

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

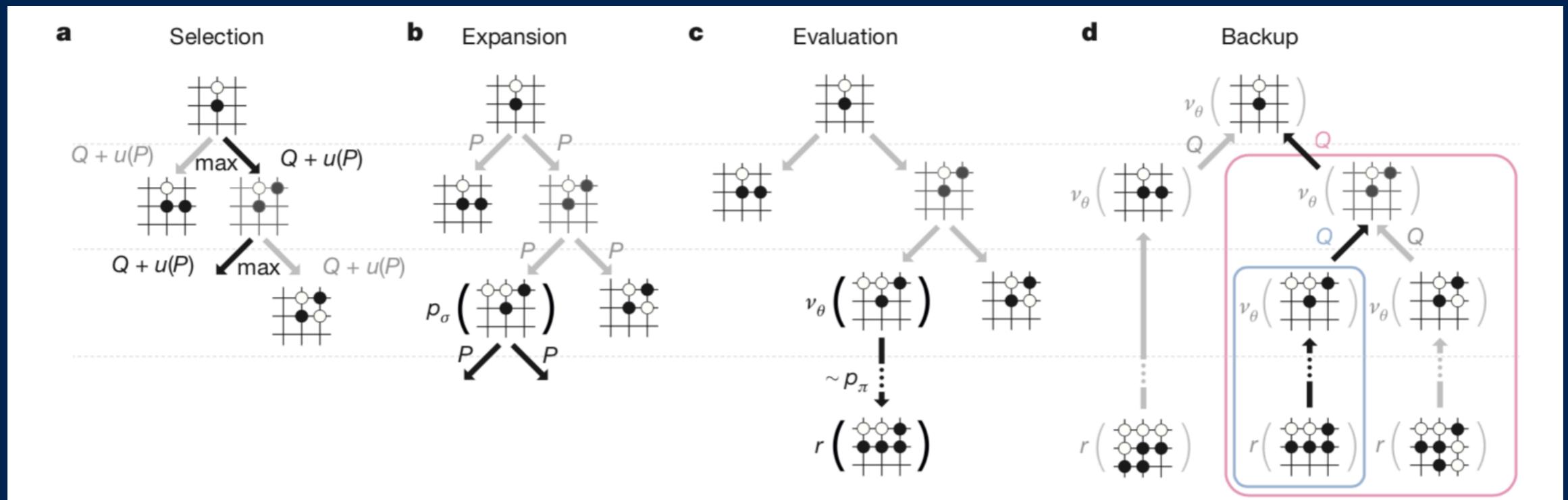


go: game/rules

- origin: asia, ~2500 years ago
- 19x19 board (361 squares), fill with stones
- squares + captures → 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 \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

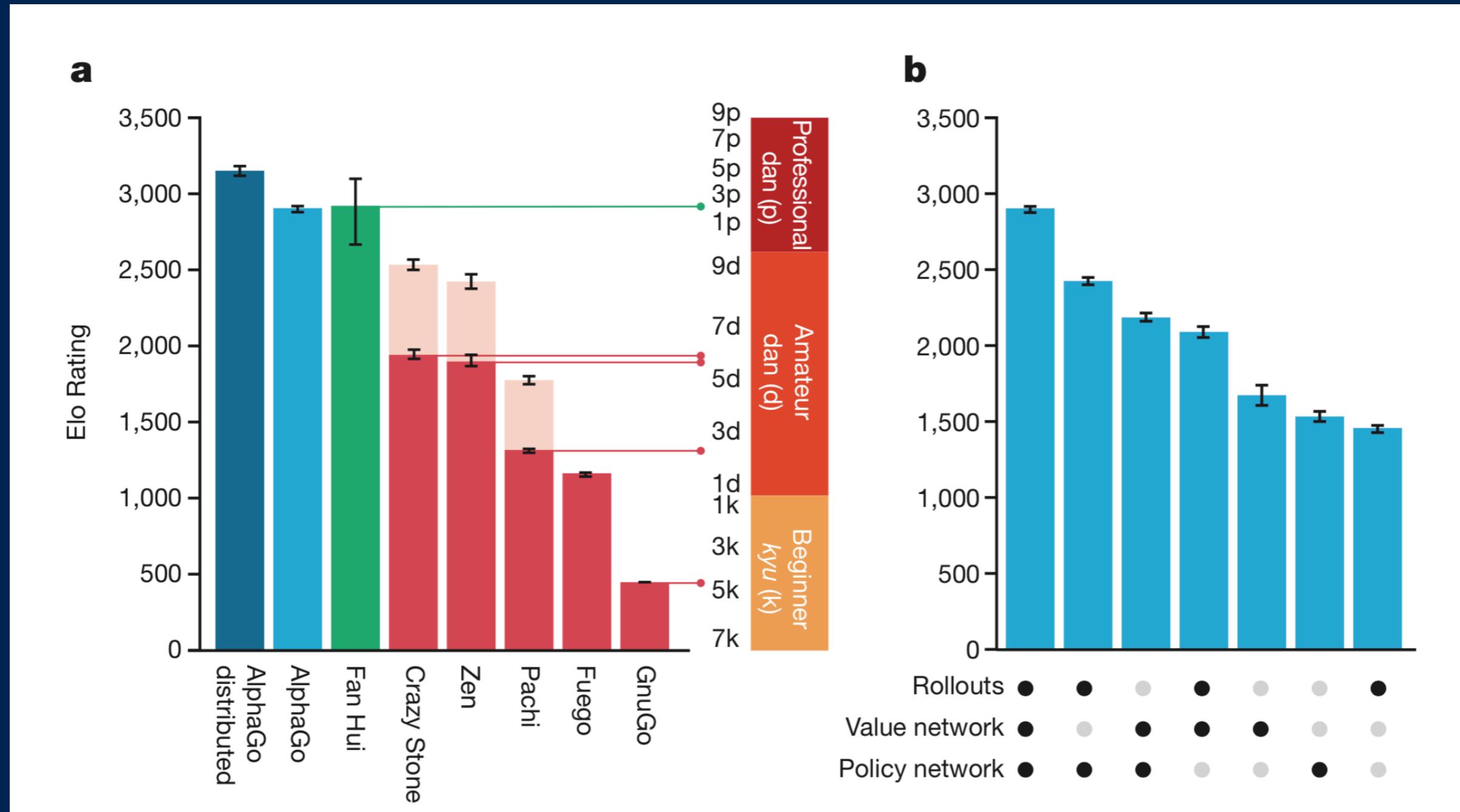
policy network

- **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](https://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

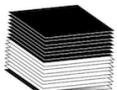
SELF PLAY

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

 π 

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

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

PREDICTIONS	P	Cross-entropy	π	ACTUAL
	V	+		
		Mean-squared error		
		+		
		Regularisation		

