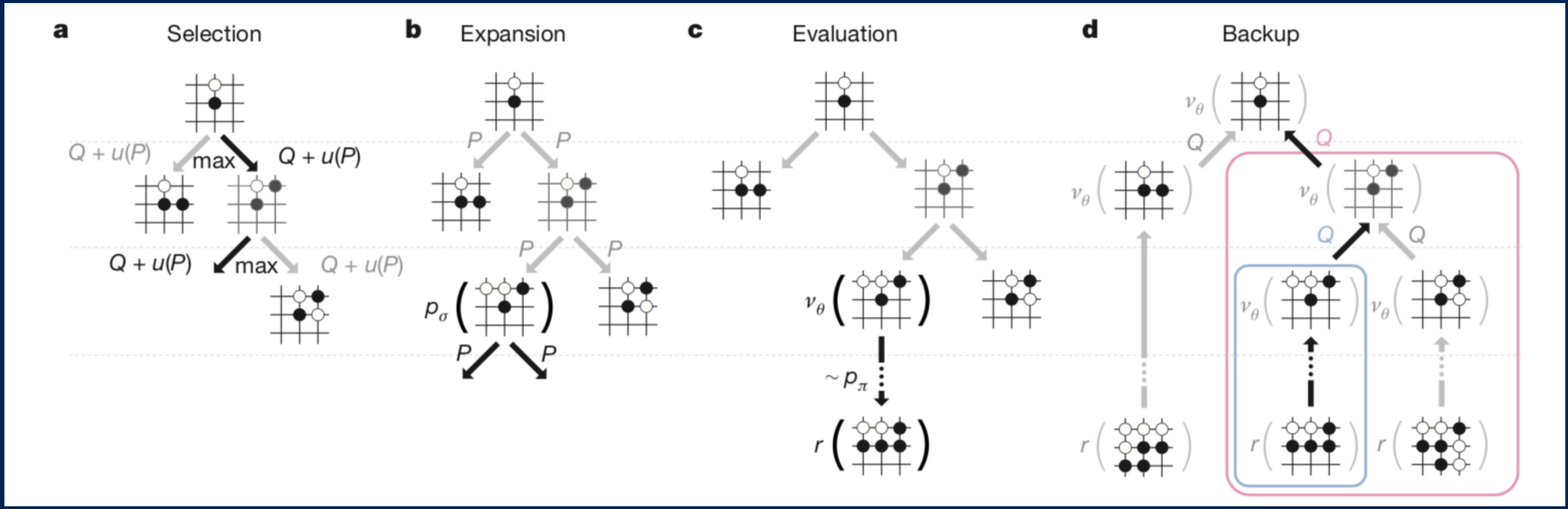
go: board

go: game/rules

- origin: asia, ~2500 years ago
- 19x19 board (361 squares), fill with stones
- squares + captures —> score (chinese)
- black - 7.5 (komi) > white —> winner (no draws)
- ~300 moves —> ~1e170 game complexity (chess: ~1e50, # atoms in universe: ~1e80)

uct (2006)

- multi-armed bandit problem: how do you win most money from room of slot machines, given X quarters?
- basic idea: explore new machines (policy), calculate reward of a machine (value)
- ucb: put statistical bound on losses —> maximize gains



random rollouts (2009)

- take current board state, pick candidate move to explore/evaluate
- alternate adding stones randomly till both sides cannot play (pass)
- score via chinese rules —> move win/loss predictor —> update value of candidate move
- uct + random rollouts —> mcts —> solves go! (minor bug: universe will die of heat death first)

alpha go (2016)

alpha go: fan/lee/master

- engine/year: fan/2015, lee/2016, master/2017
- take mcts approach, but improve search:
  - use policy network to quickly make moves (test good moves rather than random ones)
  - use value network to predict winning odds (cheaper estimates for faster exploration)
  - finally, use mcts to perform deeper evaluation as needed

