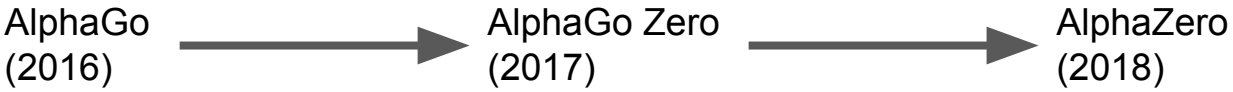
Exploring the Reinforcement Learning Techniques in AlphaGo

Ryan Chesler
San Diego Machine Learning

Problem statement

- Solve highly complex game of Go
- Impossible to exhaustively search because far too many possibilities to explore
- 19x19 board, Can represent game complexity as a search tree with breadth ~=250 and depth ~=150
- Need to find a way to simplify search procedure and determine what good moves are

Sequence of papers from DeepMind



Created very strong Go playing agent using custom procedures highly tuned to the specific game utilizing a Go database to bootstrap learning

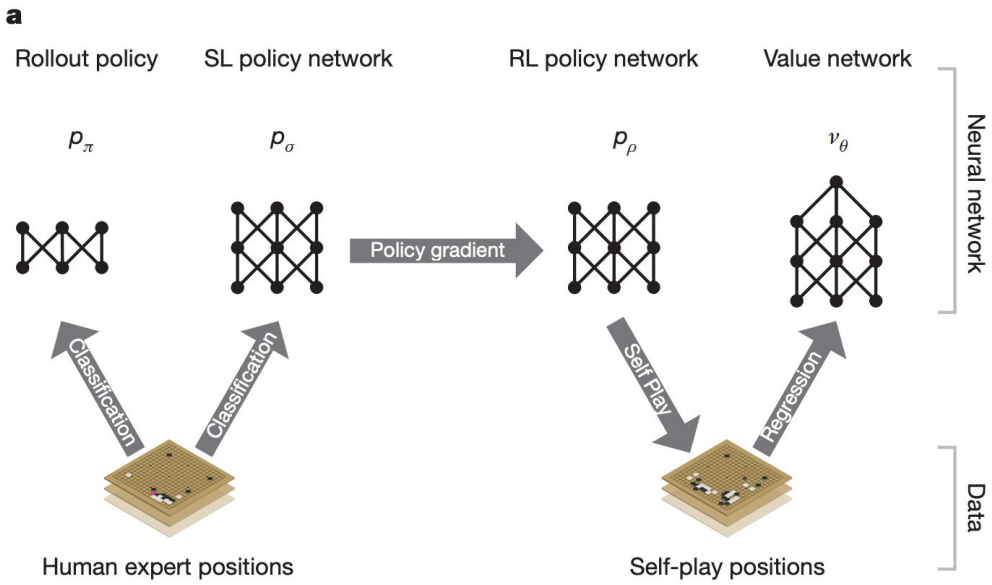
Stripped away a lot of the prior knowledge and made learning more efficient

Tabula rasa

Removed additional prior knowledge and generalized to additional games like chess and shogi

AlphaGo’s many networks

- Rollout policy network
- Supervised Learning policy Network
- Reinforcement Learning policy network
- Value network



Mastering the game of Go with deep neural networks and tree search

AlphaGo (Rollout Policy Network)

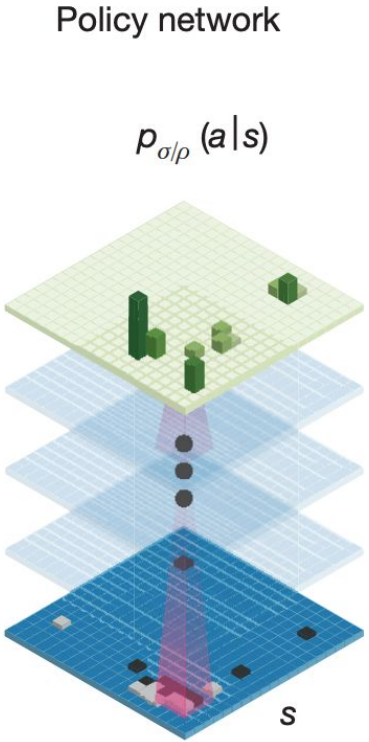
- Simple network that can very rapidly return decent move recommendations
- Trained to take in small pattern features derived from the board state and output probabilities of moves
- Trained against KGS Go database with 30 million positions from professional play
- Able to achieve 24.2% accuracy
- Decisions only take 2 microseconds

