

solving go

by brett koonce
march 6, 2018

static.brettkoonce.com/presentations/go.pdf

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

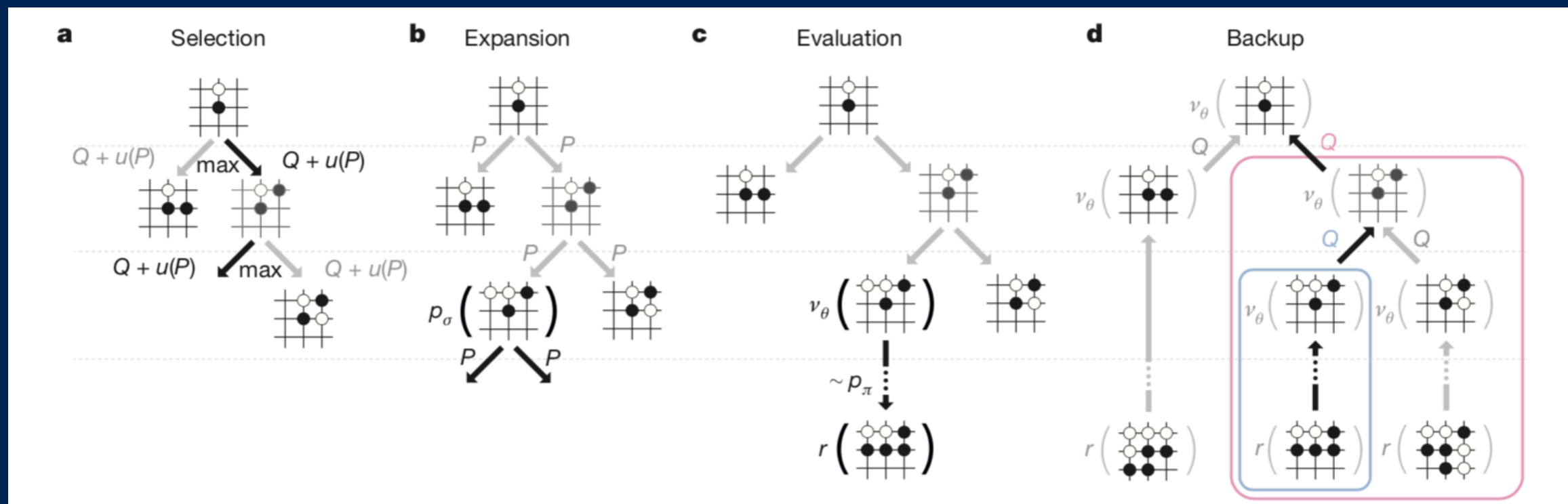


go: game/rules

- **origin: asia, ~2500 years ago**
- **19x19 board (361 squares), fill with stones**
- **squares + captures \rightarrow score (chinese)**
- **black - 7.5 (komi) $>$ white \rightarrow winner (no draws)**
- **~300 moves \rightarrow ~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 \rightarrow 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**