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'

Current position of
black's stones

...and for the previous
7 time periods

19 x 19 x 17 stack

1	1	1
1	0	0
0	0	1

Current position of
white's stones

...and for the previous
7 time periods

All 1 if black to play
All 0 if white to play

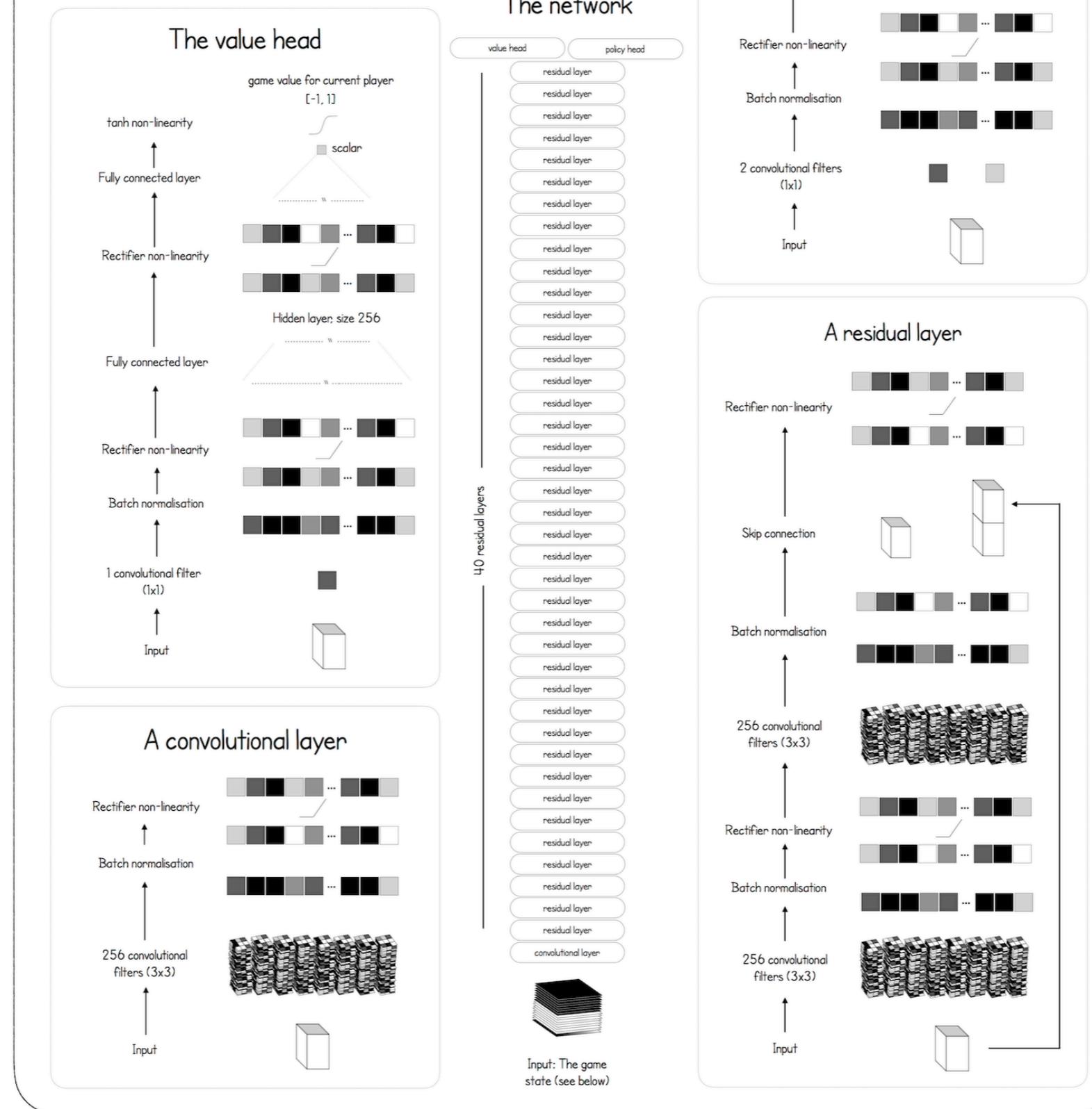
This stack is the input to the deep neural network

THE DEEP NEURAL NETWORK ARCHITECTURE

How AlphaGo Zero assesses new positions

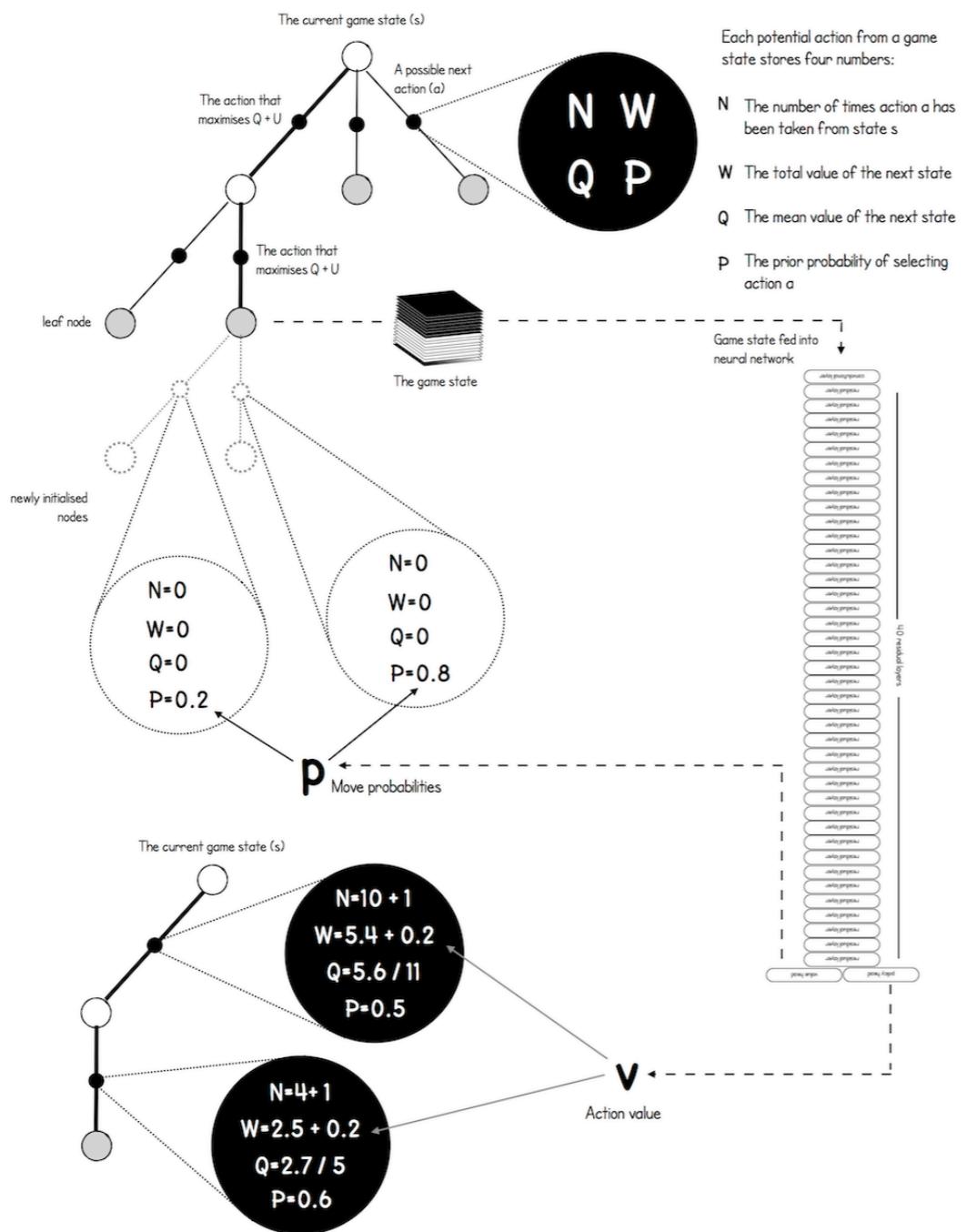
The network learns 'tabula rasa' (from a blank slate)