Extended Data Table 4 | Input features for rollout and tree policy

Feature	# of patterns	Description
Response	1	Whether move matches one or more response pattern features
Save atari	1	Move saves stone(s) from capture
Neighbour	8	Move is 8-connected to previous move
Nakade	8192	Move matches a *nakade* pattern at captured stone
Response pattern	32207	Move matches 12-point diamond pattern near previous move
Non-response pattern	69338	Move matches 3×3 pattern around move
Self-atari	1	Move allows stones to be captured
Last move distance	34	Manhattan distance to previous two moves
Non-response pattern	32207	Move matches 12-point diamond pattern centred around move

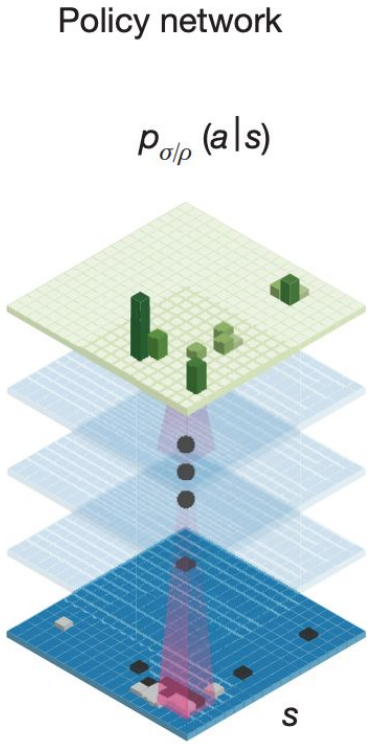
AlphaGo (Supervised Learning Policy Network)

- Similar objective to the rollout policy network
- Utilizes a simple ~15 layer convolutional neural network that takes in board features in the shape 19x19x48 and then outputs move probability for all 19x19 positions
- Takes 2 milliseconds to make predictions (nearly 1000x slower than rollout network)
- Able to achieve 57% accuracy with all features and 55.7% just using raw board position and move history



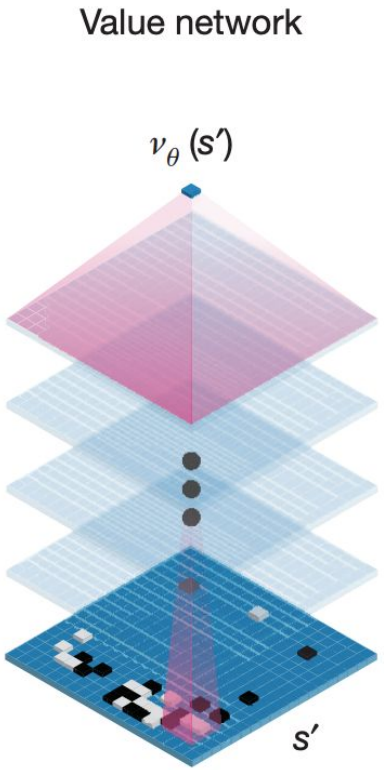
AlphaGo (Reinforcement Learning Policy Network)

- Initialized as a copy of the supervised learning policy network
- Utilizes self-play in order to continue improving win-rate
- Uses policy gradient, specifically REINFORCE
- Optimizes to win the game instead of just to imitate pro moves
- Achieves a win rate of 80% over the original supervised learning policy network