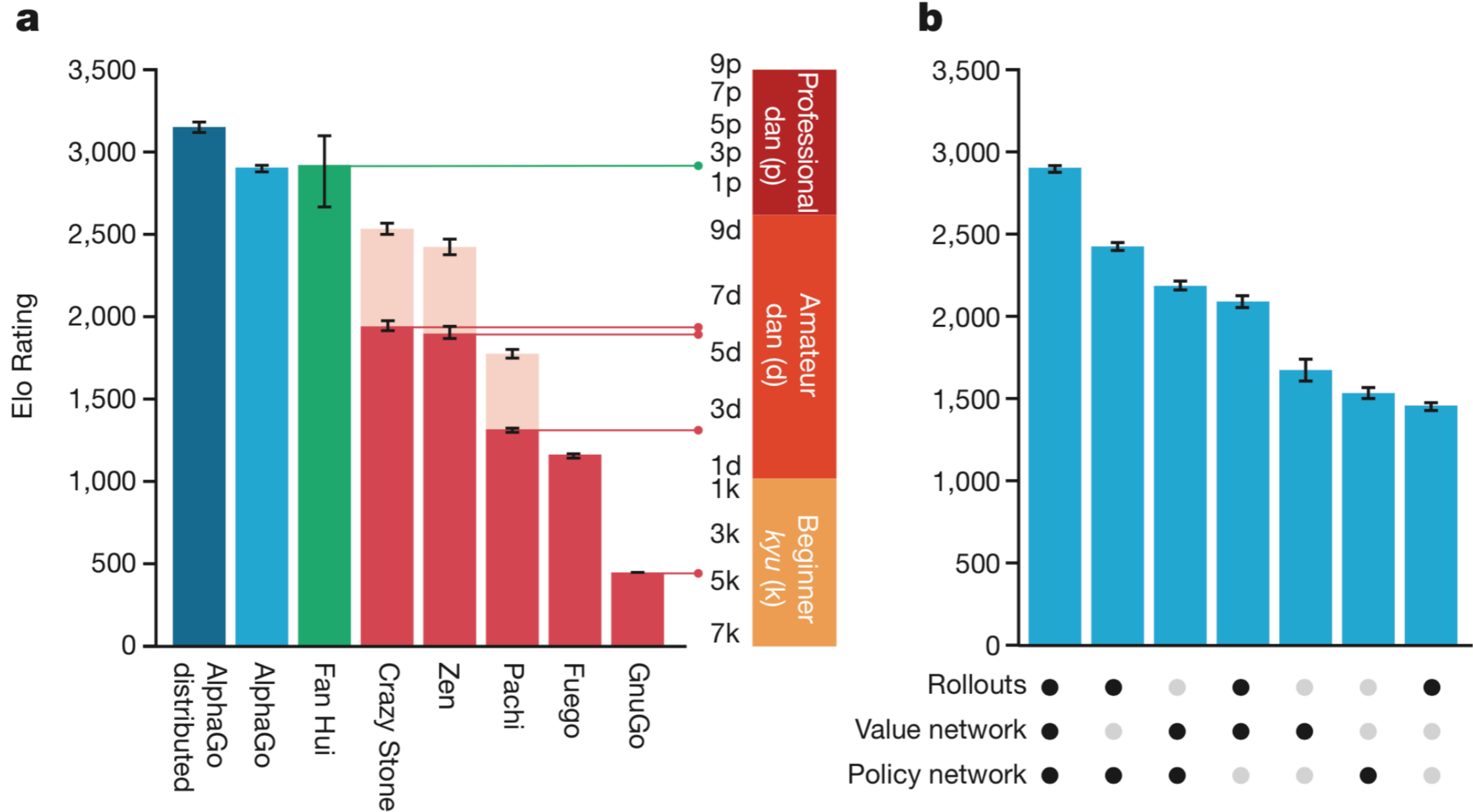
policy network

- 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

ALPHAGO ZERO CHEAT SHEET

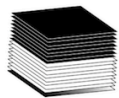
The training pipeline for AlphaGo Zero consists of three stages, executed in parallel

SELF PLAY

Create a 'training set'

The best current player plays 25,000 games against itself
See MCTS section to understand how AlphaGo Zero selects each move

At each move, the following information is stored



The game state
(see 'What is a Game State' section')

 π

The search probabilities
(from the MCTS)



The winner
(+1 if this player won, -1 if this player lost - added once the game has finished)

RETRAIN NETWORK

Optimise the network weights

A TRAINING LOOP

Sample a mini-batch of 2048 positions from the last 500,000 games

Retrain the current neural network on these positions
- The game states are the input (see 'Deep Neural Network Architecture')

Loss Function

Compares predictions from the neural network with the search probabilities and actual winner

$$\text{PREDICTIONS} \begin{matrix} \mathbf{p} \\ \mathbf{v} \end{matrix} + \begin{matrix} \text{Cross-entropy} \\ \text{Mean-squared error} \\ \text{Regularisation} \end{matrix} + \begin{matrix} \mathbf{\pi} \\ \text{trophy} \end{matrix} \text{ ACTUAL}$$

After every 1,000 training loops, evaluate the network

EVALUATE NETWORK

Test to see if the new network is stronger

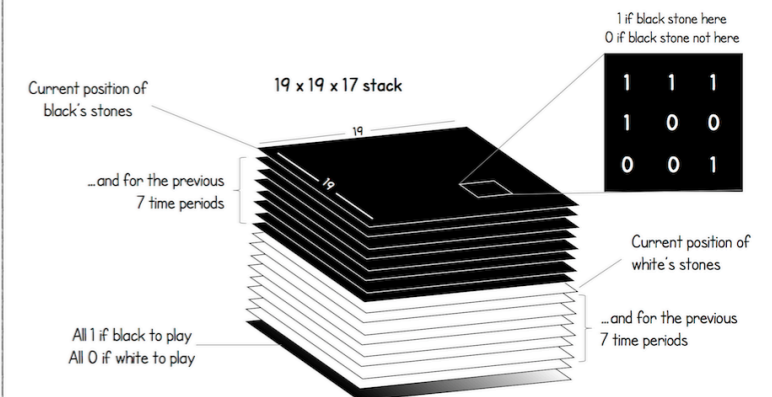
Play 400 games between the latest neural network and the current best neural network

Both players use MCTS to select their moves, with their respective neural networks to evaluate leaf nodes