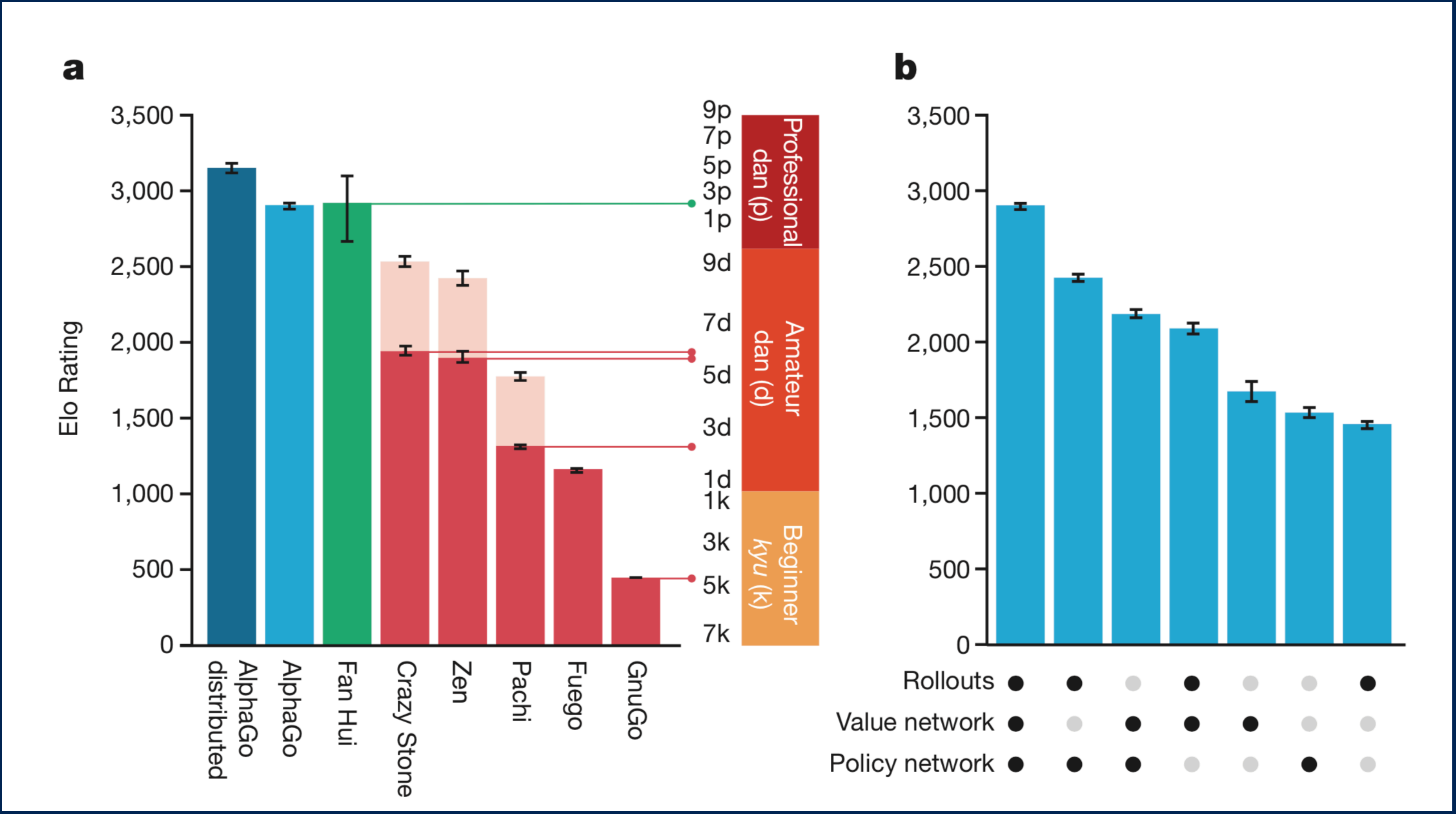
policy network

- human games (~150k) + supervised learning —> cnn policy network (given position, predict next move)
- use policy network + MCTS —> play more games (human + computer) —> train again —> better policy network (e.g. reinforcement learning)
- use policy network to make moves rapidly —>
  - 55% accuracy in 3ms, 24% accuracy in 2µs
  - policy network alone can defeat many engines*

value network

- from given input state, can we predict who will win, without performing a rollout simulation?
- build CNN to predict winning probability %
- train: mse between prediction and outcome
- overtrains to input games, so have to relax network (e.g. rotate/flip games)
- use value network to predict expected win/loss of moves without rollout (15000x faster)

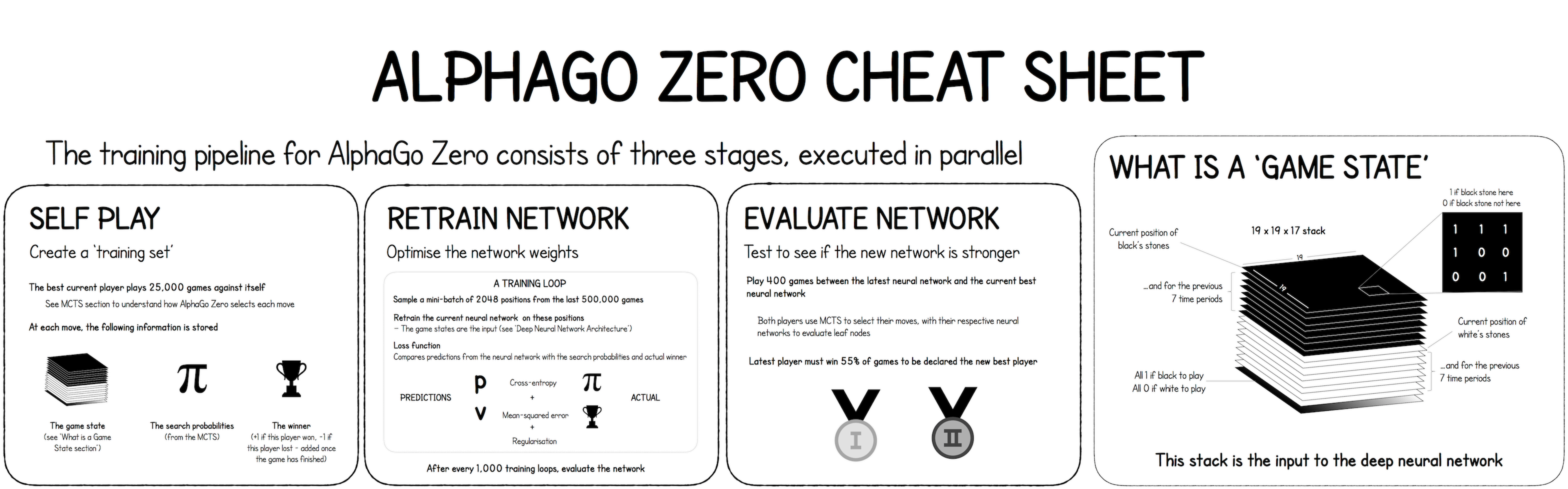
alpha go: performance



alpha go: zero (2017)

- input a position —> use single network (combined policy + value) to predict best move and winning odds —> build game tree
- play games against self (tabula rasa), train new network to categorize wins/losses and reduce prediction error
- evaluate new network against old, pick winner
- repeat 700k generations —> master level play