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



...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

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

where τ is a temperature parameter; controlling exploration

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...

$$Q + U$$

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

The mean value of the next state

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:

$$p \quad \text{Move probabilities}$$

$$v \quad \text{Value of the state (for the current player)}$$

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:

$$N \rightarrow N + 1$$

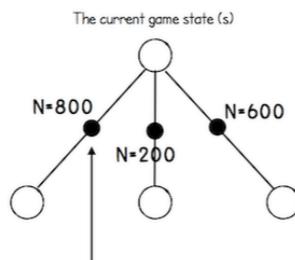
$$W \rightarrow W + v$$

$$Q = W / N$$

Other points

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

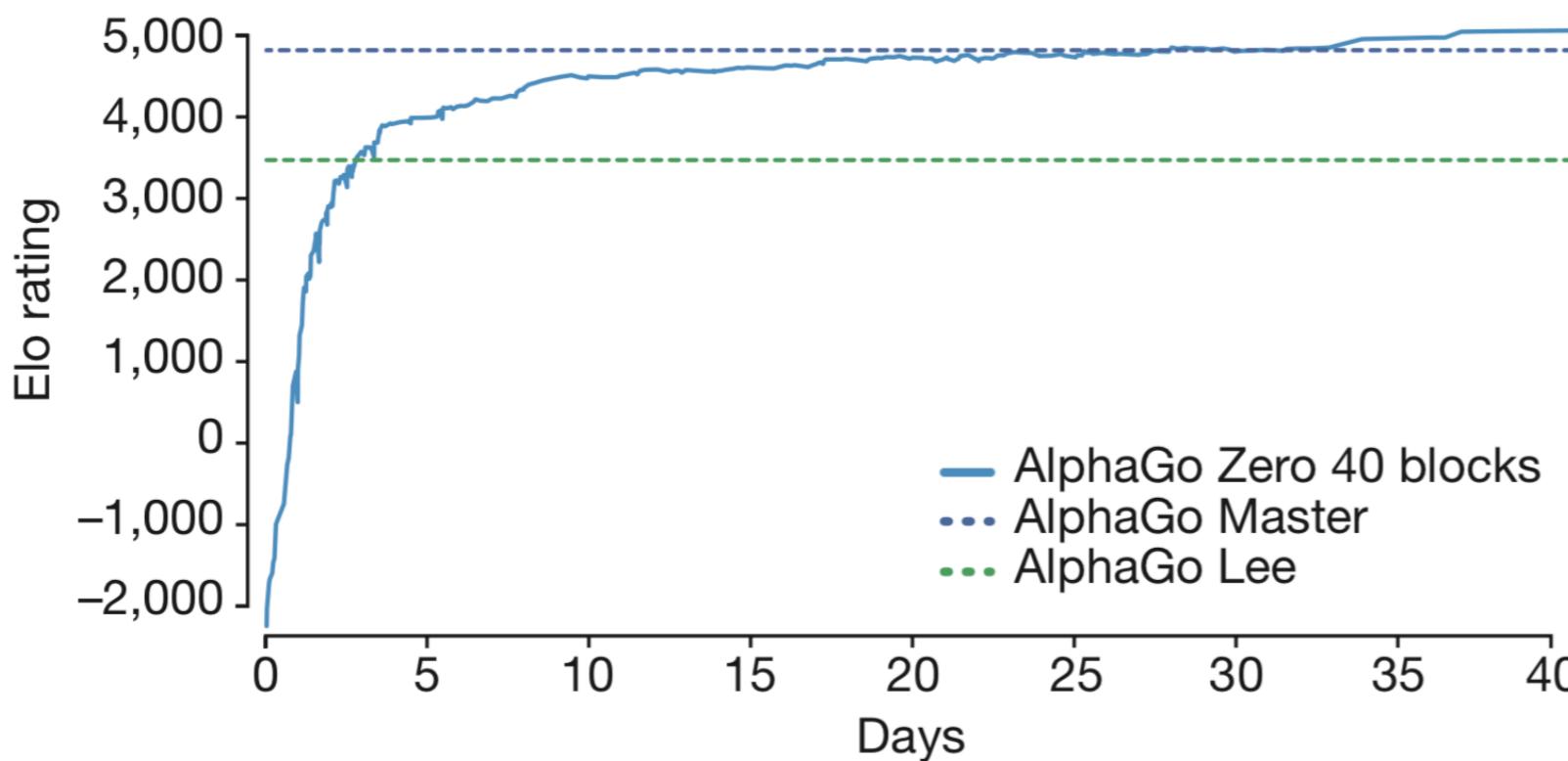
- The rest of the tree is discarded



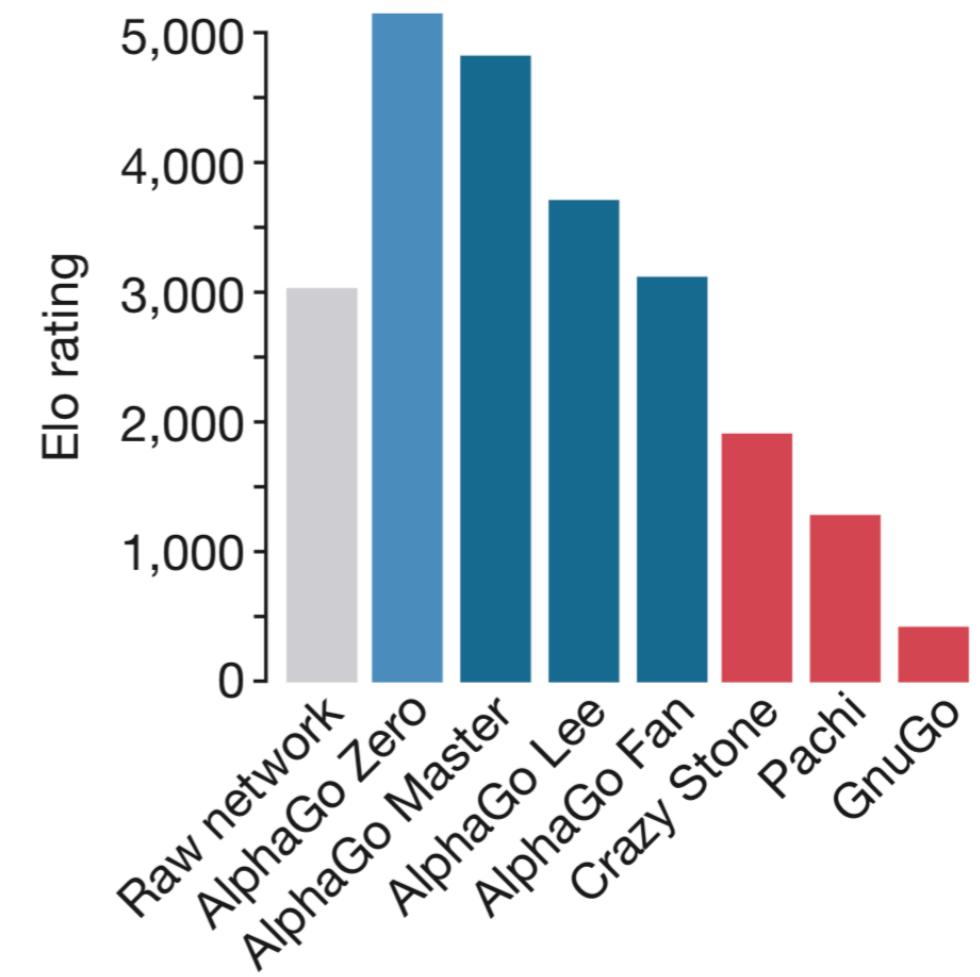
Choose this move if deterministic
If stochastic, sample from categorical distribution
 π with probabilities (0.5, 0.125, 0.375)

alpha go zero: train

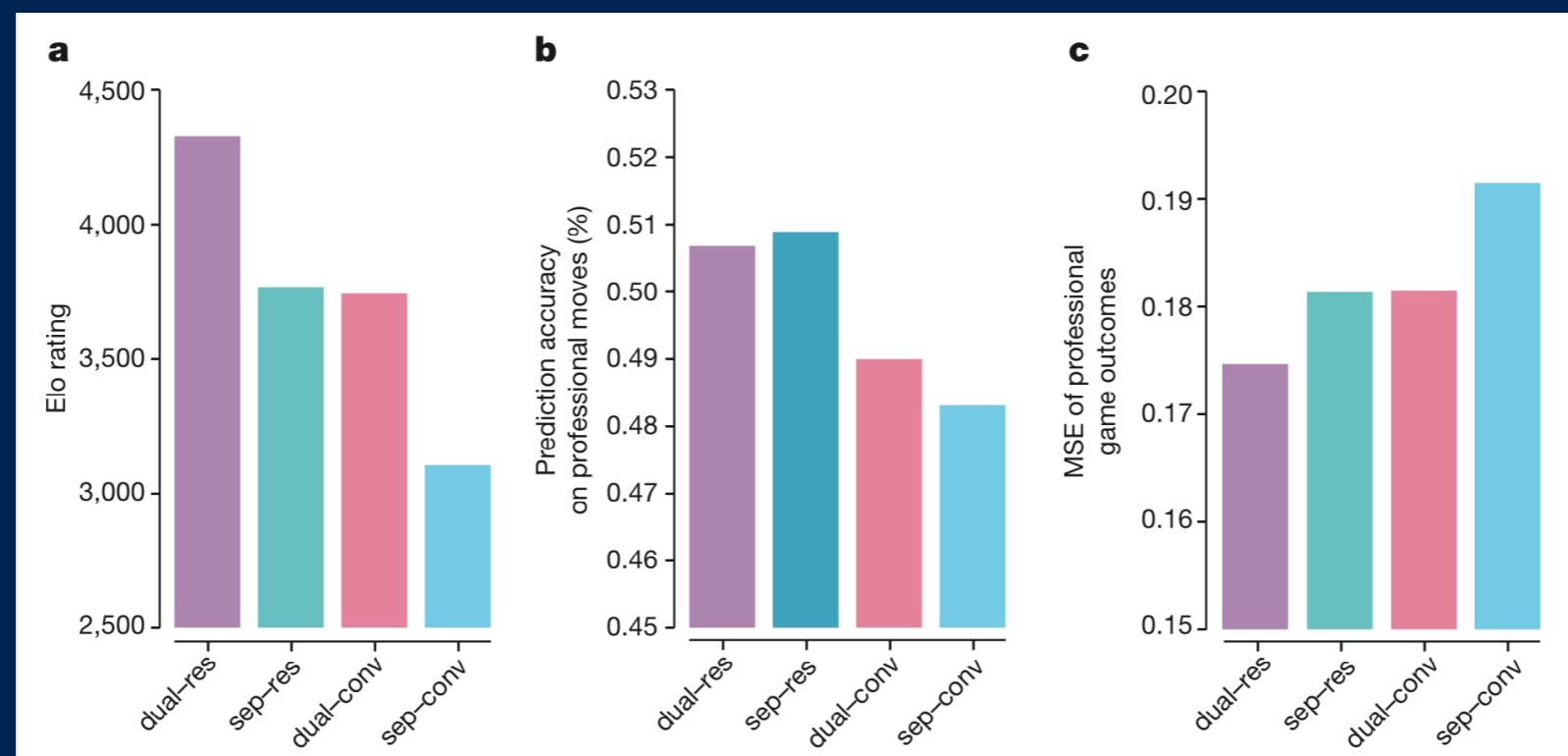
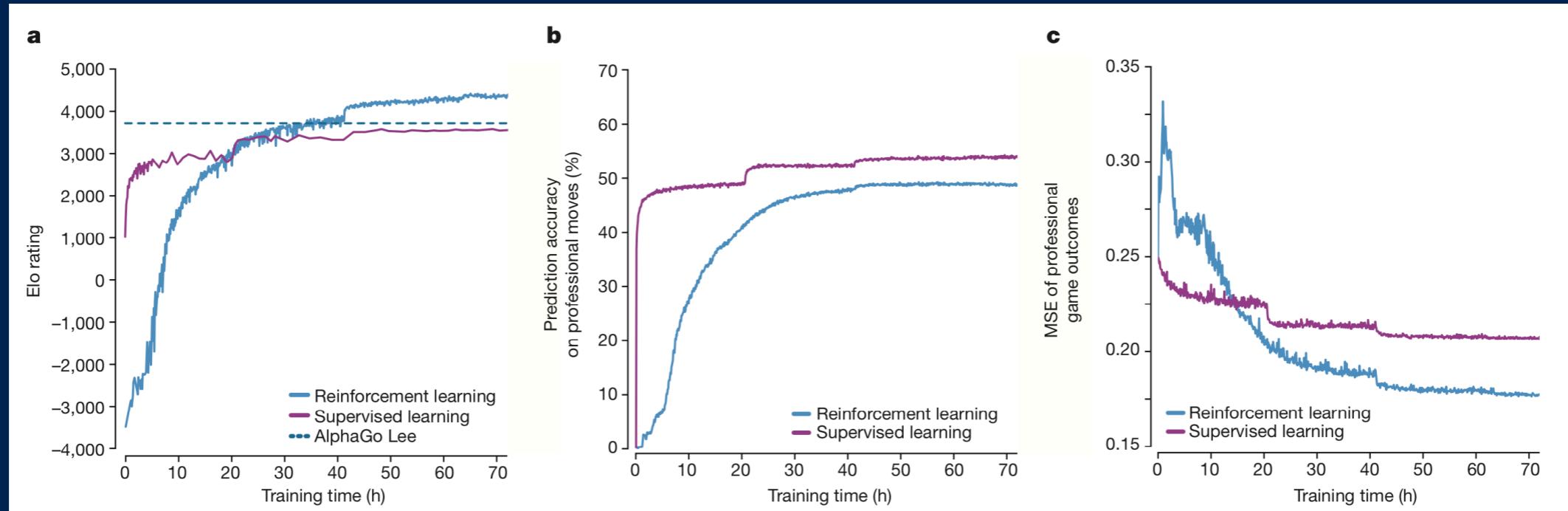
a