Latest player must win 55% of games to be declared the new best player

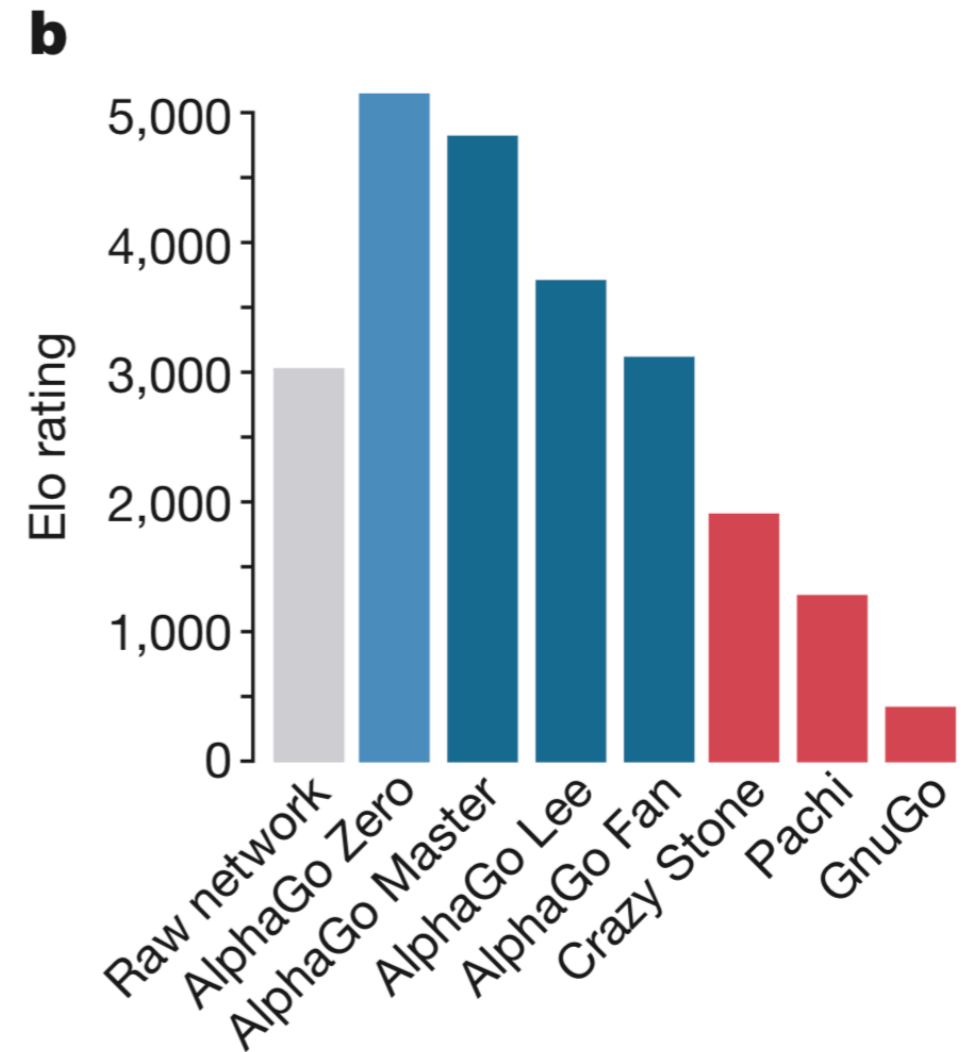
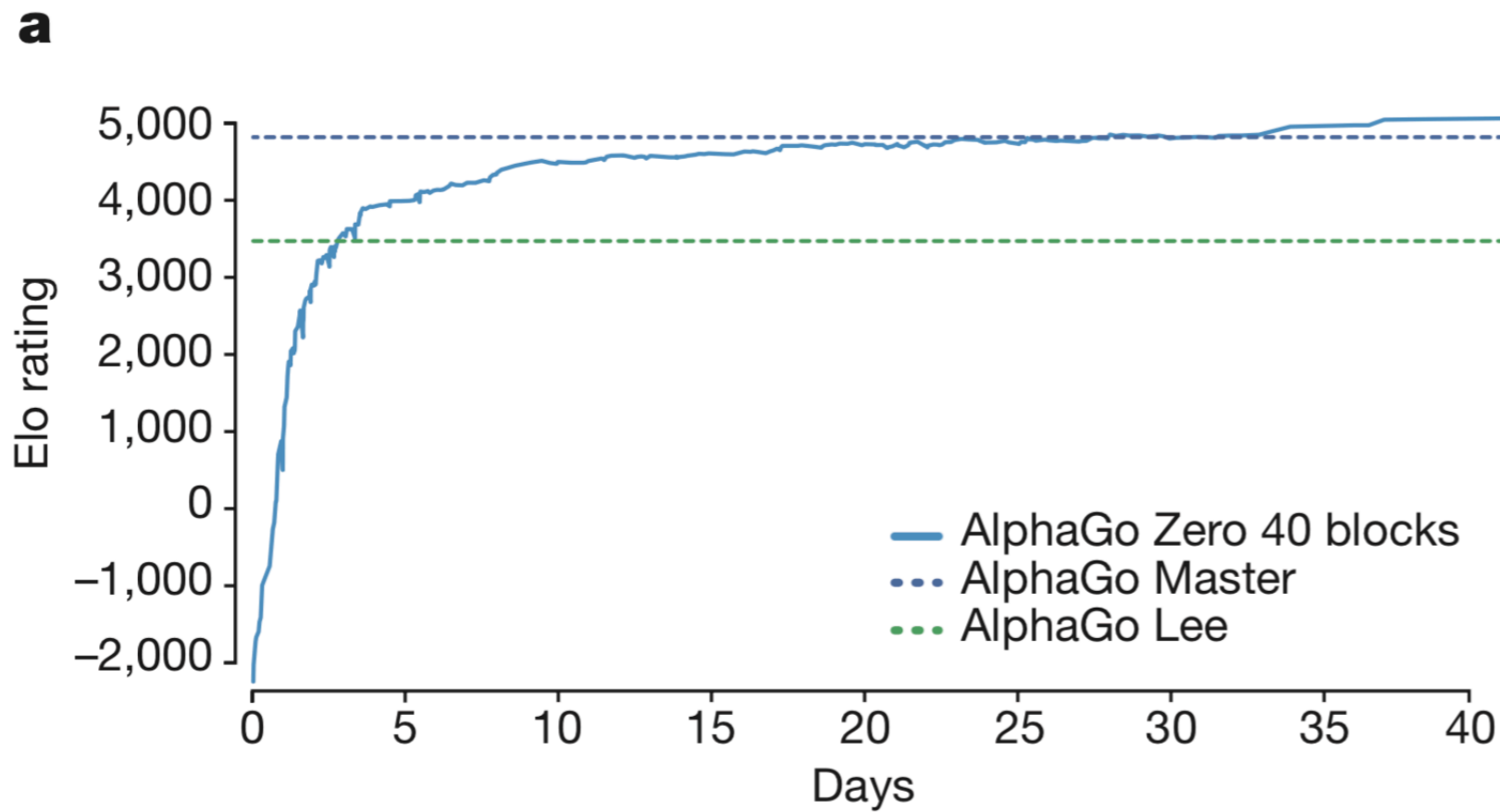


WHAT IS A 'GAME STATE'