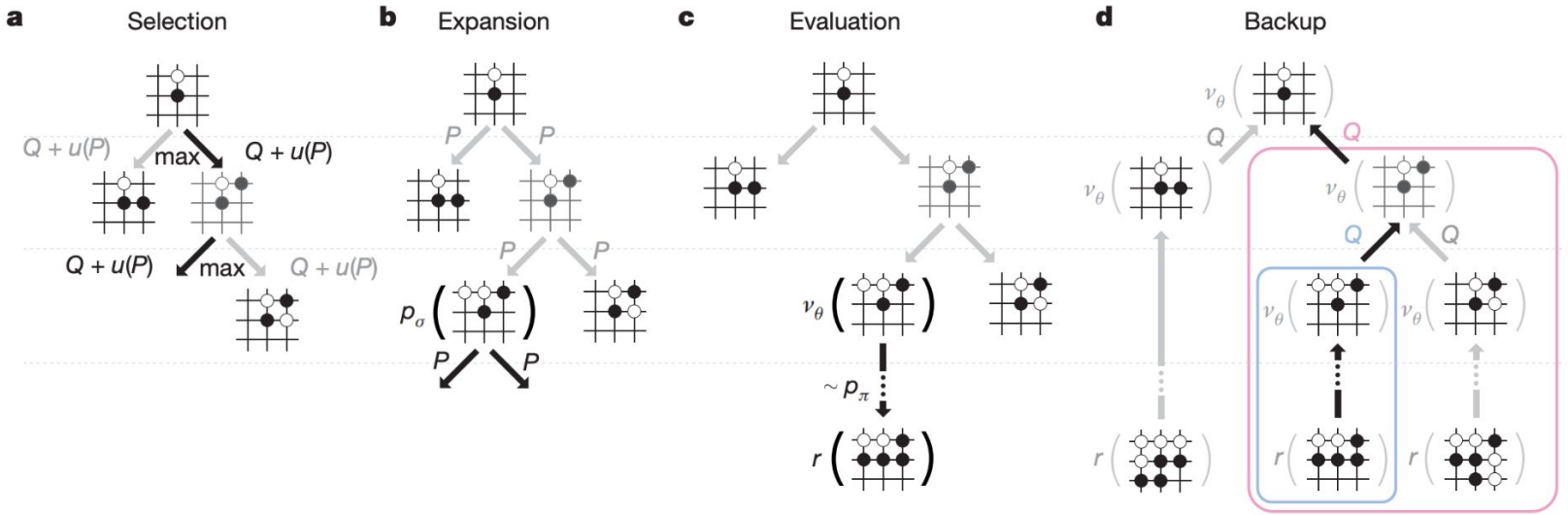
AlphaGo (Value Network)

- Similar to the supervised learning and reinforcement learning policy networks except not trying to predict positions, trying to evaluate the quality of the future position
- Same input representation and simply trying to predict win or loss from a given position
- Ran into a leakage issue when training on historical games because highly correlated. Needed to use individual positions generated from 30 million self-play games to prevent this
- Able to predict wins or losses with mean squared error of .234



AlphaGo (Monte Carlo Tree Search)

- Operates in 4 stages shown below



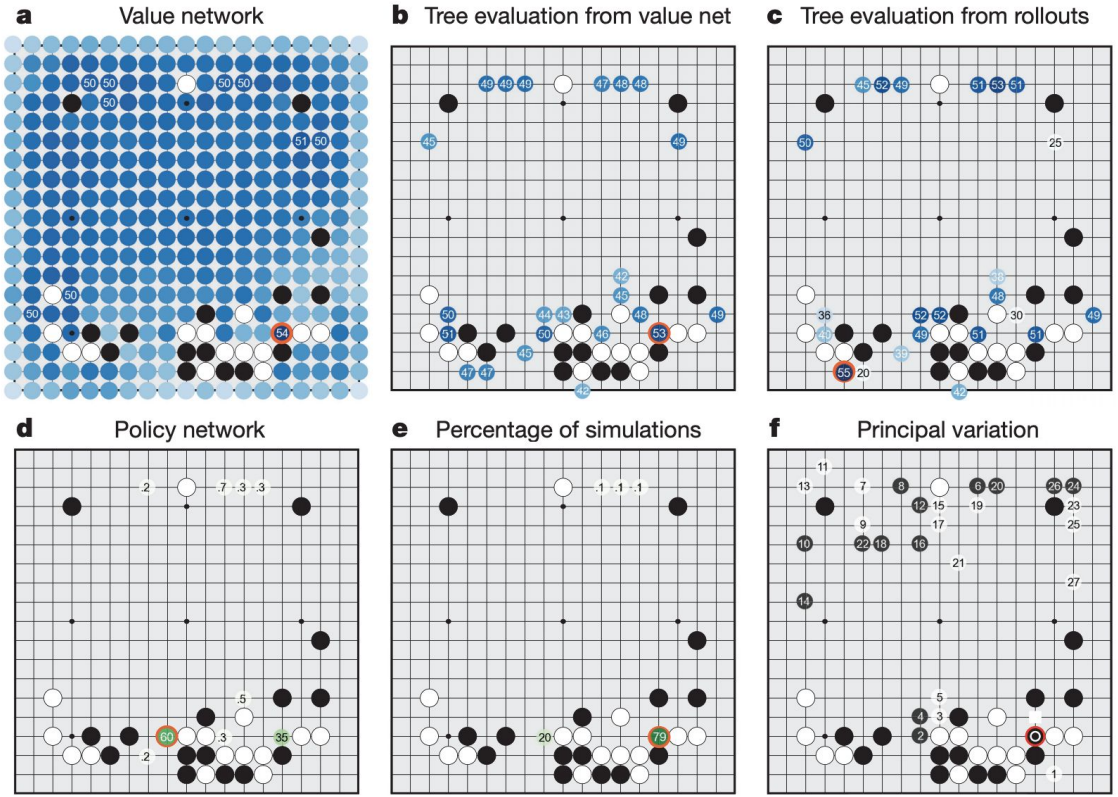
AlphaGo (Monte Carlo Tree Search cont.)