applied-data.science/blog/alphago-zero-cheat-sheet

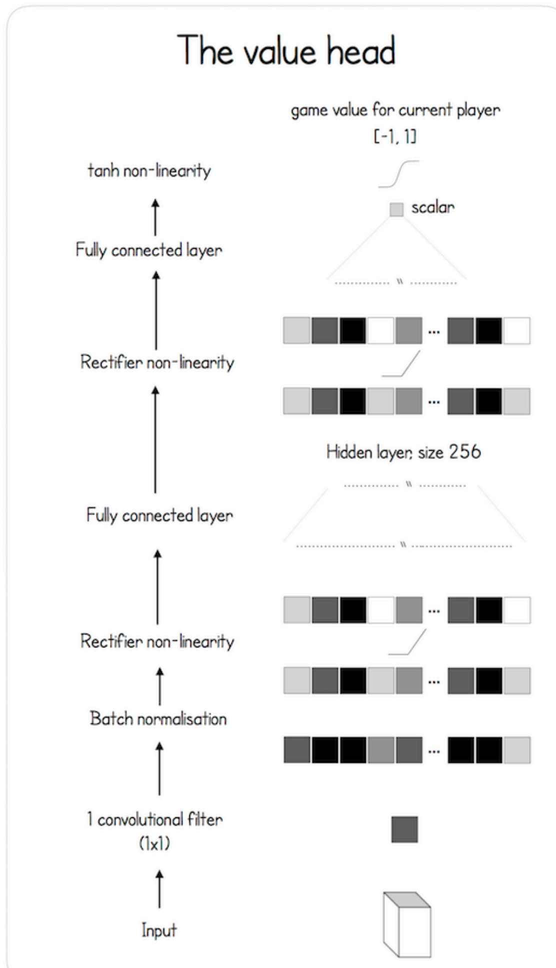


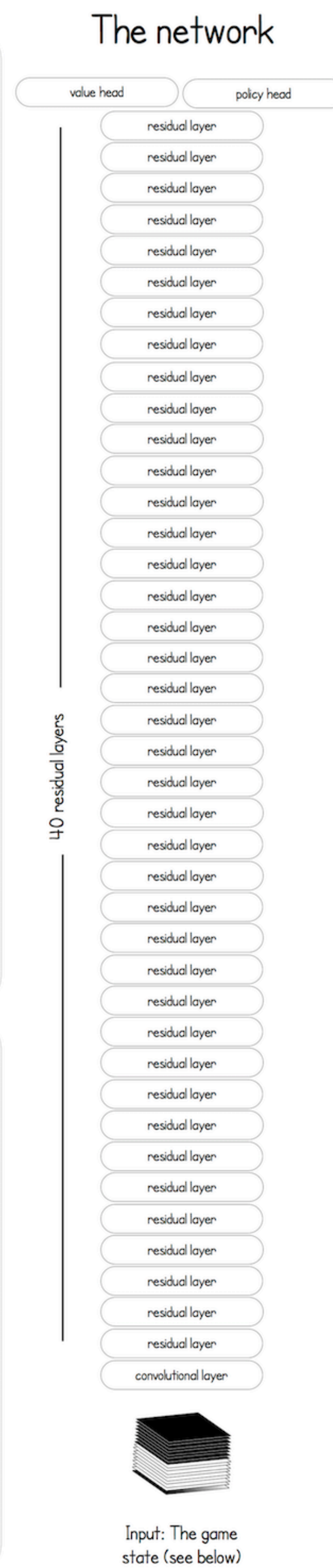
THE DEEP NEURAL NETWORK ARCHITECTURE

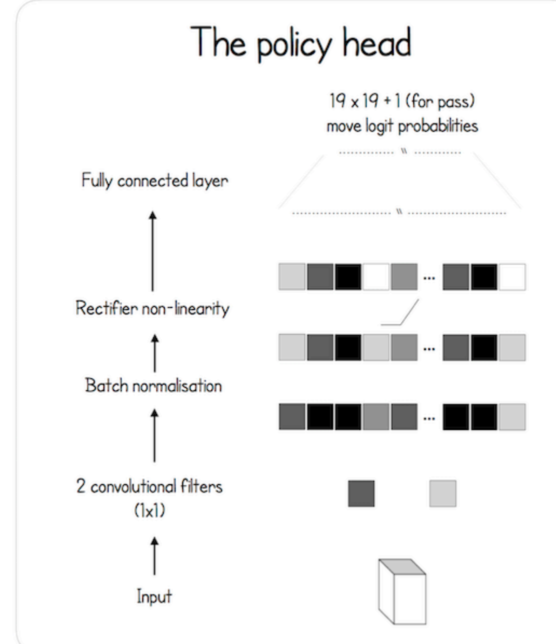
How AlphaGo Zero assesses new positions

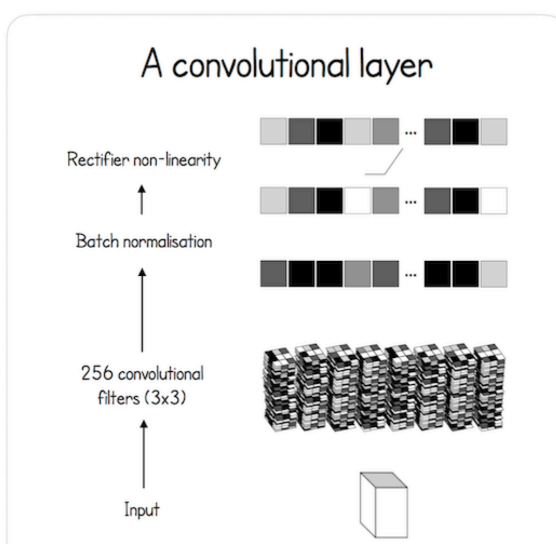
The network learns 'tabula rasa' (from a blank slate)

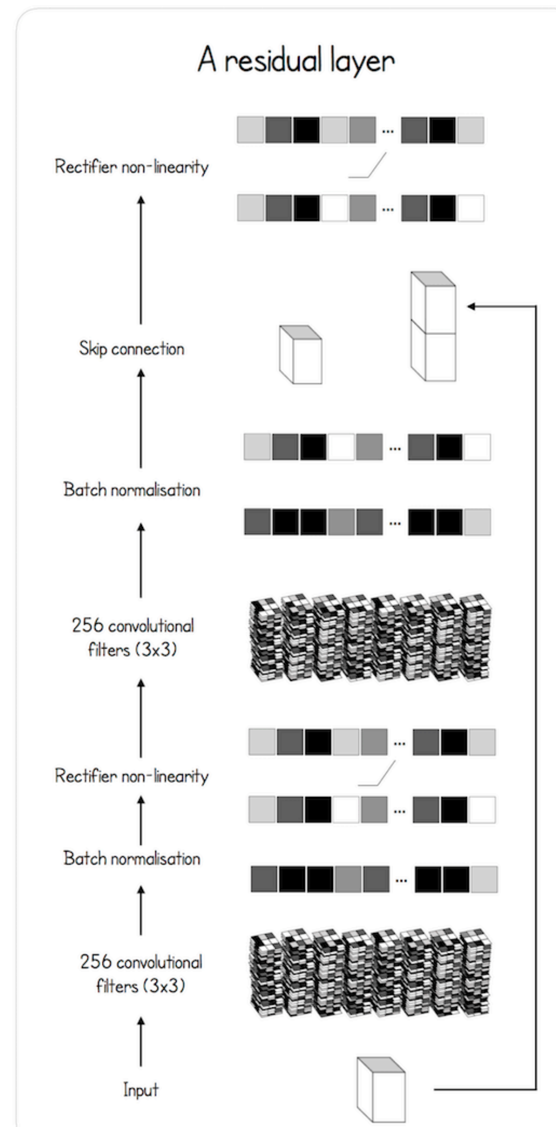
At no point is the network trained using human knowledge or expert moves





