AlphaGo

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

By Ilan Godik

Background: Go history

- Created 3,000 years ago
- Considered as poetry and art.
- Taught at schools in Korea & Japan.

Background: Go in AI

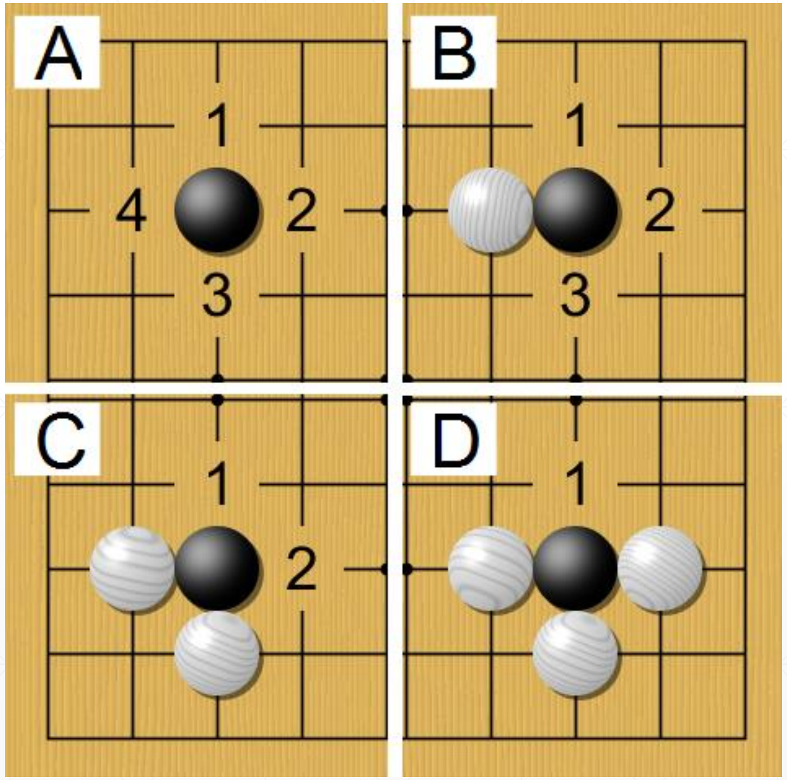
- Huge branching factor: ~200 vs. ~20 in chess.
- Huge state space: 10^{170} positions
- Hard to evaluate states
- Infamous in AI literature: Grand challenge,
  “at least 10 years until solved”
- Literature Pre-AlphaGo & Post-AlphaGo