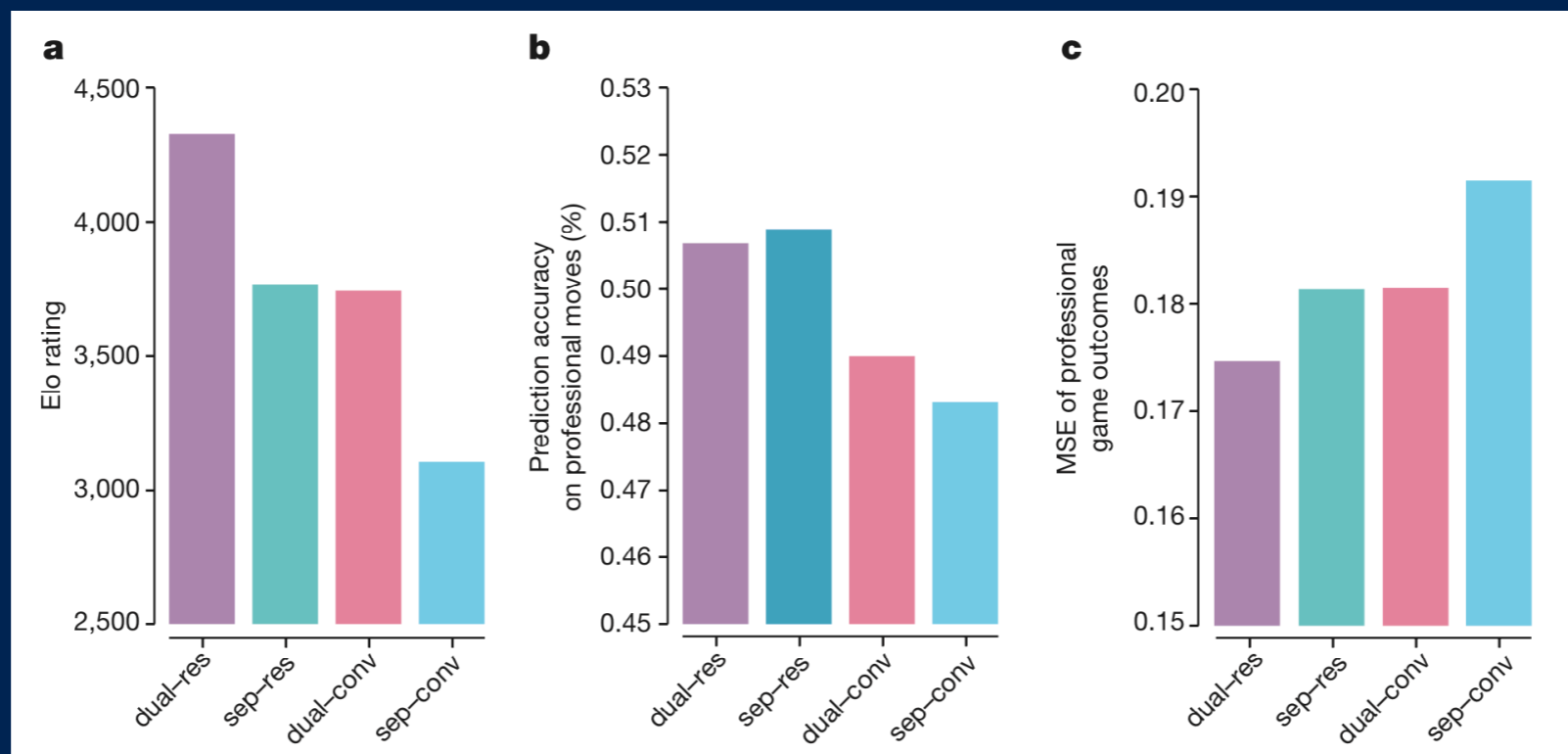
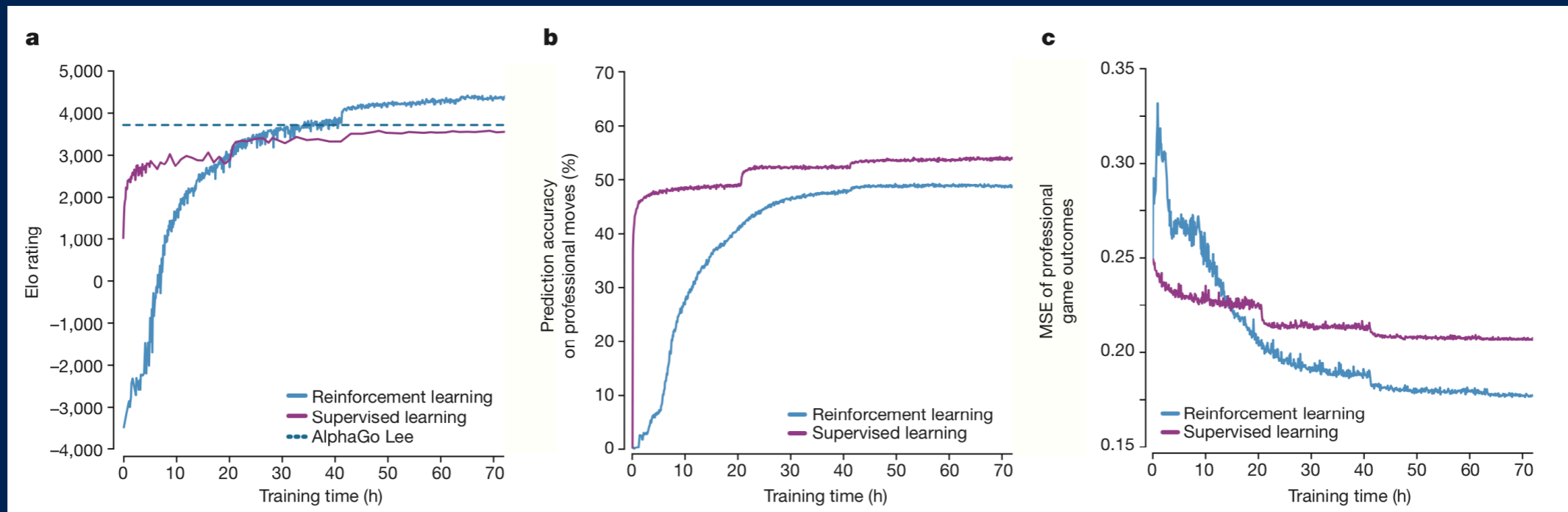