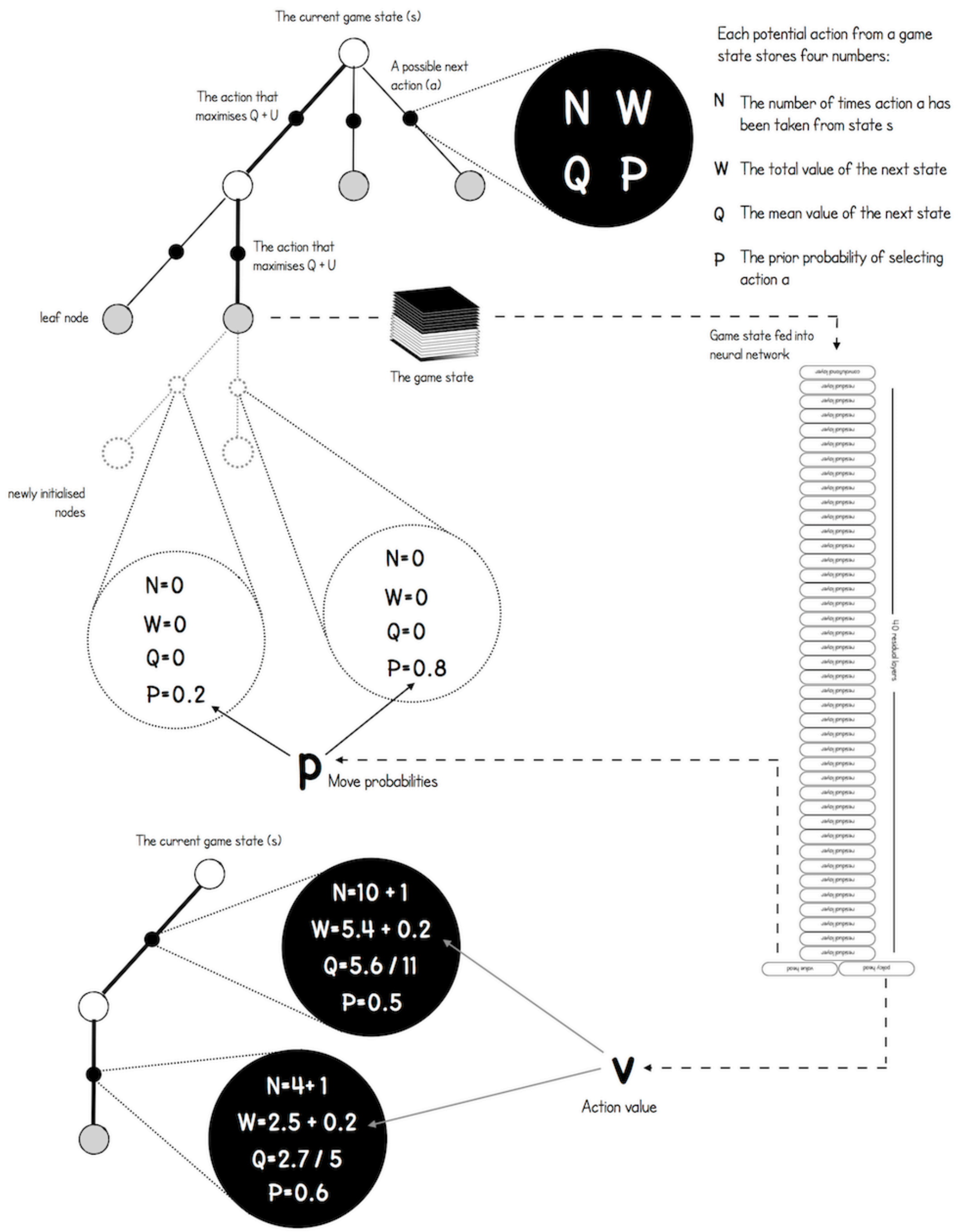






MONTE CARLO TREE SEARCH (MCTS)

How AlphaGo Zero chooses its next move



First, run the following simulation 1,600 times…

Start at the root node of the tree (the current game state)

1. Choose the action that maximises…

$$Q + U$$

The mean value of the next state

A function of P and N that increases if an action hasn't been explored much, relative to the other actions, or if the prior probability of the action is high

Early on in the simulation, U dominates (more exploration), but later, Q is more important (less exploration)

2. Continue until a leaf node is reached

The game state of the leaf node is passed into the neural network, which outputs predictions about two things:

p Move probabilities

v Value of the state (for the current player)

The move probabilities p are attached to the new feasible actions from the leaf node

3. Backup previous edges

Each edge that was traversed to get to the leaf node is updated as follows:

$$N \rightarrow N + 1$$
$$W \rightarrow W + v$$
$$Q = W / N$$

…then select a move

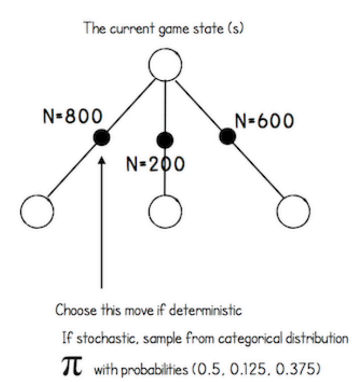
After 1,600 simulations, the move can either be chosen:

Deterministically (for competitive play)
Choose the action from the current state with greatest N

Stochastically (for exploratory play)
Choose the action from the current state from the distribution

$$\pi \sim N^{1/\tau}$$

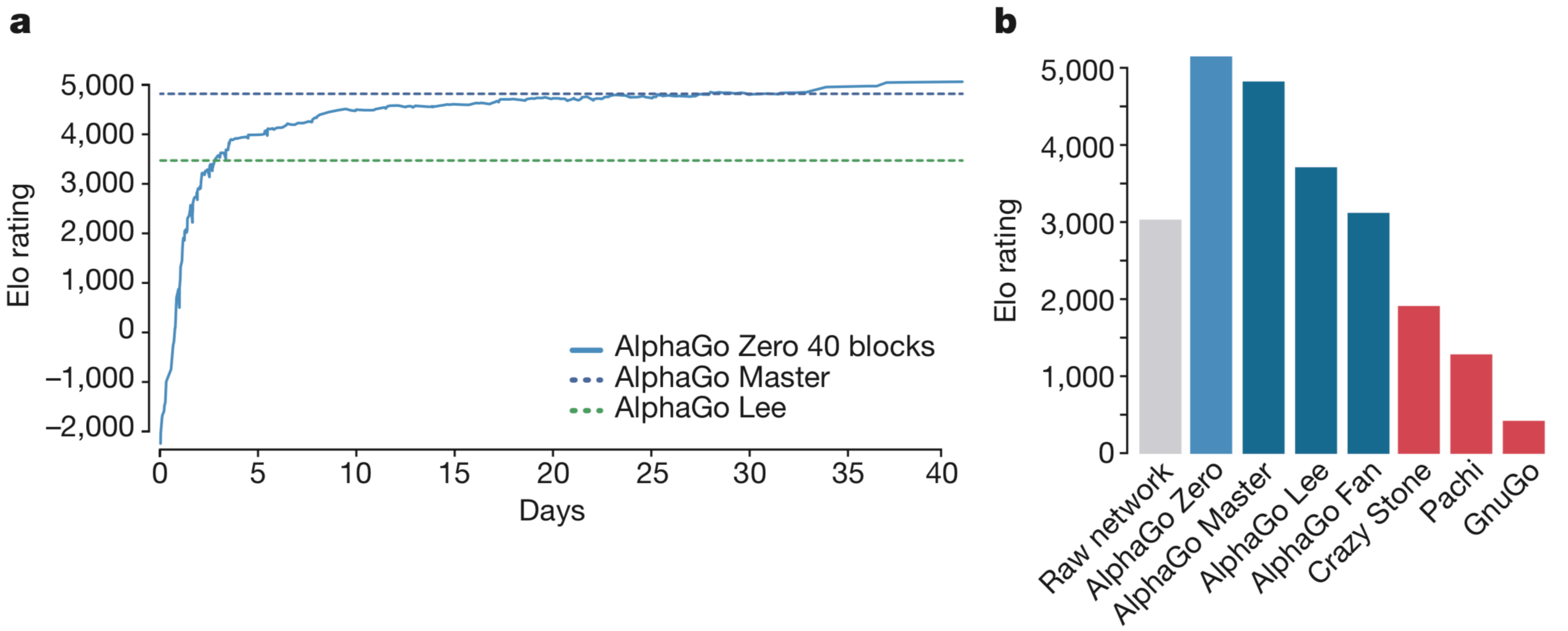
where τ is a temperature parameter, controlling exploration