Rules of Go

- Black starts
- 19x19 board
- Each player places a stone in his turn

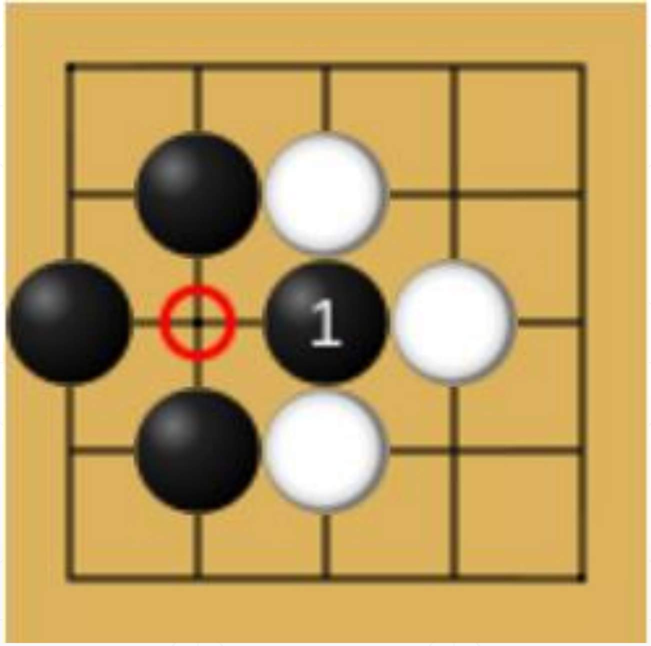
Go: Rule #1

- 0 Liberties => Stone removed



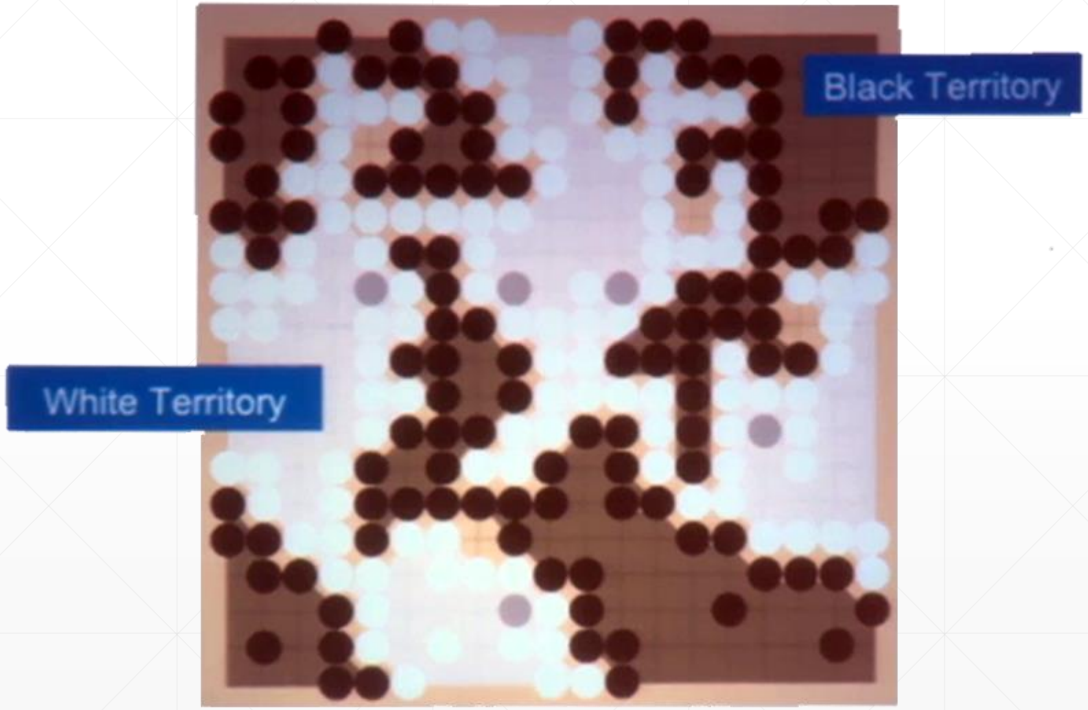
Go: Rule #2

- No repeating boards



Go: Winner

- Stones + Territory



Perfect information Games

- Optimal value function: $v^*(s)$
  - Represents the winner under perfect play
- Can be obtained by exhaustive minimax search.
  - Infeasible

Solution: Reduce Depth

1. Reduce depth:
   - Approximate $v^*(s)$ with $v(s)$
     - Up until now: heuristics
     - Success in Chess, Checkers & Reversi
     - Go is to complex for human-curated heuristics
   - Called “Value Function”

Solution: Reduce Breadth

2. Reduce breadth:

- Sample from a probability distribution
  - $P(a|s)$
- What nodes to explore
- Guiding rollouts

Solution: MCTS

3. MCTS: Monte Carlo Tree Search

- A solution to the multi-arm bandit problem
- Balance Exploration vs. Exploitation
- Average rollouts to the end of the game
  - Converges to $v^*(s)$
- Visited paths converge to optimal play

Previous attempts

- <u>Policies</u> & <u>Value functions</u> were <u>heuristic</u>
- <u>Linear combinations</u> of <u>hand-crafted features</u>

Deep Convolutional Neural Networks

- Huge success:
  - Image classification
  - Face recognition
  - Playing Atari games
- Many layers of neurons
- Overlapping patterns, convolutional layers
- Increasingly abstract & localized representation of the input