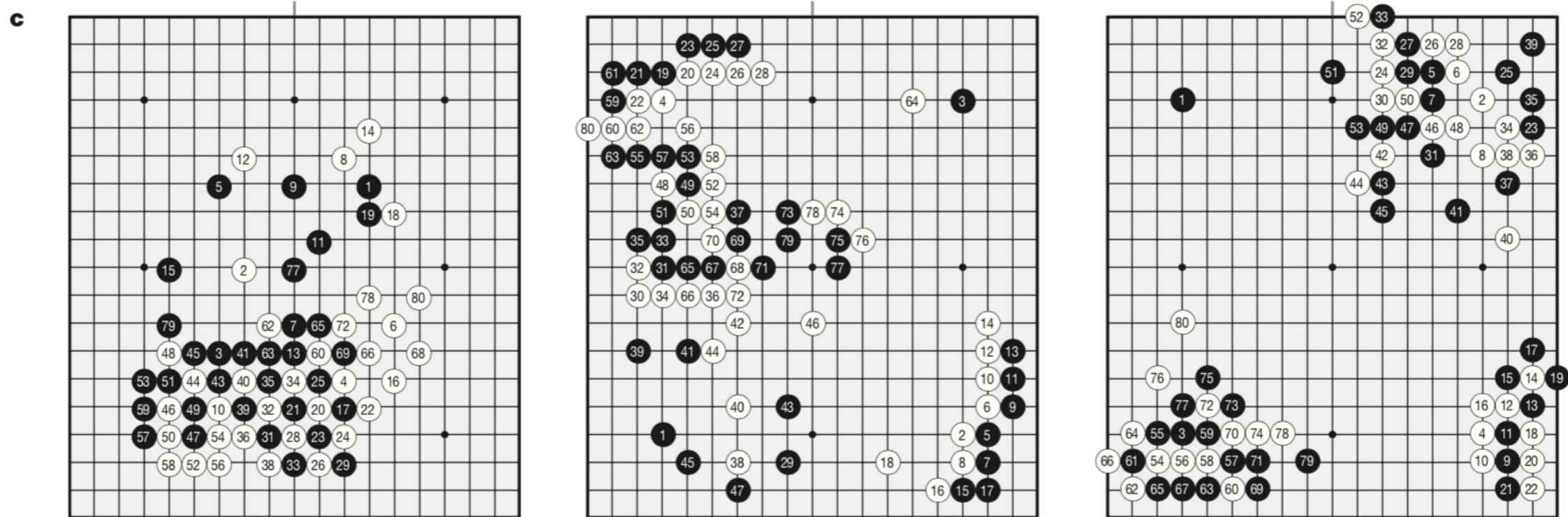
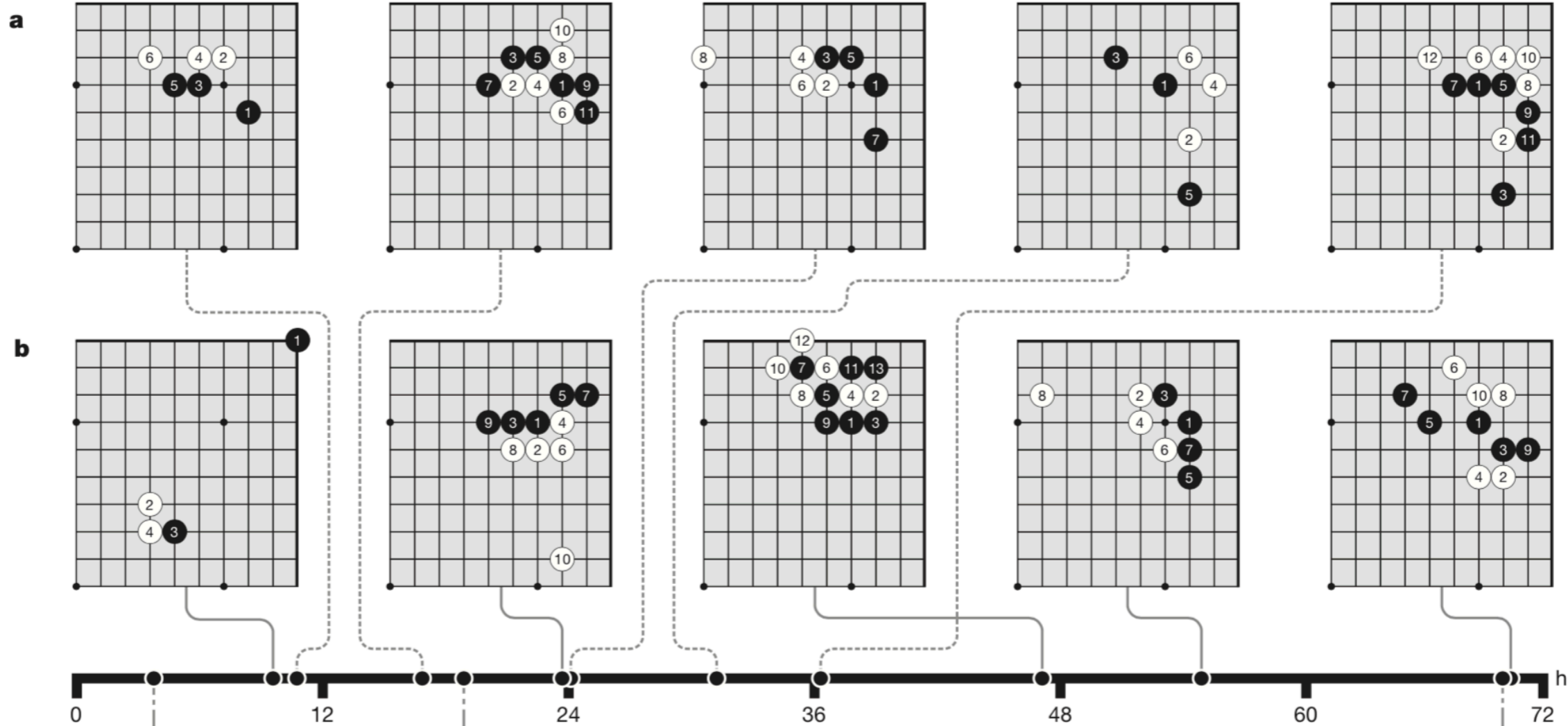
This stack is the input to the deep neural network