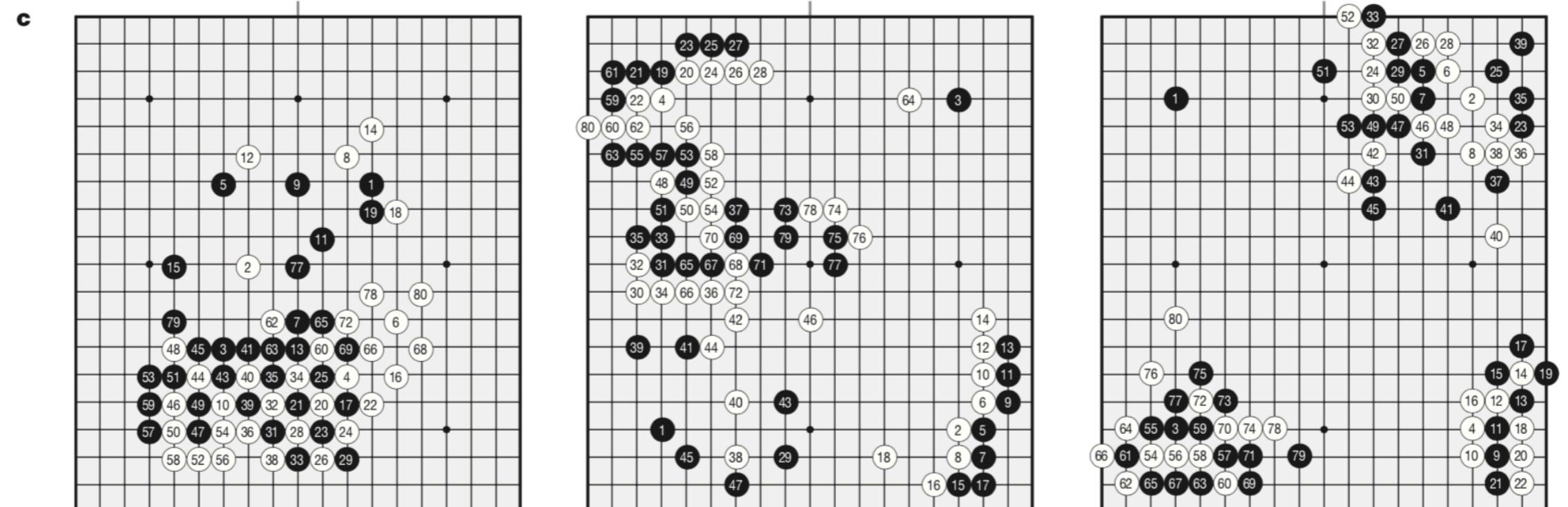
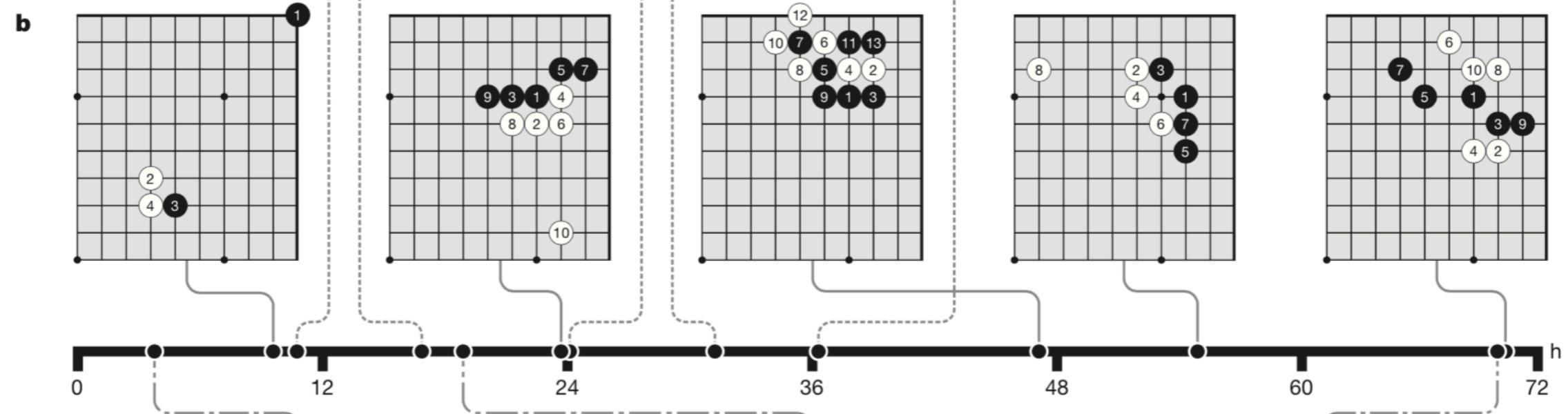
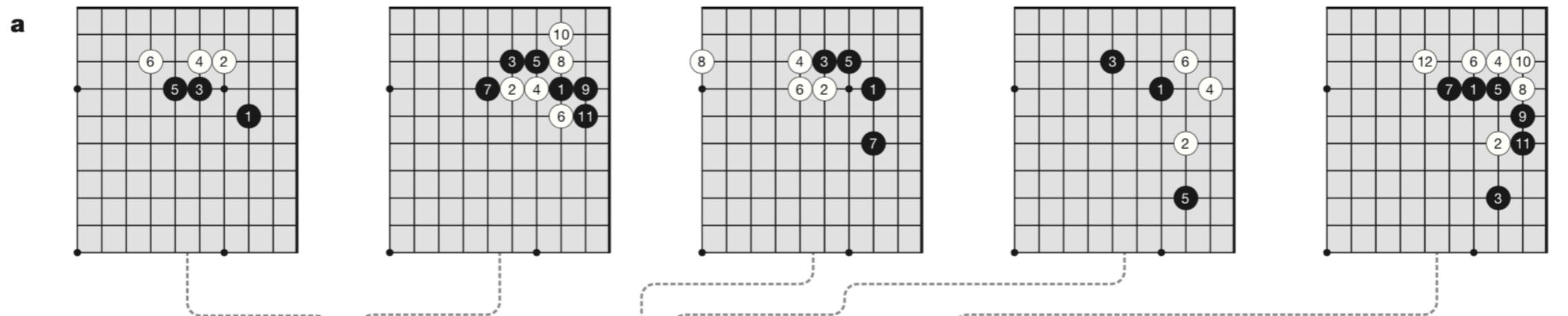