- Reinforcement learning policy network recommends probabilities to explore all moves, certain moves are more deeply explored

- Quality of move is evaluated by two separate mechanisms:
  - Value network gives an estimated win probability from a given position
  - Ultra-fast policy rollout network samples games to termination
- Tree is explored using upper confidence bound using the mixture of this evaluation and number of times each branch is explored

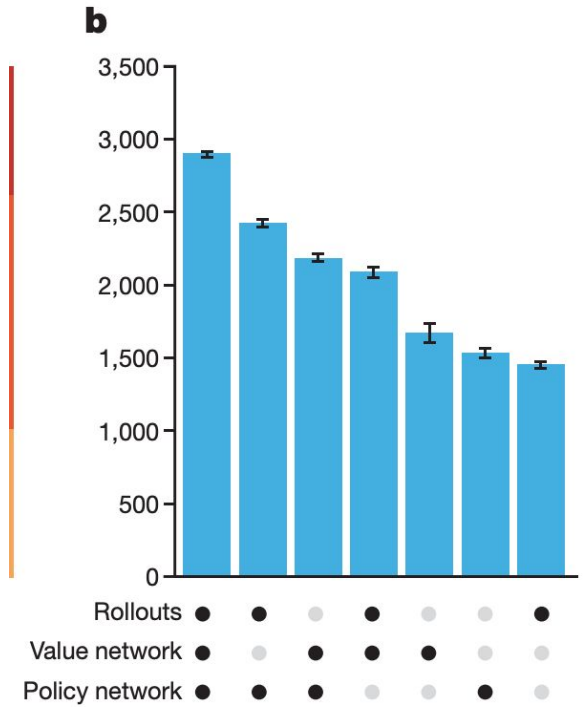
- Final move is decided by how many times a leaf is visited, not by value network or policy rollouts evaluation

AlphaGo (Move selection visualizations)



AlphaGo (evaluating strength of individual networks)

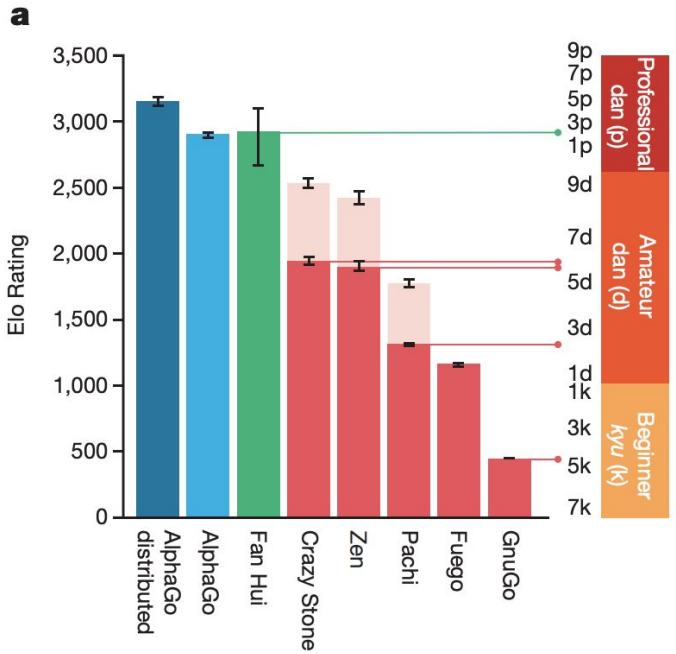
- Able to evaluate only using pieces of this ensemble of networks
- Each contributes in making the best decisions efficiently