alpha go zero: train



rl v. sl + resnet v. cnn

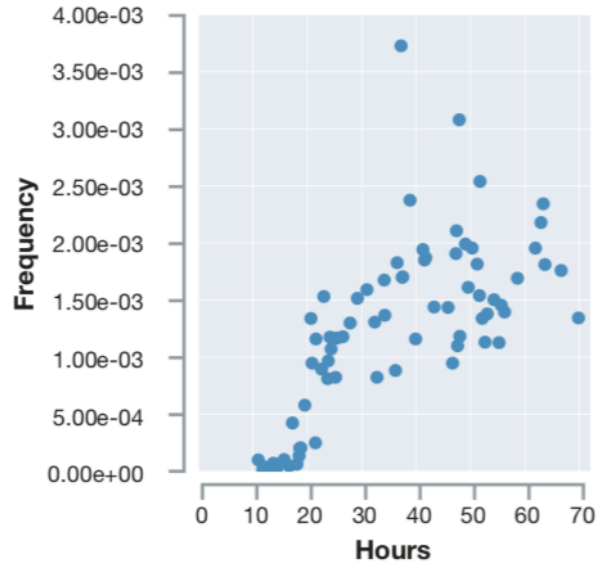




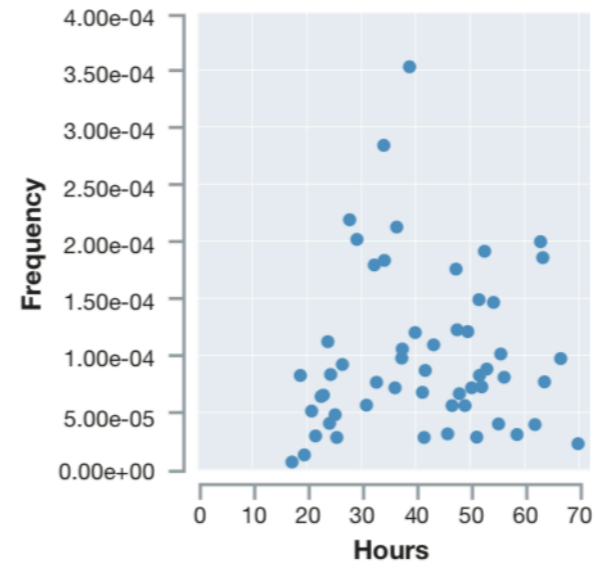
27 at 17 30 at 20 37 at 21 42 at 34 55 at 44 61 at 39
 64 at 40 67 at 39 70 at 40 71 at 25 73 at 21 74 at 60
 75 at 39 76 at 34