b



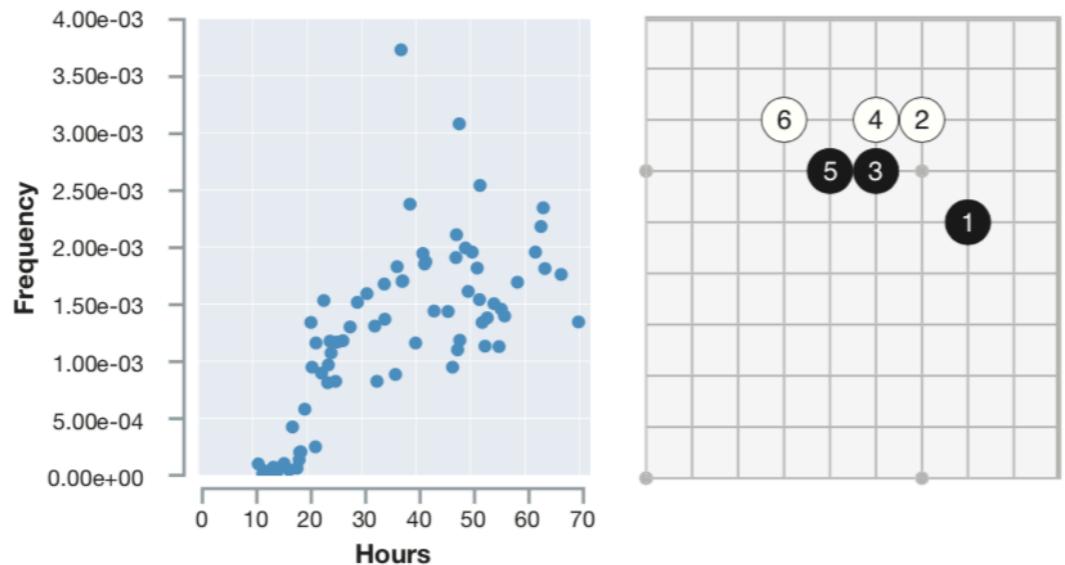
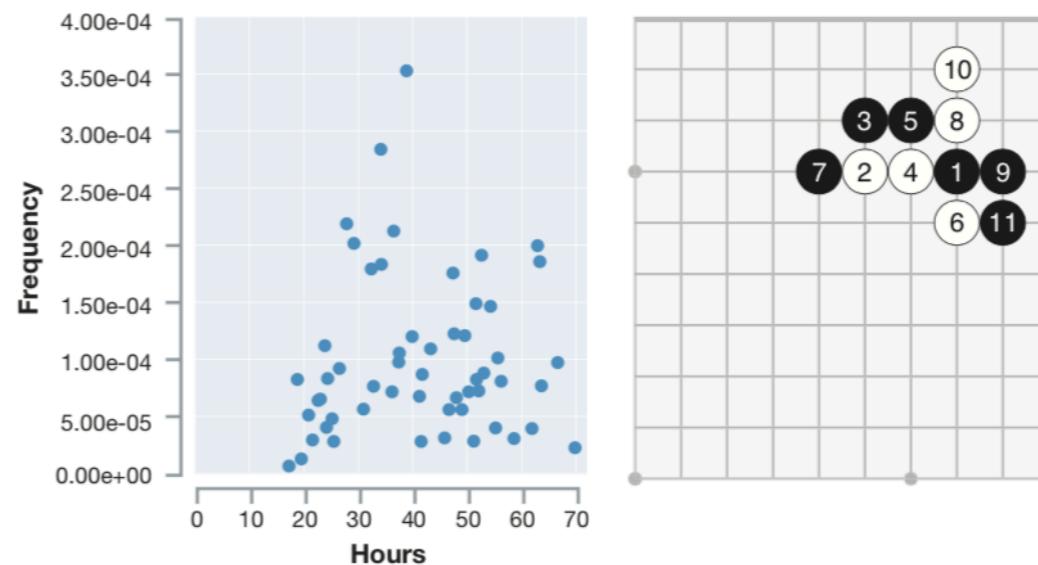
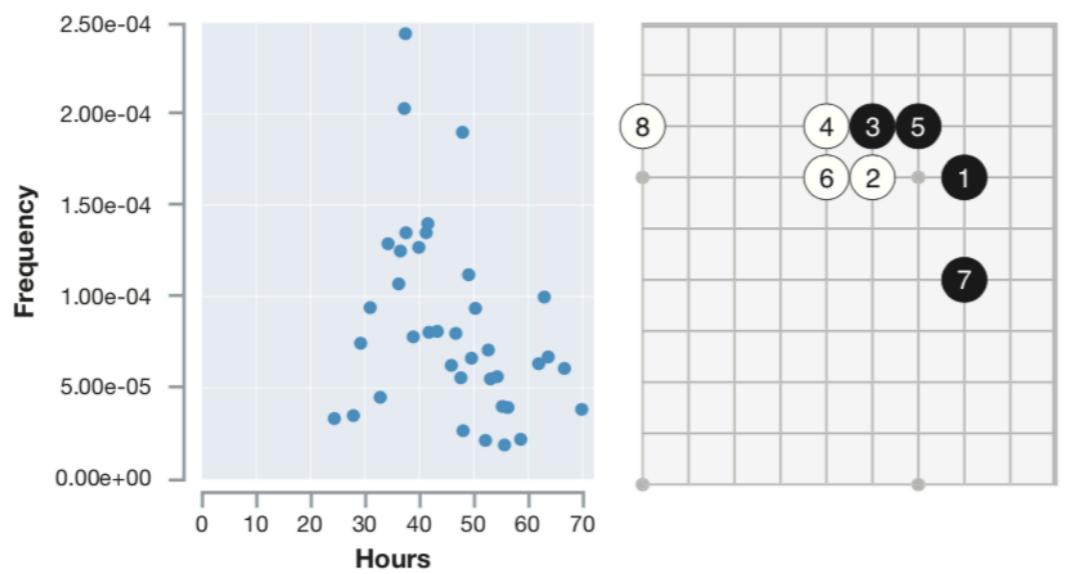
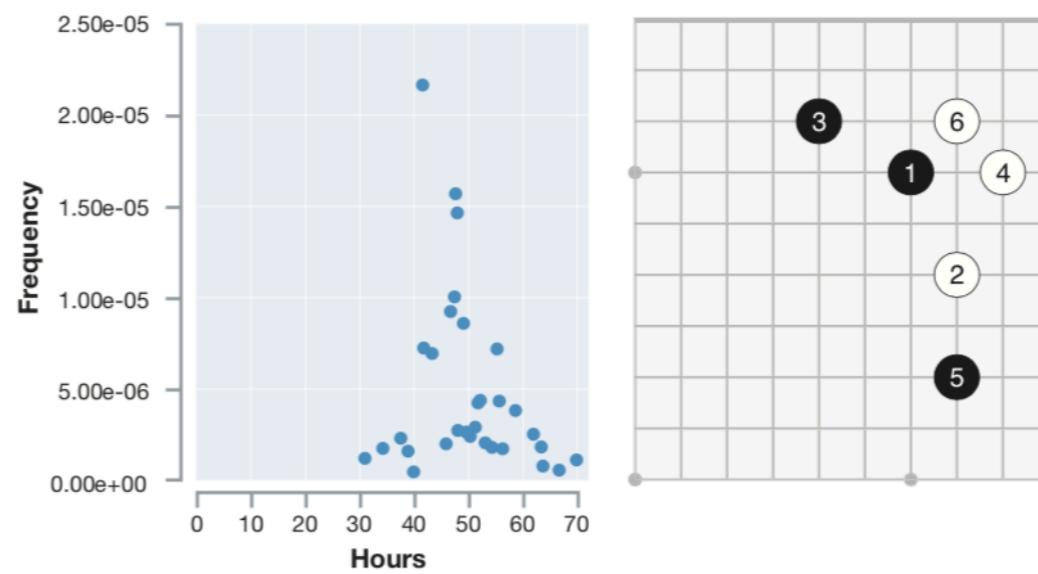
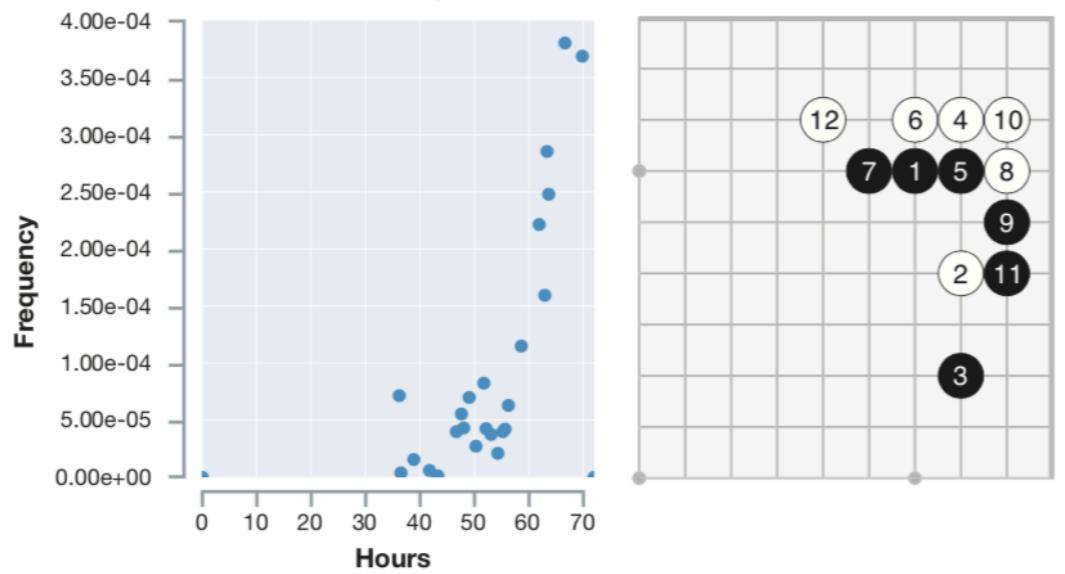