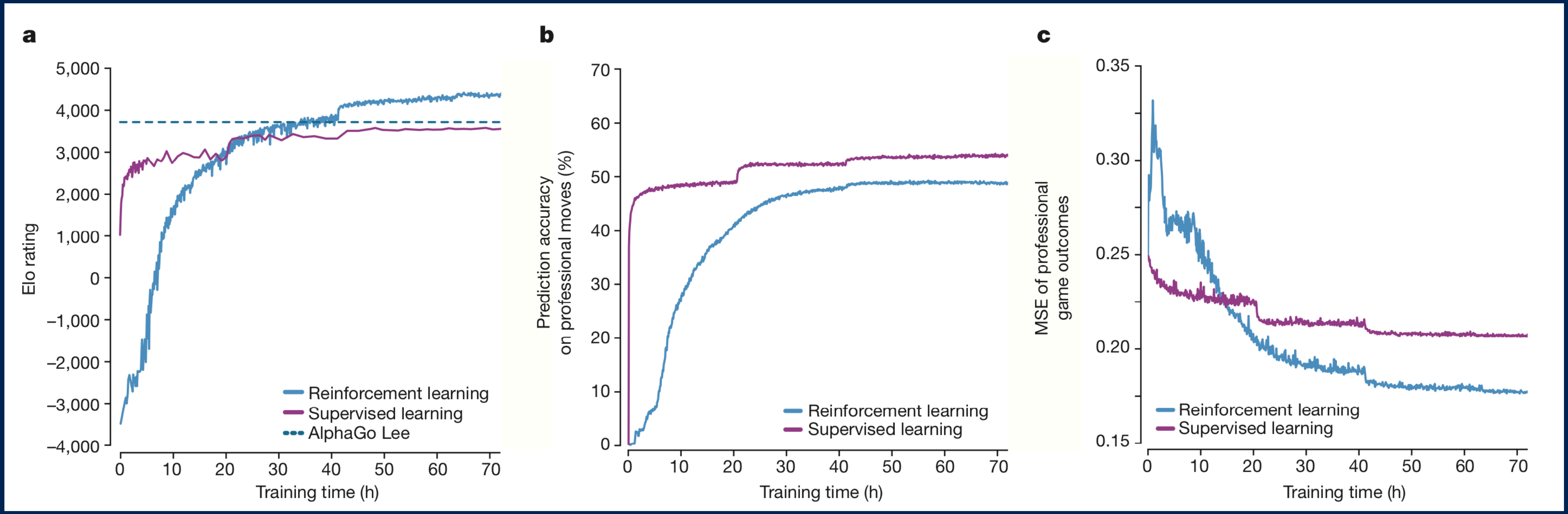
Other points

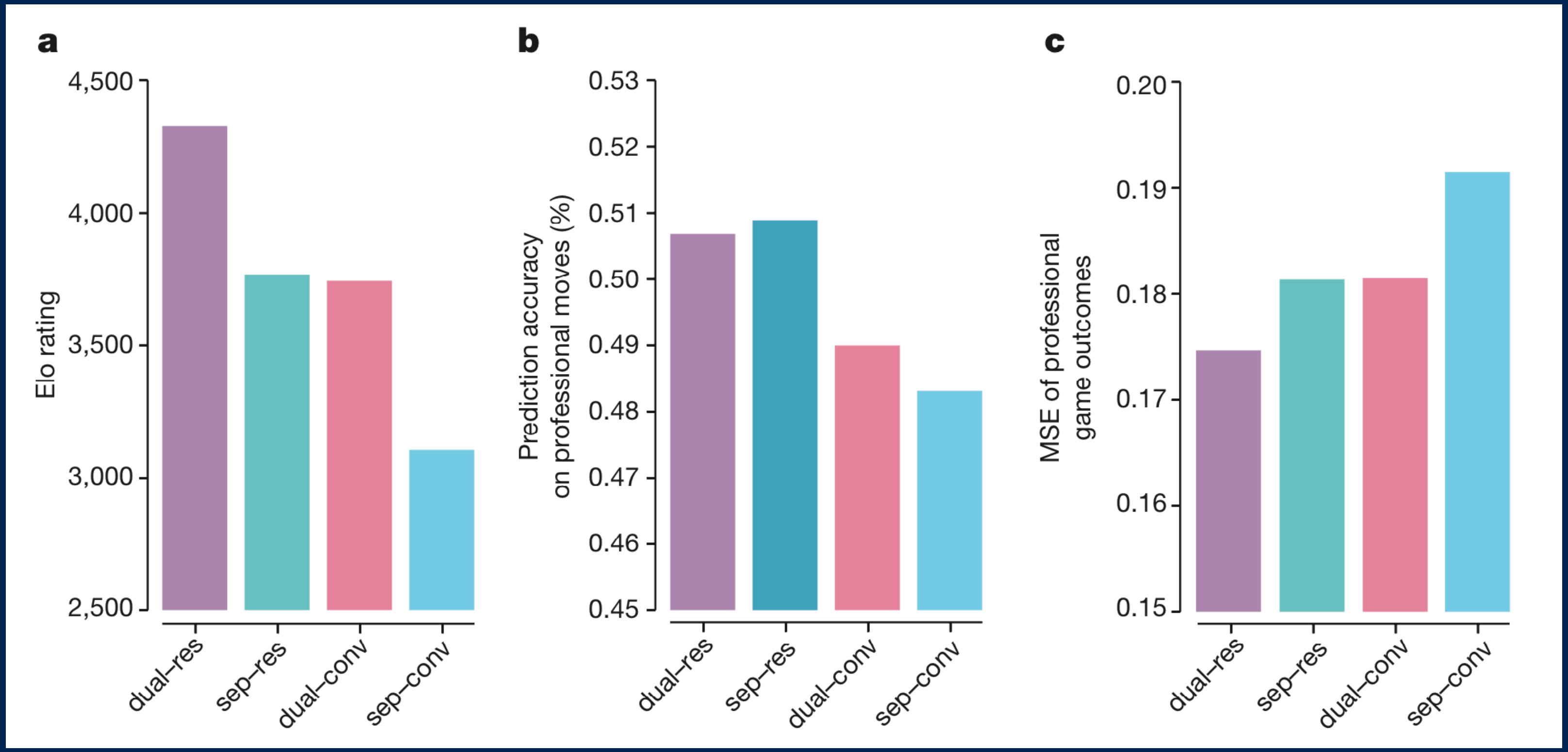
- The sub-tree from the chosen move is retained for calculating subsequent moves
- The rest of the tree is discarded