AlphaGo (Results)

- Using a distributed version of AlphaGo DeepMind used 1202 CPUs and 176 GPUs to beat professional players and all other Go engines using 5s per move

- Elo of somewhere around 3k

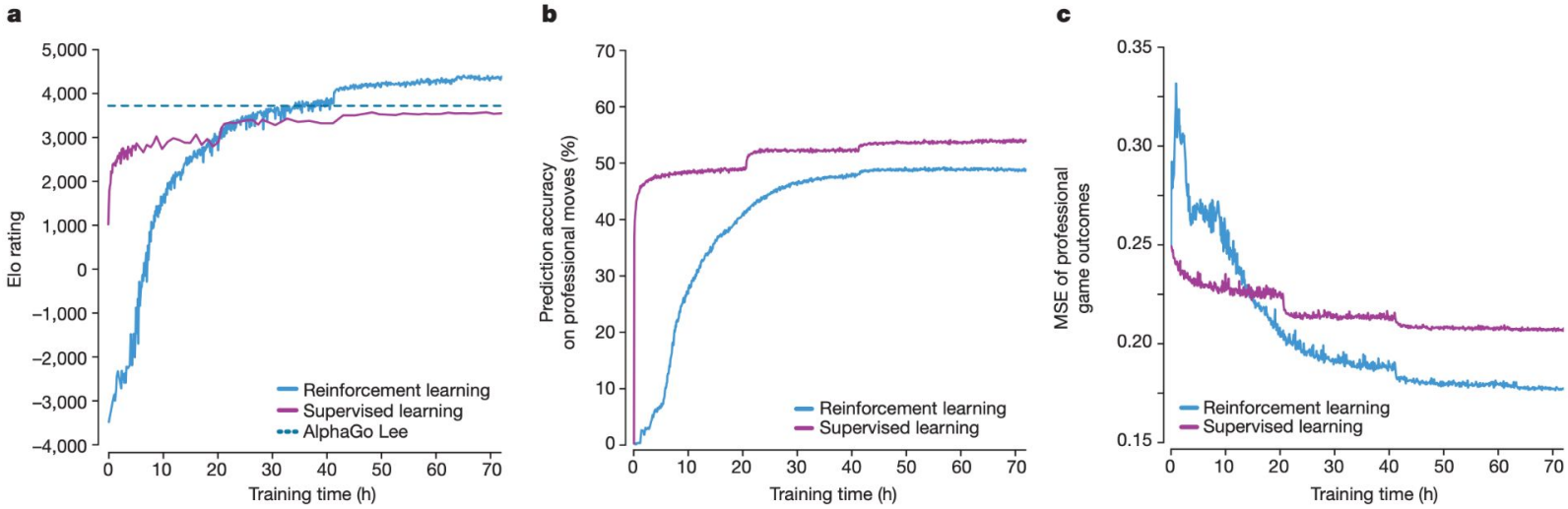


AlphaGo Zero Improvements

- Trained strictly from scratch, no KGS Go Database
- Upgraded networks to be deeper and utilize residual blocks
- Integrated some aspects of monte carlo tree search into the training procedure
- Eliminates fast rollout network and combines value and policy networks
- Able to beat AlphaGo with 4TPUs when AlphaGo is given 176 GPUs or 48 TPUs