rl v. sl + resnet v. cnn





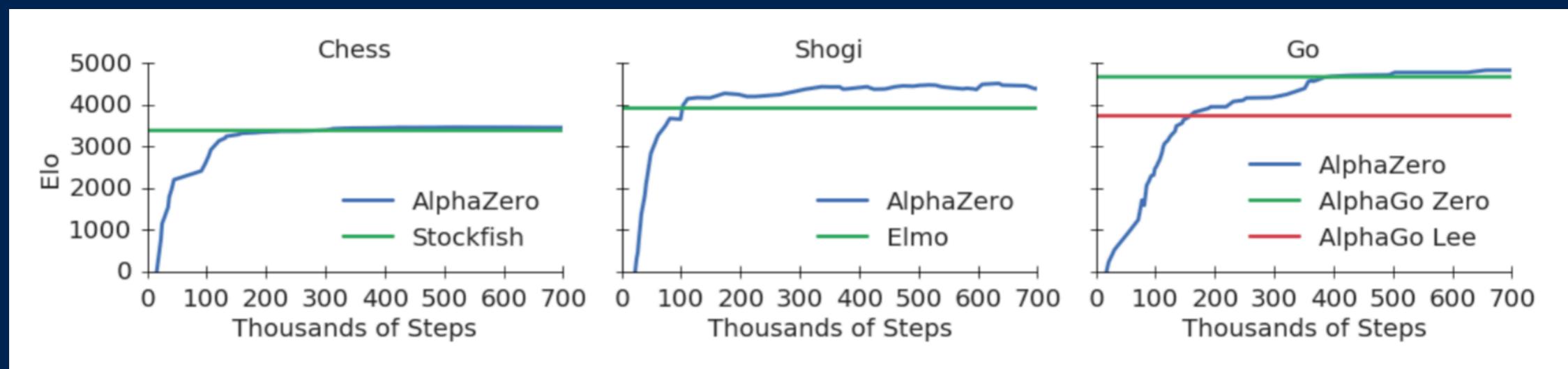
27 at 17 30 at 20 37 at 21 42 at 34 55 at 44 61 at 39
 64 at 40 67 at 39 70 at 40 71 at 25 73 at 21 74 at 60
 75 at 39 76 at 34