alpha go zero: train

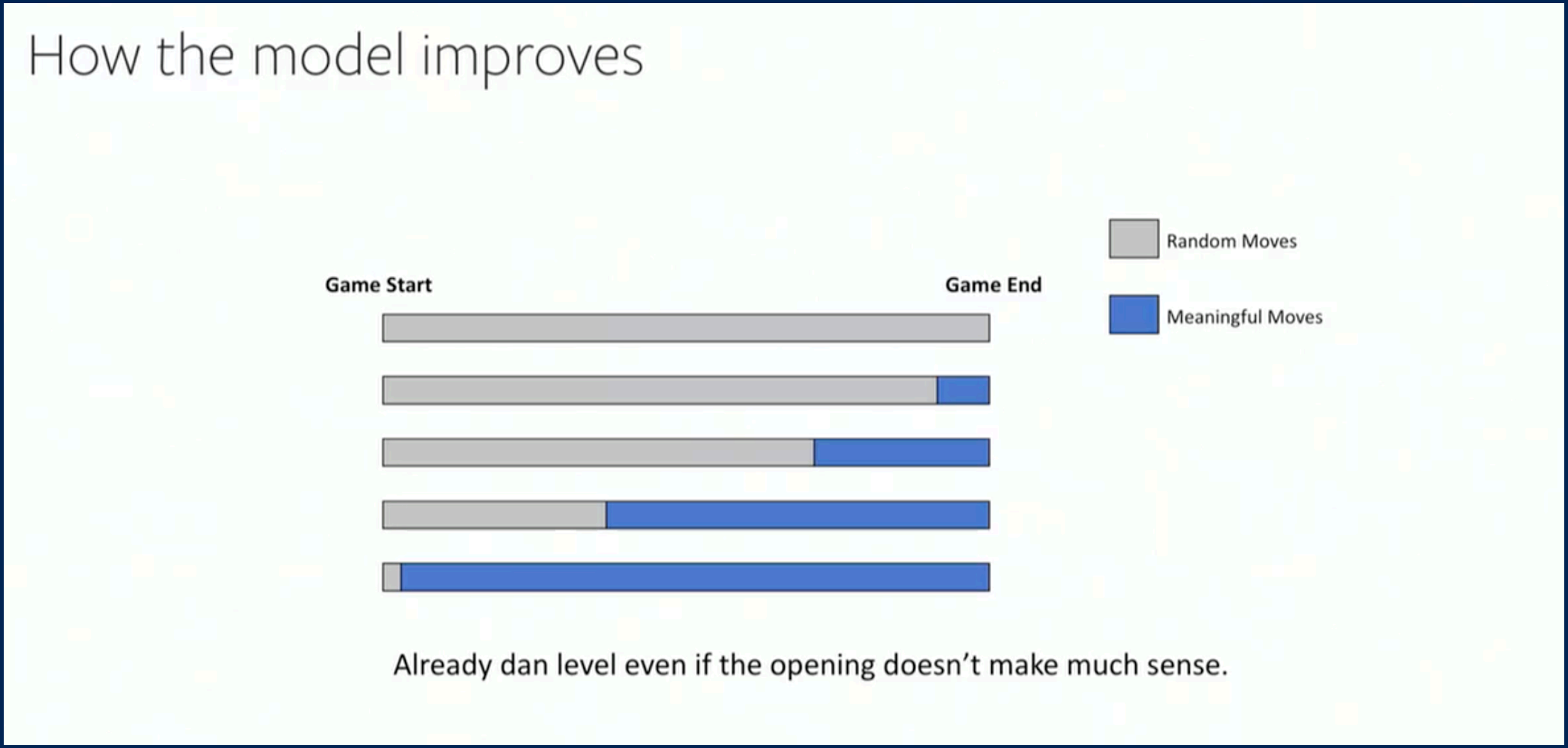


rl v. sl + resnet v. cnn





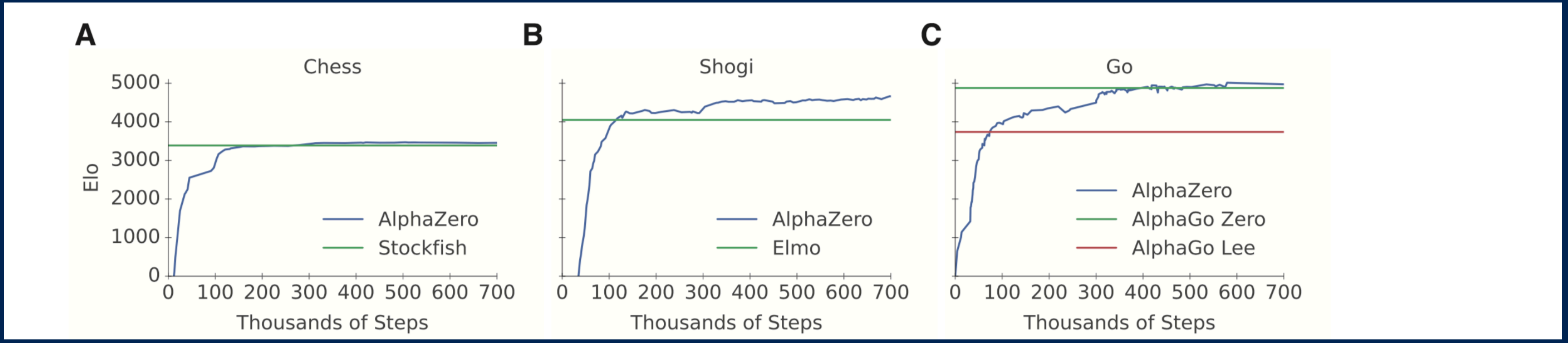
end to end learning



- pytorch Yuandong Tian talk (october 18)

alpha zero (dec 18)