68 at 61

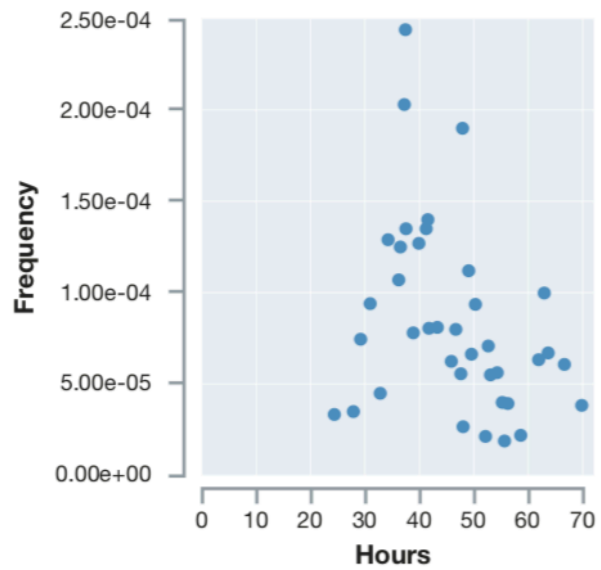
5-3 point press



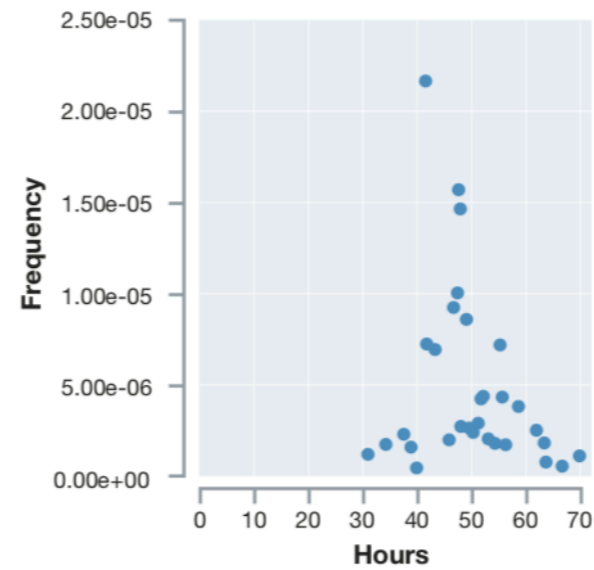
Small avalanche



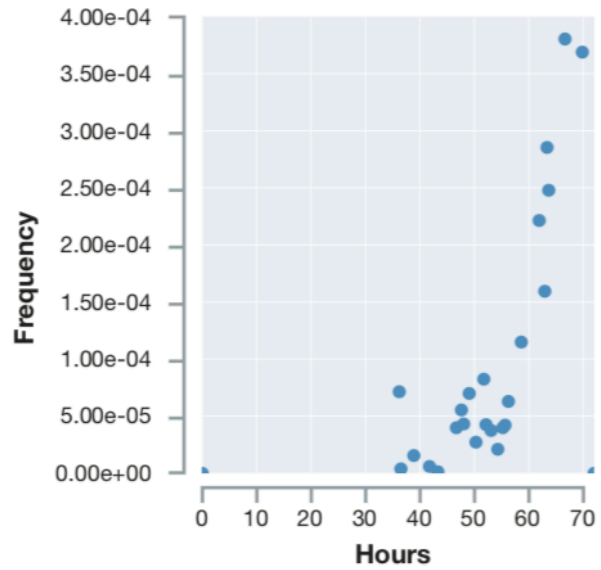
Attach and draw back



Knight's move pincer

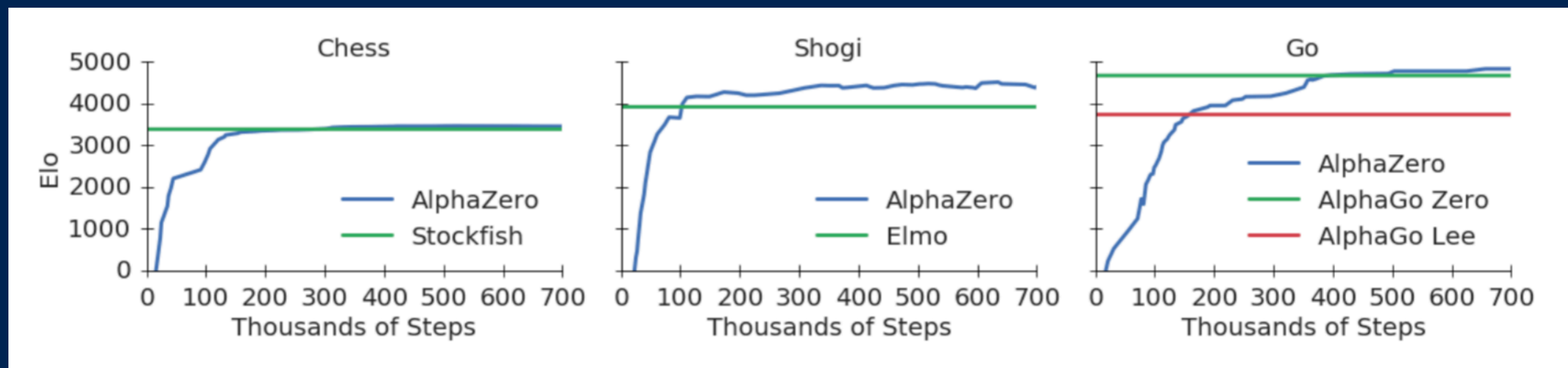


Pincer 3-3 point

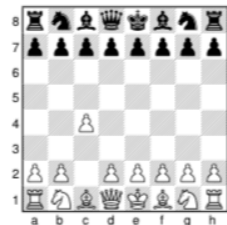


alpha go zero (dec 17)

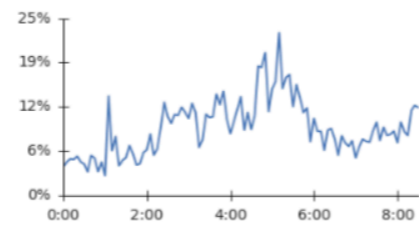
- generalized version of alpha go approach (no go-specific knowledge)
- input board state, possible moves, evaluation function
—> generates policy/value networks via self play
- teaches self how to play, improves to master level