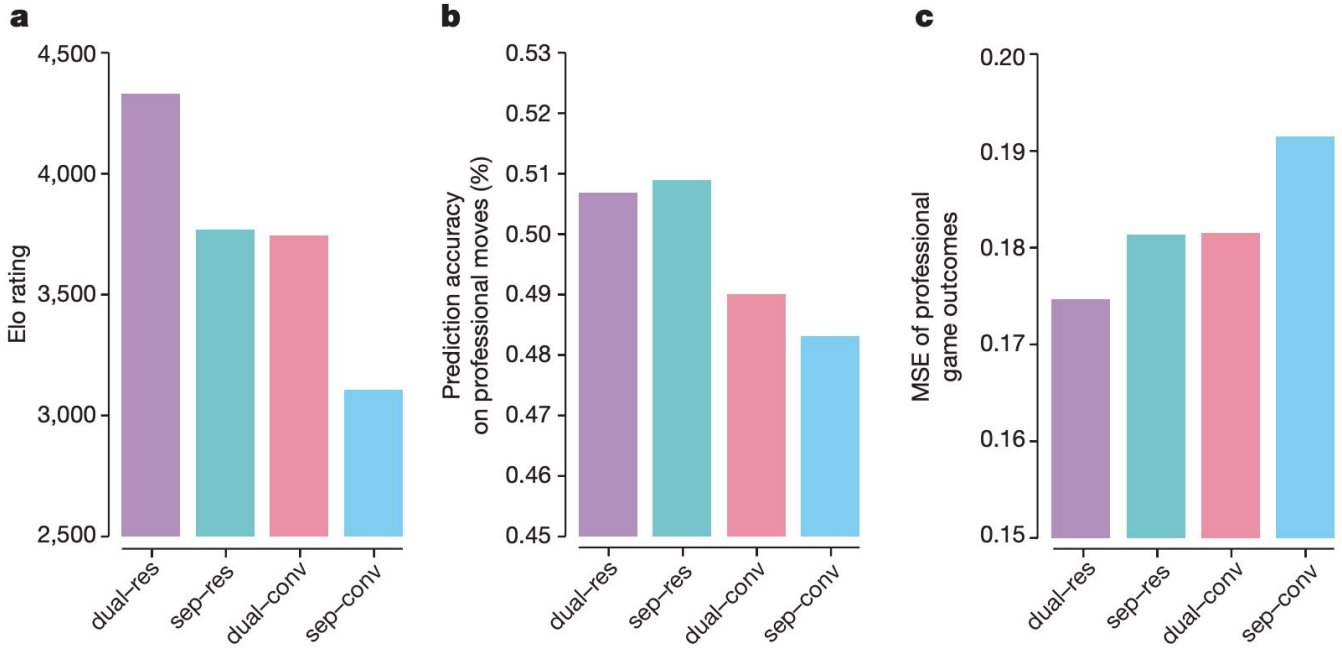
Mastering the game of Go without human knowledge

AlphaGo Zero (Learning Curves)