- generalized version of alpha go approach (no go-specific knowledge)
- input board state, possible moves, evaluation function —> generates policy/value networks via self play
- teaches self how to play, improves to master level



alpha zero

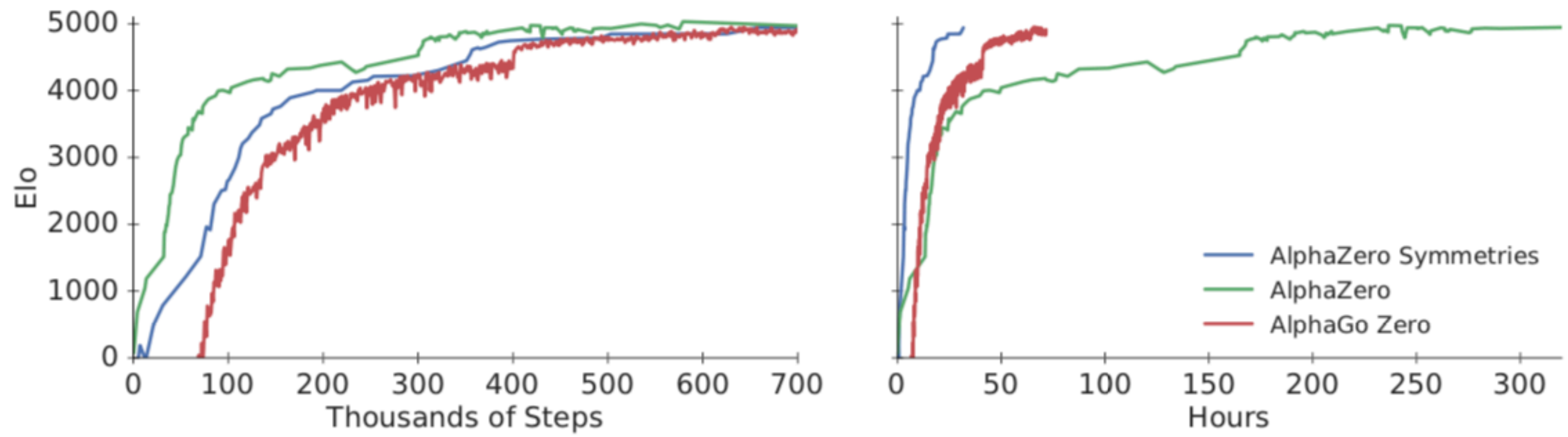


Figure S1: **Learning curves showing the Elo performance during training in Go.** Comparison between AlphaZero, a version of AlphaZero that exploits knowledge of symmetries in a similar manner to AlphaGo Zero, and the previously published AlphaGo Zero. AlphaZero generates approximately $1/8$ as many positions per training step, and therefore uses eight times more wall clock time, than the symmetry-augmented algorithms.

- symmetries (inverse relaxation)