A10: English Opening

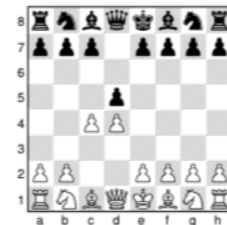


w 20/30/0, b 8/40/2

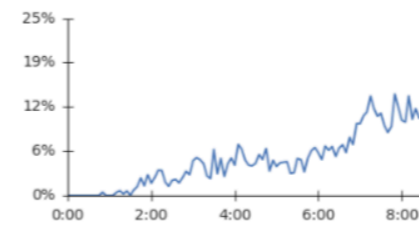


1...e5 g3 d5 cxd5 f6 g2 fxd5 f3

D06: Queens Gambit



w 16/34/0, b 1/47/2

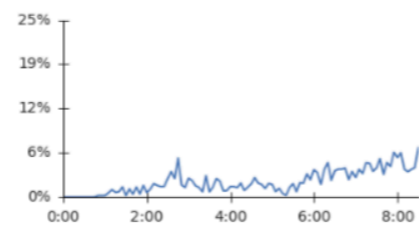


2...c6 f3 c3 f6 f3 a6 g3 c4 a4

A46: Queens Pawn Game

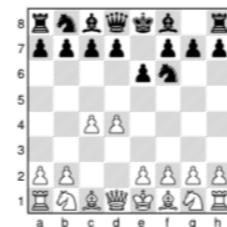


w 24/26/0, b 3/47/0

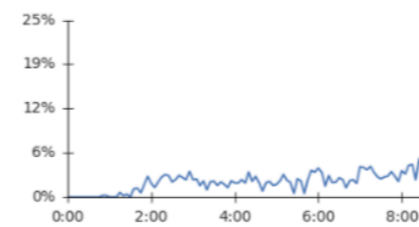


2...d5 c4 e6 f3 e7 f4 O-O e3

E00: Queens Pawn Game

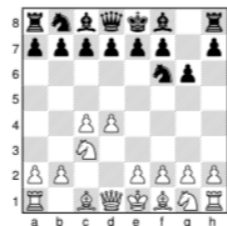


w 17/33/0, b 5/44/1

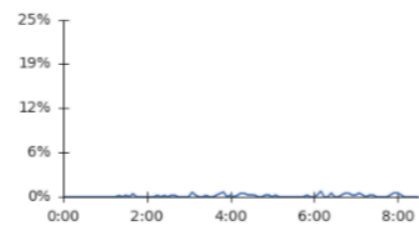


3.f3 d5 f3 c3 b4 g5 h6 a4 c6

E61: Kings Indian Defence

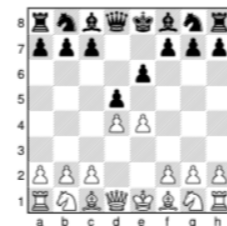


w 16/34/0, b 0/48/2

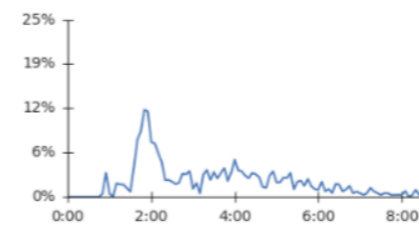


3...d5 cxd5 fxd5 e4 fxc3 bxc3 g7 e3

C00: French Defence



w 39/11/0, b 4/46/0

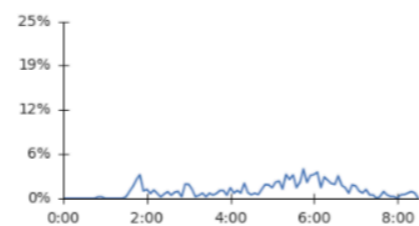


3.f3 f6 e5 d7 f4 c5 f3 e7

B50: Sicilian Defence



w 17/32/1, b 4/43/3

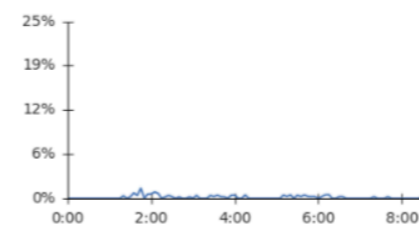


3.d4 cxd4 fxd4 f6 c3 a6 f3 e5

B30: Sicilian Defence

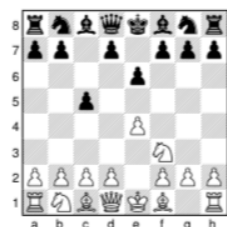


w 11/39/0, b 3/46/1

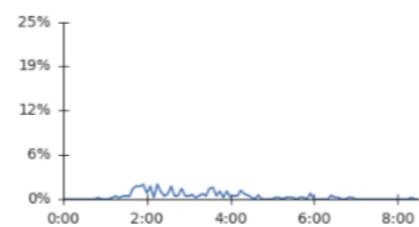


3.b5 e6 O-O e7 e1 a6 f1 d5

B40: Sicilian Defence



w 17/31/2, b 3/40/7

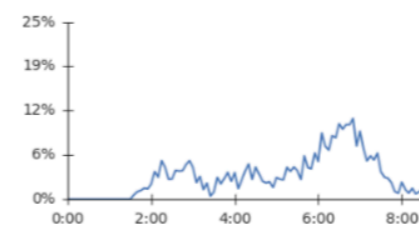


3.d4 cxd4 fxd4 c6 c3 c7 e3 a6

C60: Ruy Lopez (Spanish Opening)

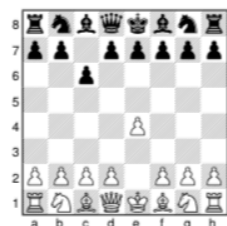


w 27/22/1, b 6/44/0

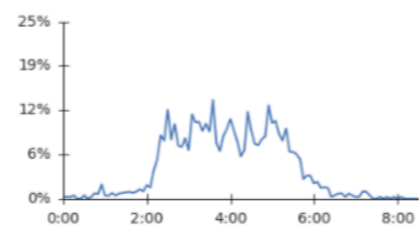


4.f4 e7 O-O f6 e1 b5 b3 O-O

B10: Caro-Kann Defence

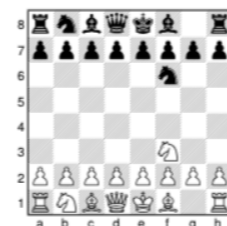


w 25/25/0, b 4/45/1

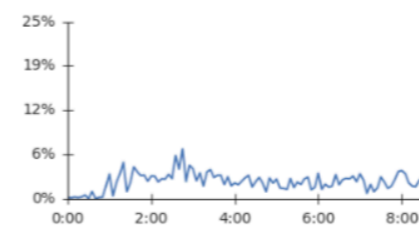


2.d4 d5 e5 f5 f3 e6 e2 a6

A05: Reti Opening



w 13/36/1, b 7/43/0



2.c4 e6 d4 d5 c3 e7 f4 O-O

Total games: w 242/353/5, b 48/533/19

Overall percentage: w 40.3/58.8/0.8, b 8.0/88.8/3.2

tic-tac-toe

- github.com/evg-tyurin/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)
- Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm (2017)

quarkworks.co

- **mobile apps for android and ios**
- **custom solutions for solving problems on-device or in the cloud**
- **keras, tensorflow lite, mobilenets, tf-coreml**
- **traditional machine learning, analytics**
- **slides: brettkoonce.com/work**