AlphaGo overview

1. Policy Networks
2. Value Network
3. MCTS
4. Distributed Computation & Time management

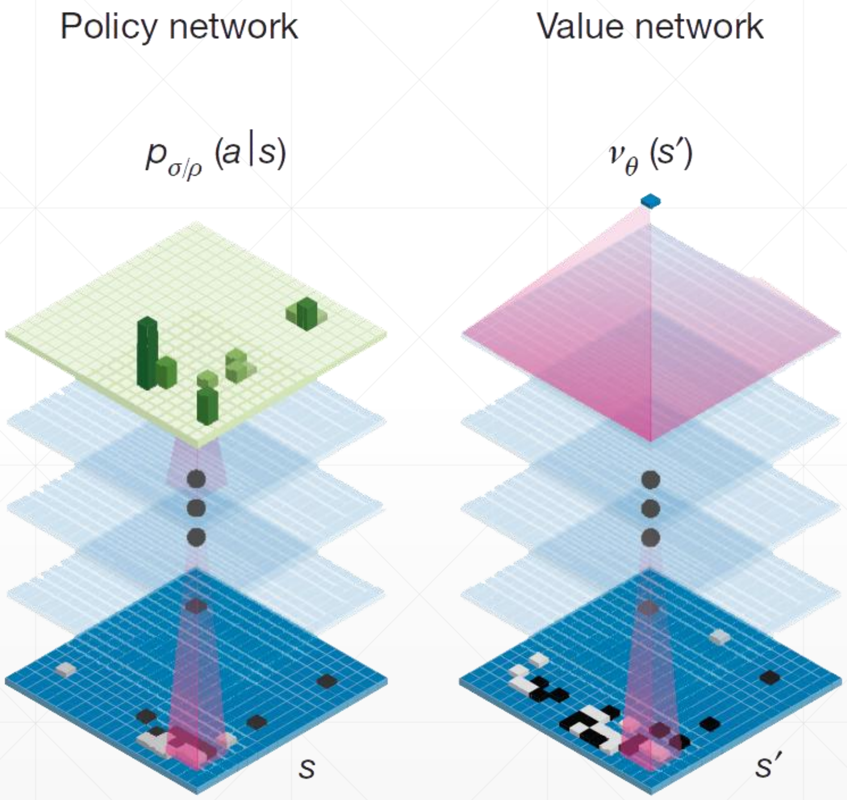
Policy Networks

1. Supervised Learning (SL) Policy Network P_σ
   - Trained by predicting expert games
2. Fast Policy Network P_π
   - For rollouts
3. Reinforcement Learning (RL) Policy Network P_ρ
   - Trained by playing against itself from P_σ