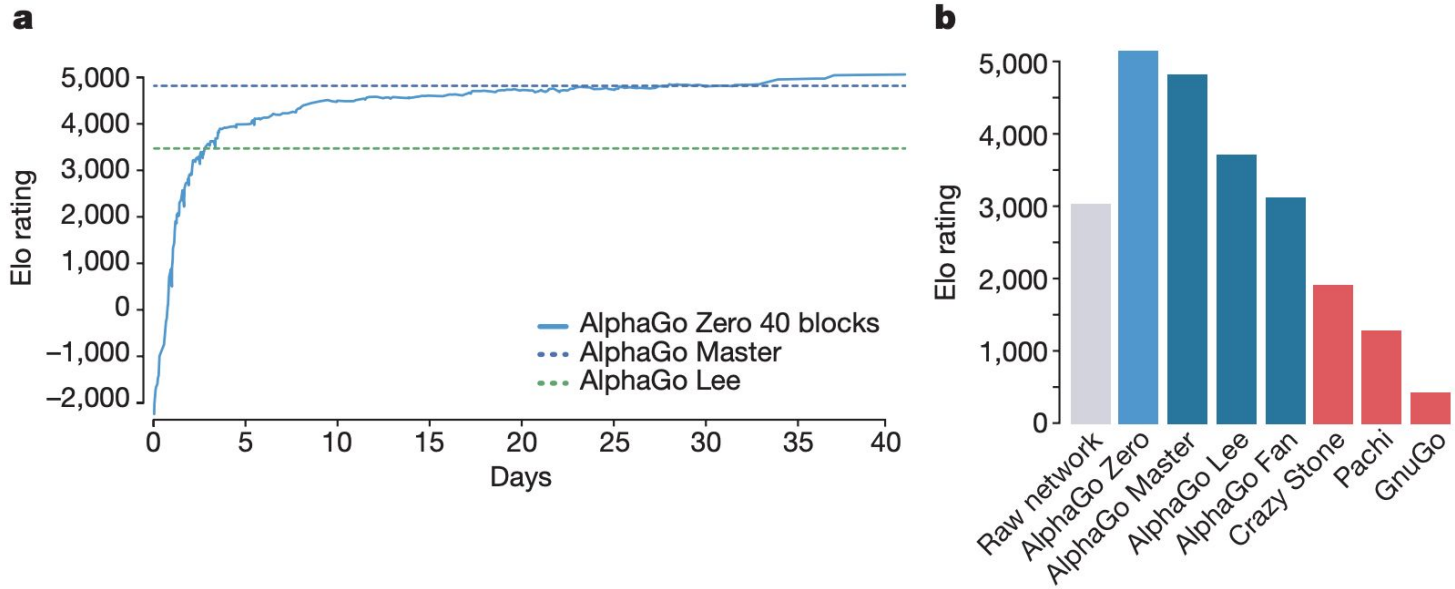


AlphaGo Zero (Architecture)

Combining networks and using residual blocks seems to sacrifice some move prediction accuracy but greatly improve value estimation and Elo



AlphaGo Zero (Elo improvement over time)



AlphaZero

- Remove all prior knowledge and generalize procedure to chess and shogi
- Remove any handcrafted features specific to the game, no augmentation tricks to take advantage of board symmetry
- Slightly changes the reward to allow for draws and optimizes for expected outcome
- No hyperparameter optimization

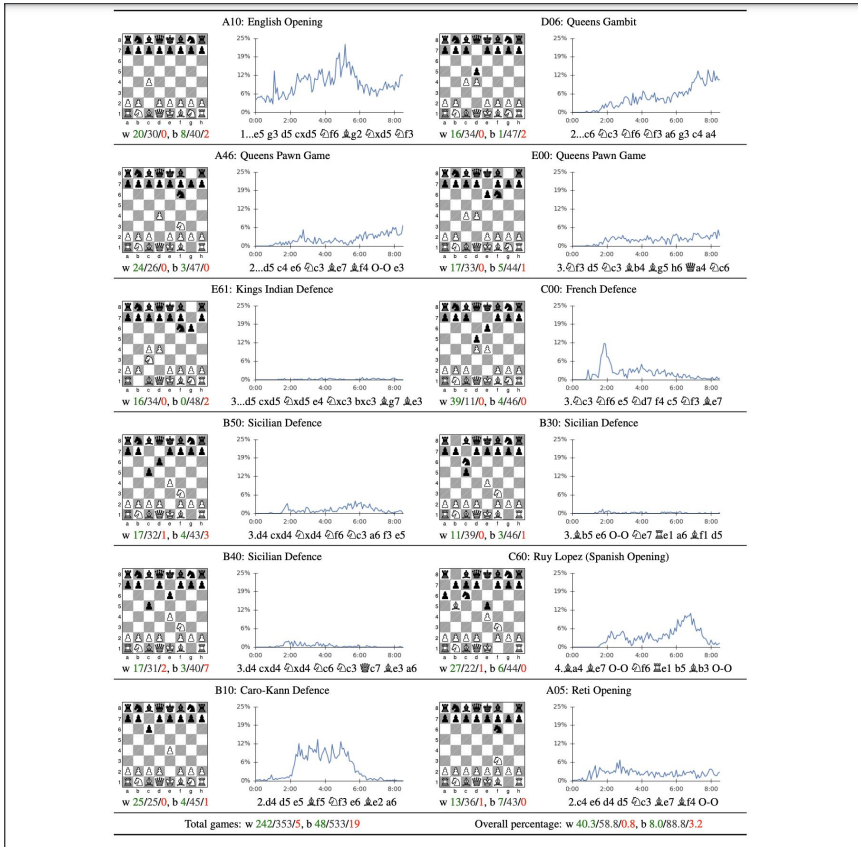
Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

AlphaZero (Search efficiency)

- Compared against strong programs like Stockfish(chess) and Elmo(shogi) AlphaZero evaluates several orders of magnitude less positions per second, but evaluates much better positions and ends up making better decisions

- AlphaZero searches 80k positions per second in chess while Stockfish evaluates 70 million

AlphaZero (Learning book moves)



AlphaZero (Training procedure/Results)

- Able to achieve state of the art across all games in “3” days

- “Training proceeded for 700,000 steps (mini-batches of size 4,096) starting from randomly initialised parameters, using 5,000 first-generation TPUs (15) to generate self-play games and 64 second-generation TPUs to train the neural networks.”