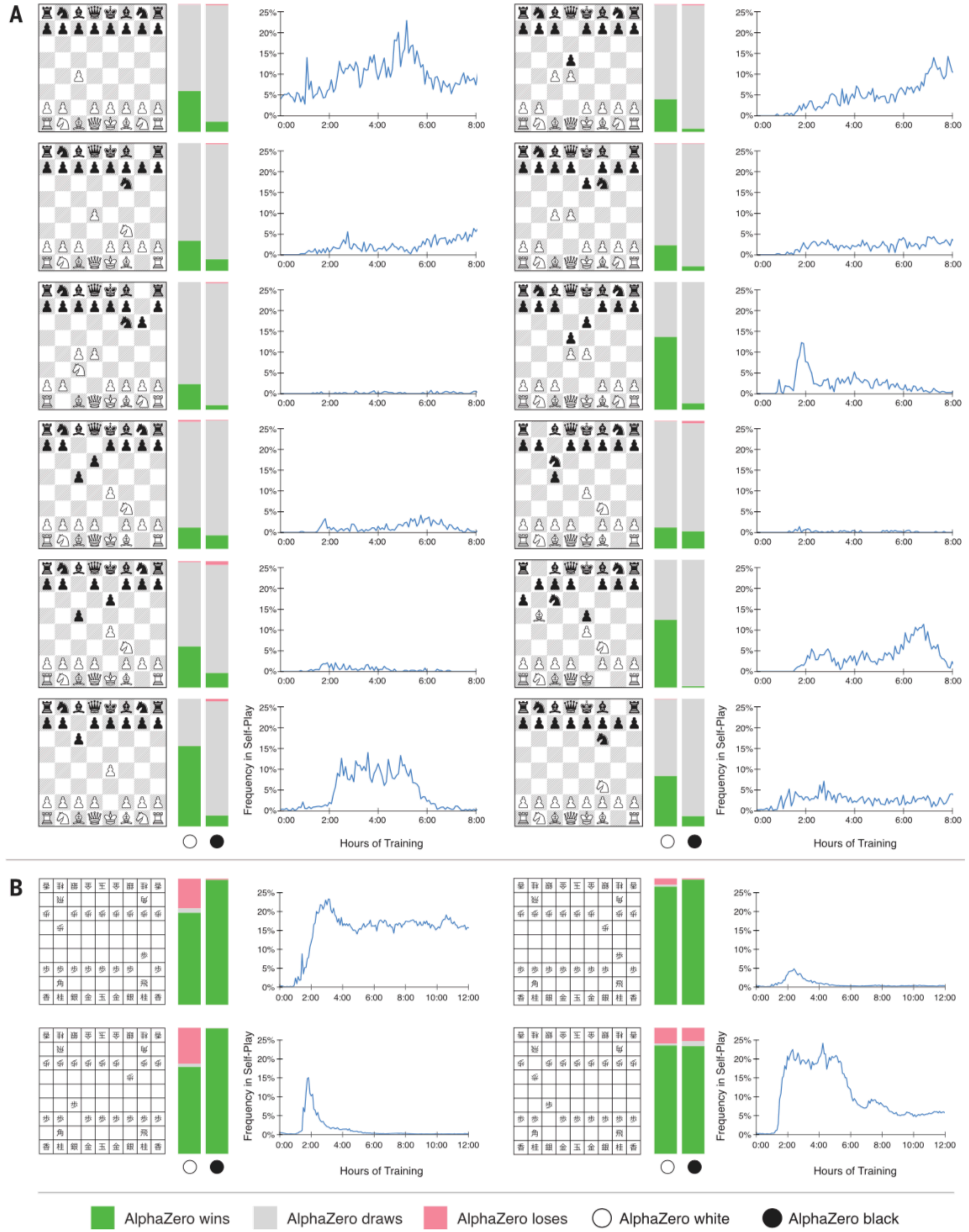
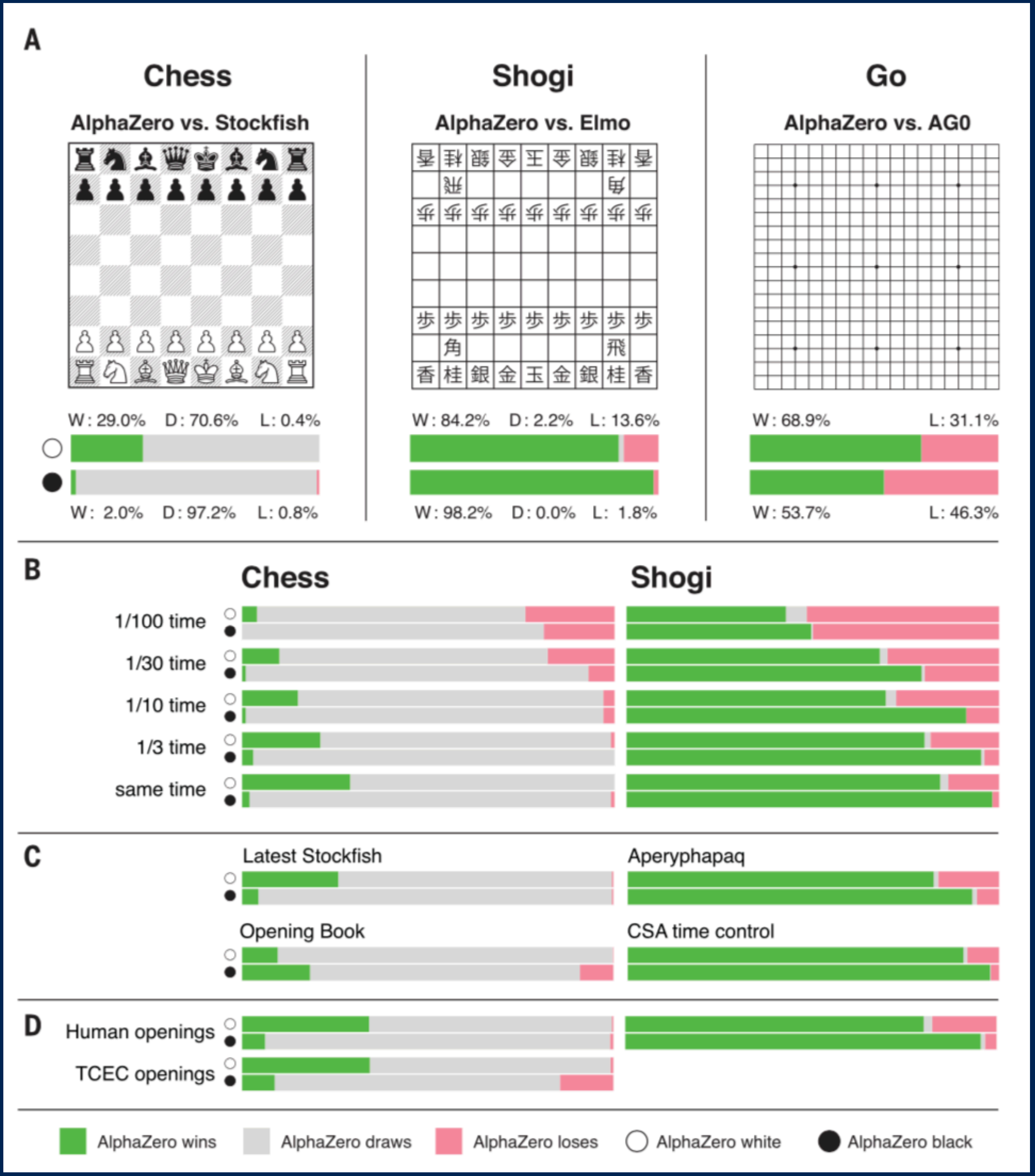


Fig. 3. Matches starting from the most popular human openings. AlphaZero plays against (**A**) Stockfish in chess and (**B**) Elmo in shogi. In the left bar, AlphaZero plays white, starting from the given position; in the right bar, AlphaZero plays black. Each bar shows the results from AlphaZero's perspective: win (green), draw (gray), or loss (red). The percentage frequency of self-play training games in which this opening was selected by AlphaZero is plotted against the duration of training, in hours.


A
Chess
AlphaZero vs. Stockfish
W: 29.0% D: 70.6% L: 0.4%
W: 2.0% D: 97.2% L: 0.8%
Shogi
AlphaZero vs. Elmo
W: 84.2% D: 2.2% L: 13.6%
W: 98.2% D: 0.0% L: 1.8%
Go
AlphaZero vs. AG0
W: 68.9% L: 31.1%
W: 53.7% L: 46.3%
B
Chess
Shogi
1/100 time
1/30 time
1/10 time
1/3 time
same time
C
Latest Stockfish
Aperyphapaq
Opening Book
CSA time control
D
Human openings
TCEC openings
AlphaZero wins
AlphaZero draws
AlphaZero loses
AlphaZero white
AlphaZero black

tic-tac-toe

- github.com/suragnair/alpha-zero-general
- thanks Surag Nair, Evgeny Tyurin!
- input: board, moves, evaluate
- loop: play games against self, train (keras/tensorflow) to recognize winners/minimize losses, evaluate new network, repeat
- alpha zero demo: play, train, test

recap

- mcts (uct + rollouts) “solves” go, but doesn’t scale
- combine expert knowledge (prior games) with value and policy networks (optimization) to surpass human players
- a single network randomly initialized can reach even greater performance via self-play
- this approach generalizes to other domains and is very human like

what is ai?

- chinese room example
- give beginner a board, rules, have them practice
- difference between master and beginner: knows what to seek, what to avoid —> they have experience
- casablanca: how many moves ahead do you think?

thanks for coming!

papers

- Reinforcement Learning and Simulation-Based Search in Computer Go (2009)
- Mastering the game of Go with deep neural networks and tree search (2016)
- Mastering the game of Go without human knowledge (2017)
- A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play (2018)