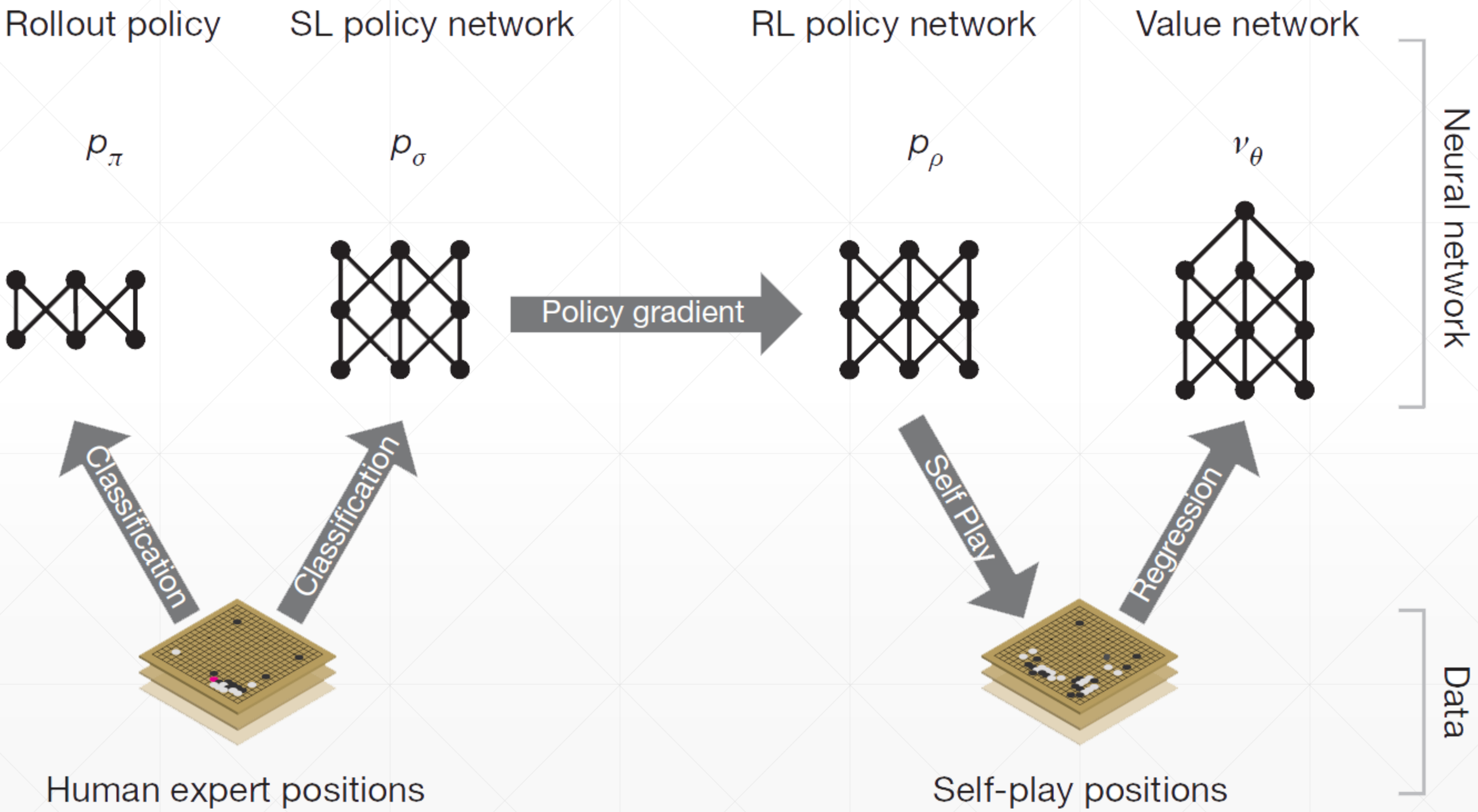
Value Network

- Learn the winner from the self-play games of P_ρ
  - (The Reinforcement Learning policy network)
  - Outputs a single scalar: $v_\theta(s)$

Policy & Value Networks



Training Overview



Supervised Learning Policy Network P_σ

- Learns $P_\sigma(a|s)$ for all valid moves a in state s.
  - The probability of a player to choose the move a
- Alternates between:
  - Convolutional layers
  - Rectified Linear Functions
- With a final layer of softmax
- Outputs probabilities for all legal moves a.

Supervised Learning Policy Network P_σ

- Trained by Stochastic Gradient Ascent
  - On random pairs (s,a) to maximize the likelihood that the human selected a in position s:
    - $\Delta\sigma \propto \frac{\partial \log P_\sigma(a|s)}{\partial \sigma}$

Supervised Learning Policy Network P_σ

- 13 Layer neural network
- Dataset: 30 Million games from the KGS Go Server
  - Augmented with Rotations & Reflections
- ~3 weeks to train on 50 GPUs

SL Policy Network: Features

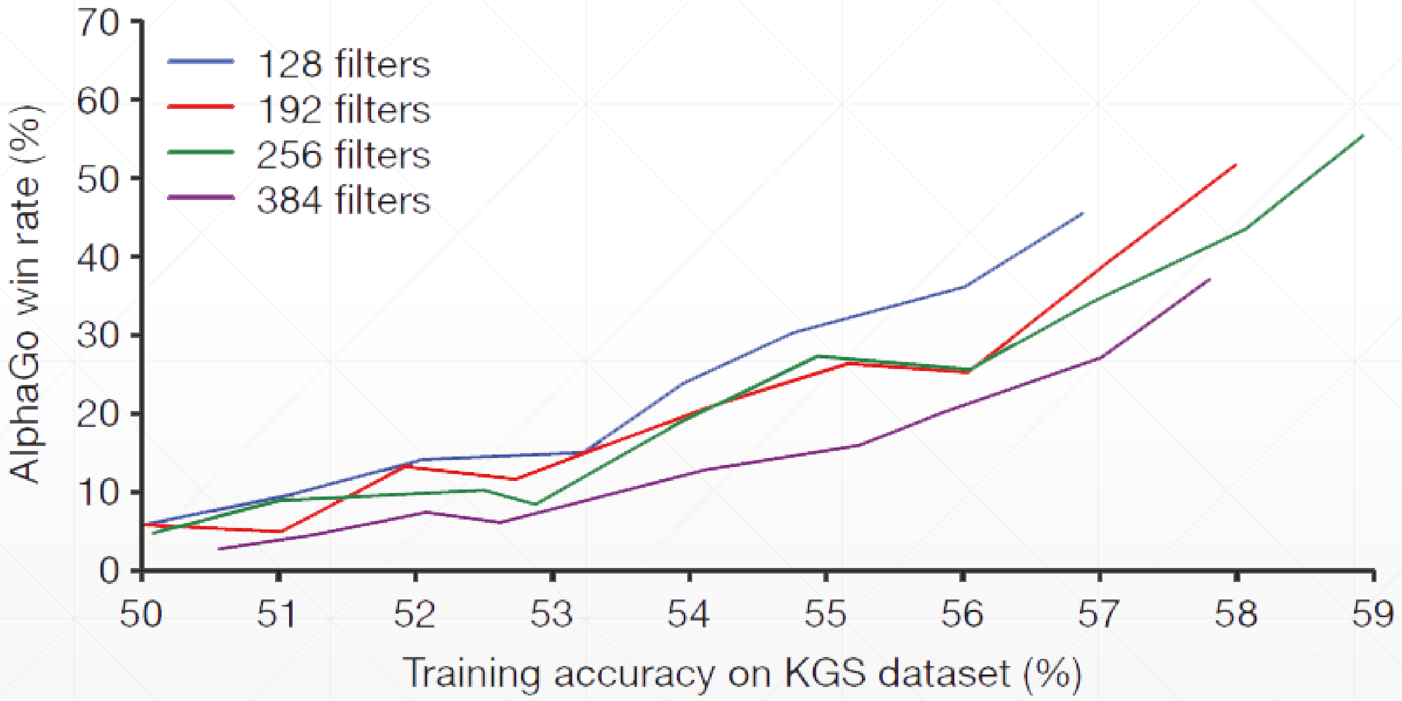
- Features:
  - Stone colors
  - Liberties before & after
  - Number of stones captured
  - 3 other basic move properties