68 at 61

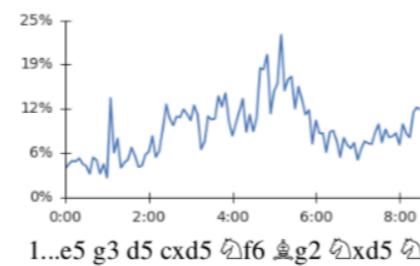
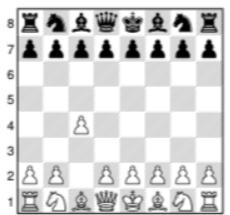
5-3 point press**Small avalanche****Attach and draw back****Knight's move pincer****Pincer 3-3 point**

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**

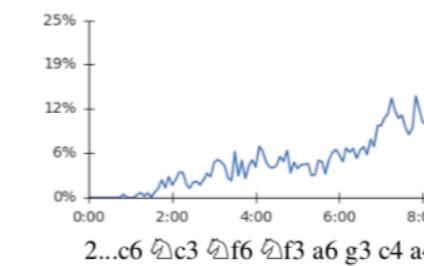


A10: English Opening



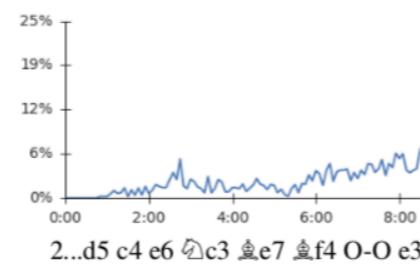
w 20/30/0, b 8/40/2

D06: Queens Gambit



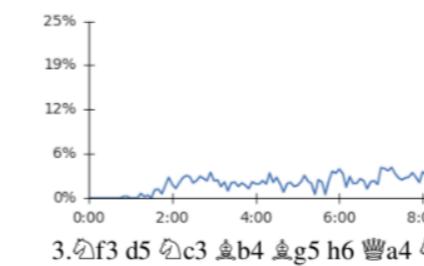
w 16/34/0, b 1/47/2

A46: Queens Pawn Game



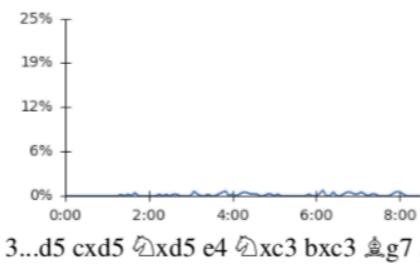
w 24/26/0, b 3/47/0

E00: Queens Pawn Game



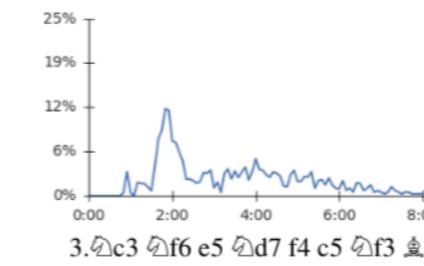
w 17/33/0, b 5/44/1

E61: Kings Indian Defence



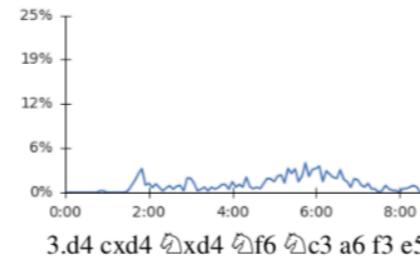
w 16/34/0, b 0/48/2

C00: French Defence



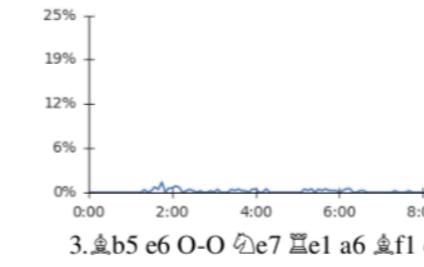
w 39/11/0, b 4/46/0

B50: Sicilian Defence



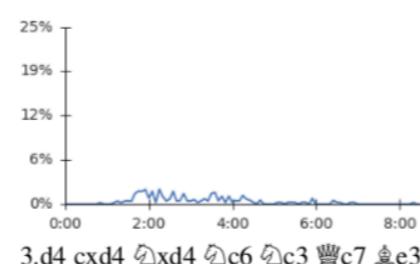
w 17/32/1, b 4/43/3

B30: Sicilian Defence



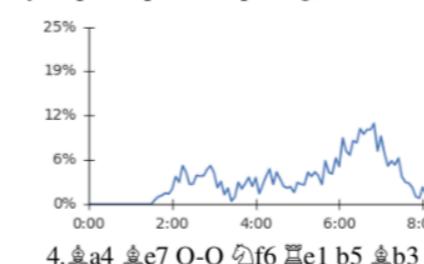
w 11/39/0, b 3/46/1

B40: Sicilian Defence



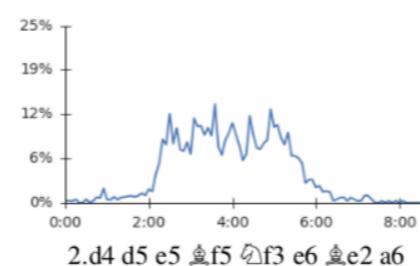
w 17/31/2, b 3/40/7

C60: Ruy Lopez (Spanish Opening)



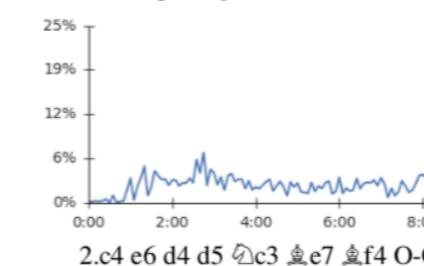
w 27/22/1, b 6/44/0

B10: Caro-Kann Defence



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

A05: Reti Opening



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

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