SL Policy Network Evaluation

- Accuracy in predicting player moves:
  - 57.0% success on all features
  - 55.7% success on board + move history only
- Improved the state-of-the-art of 44.4%

SL Policy Network Evaluation

- Every percent gives HUGE playing power



Rollout Policy Network P_π

- Larger networks predict better,
  - But are slower to evaluate during search
- Rollout network $P_\pi(a|s)$ trained on small local features/heuristics
  - 24.2% accuracy
- Much faster:
  - $2\mu s$ to evaluate P_π
  - 3ms to evaluate P_σ

Reinforcement Learning Policy Network P_ρ

- Same structure as P_σ, initialized to the weights of P_σ.
- Plays against a random previous iteration of itself, to avoid overfitting to the last one.
- Play by sampling from the Policy Distribution P_ρ

Reinforcement Learning Policy Network P_ρ

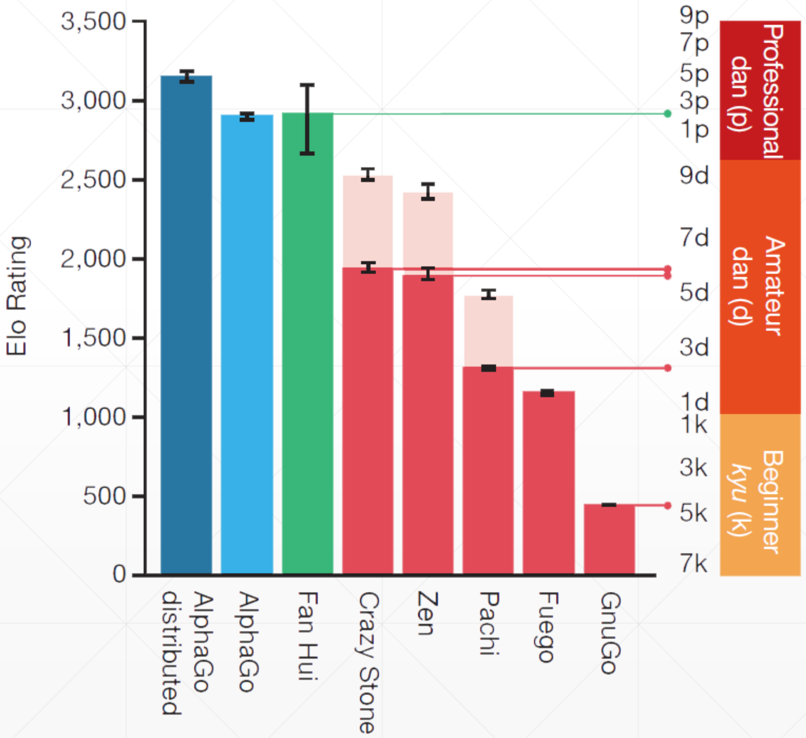
- Reward function z_t:
  - +1 for winning
  - -1 for losing
- Update all weights by Stochastic Gradient Ascent:
  - $\forall t.\ \Delta\rho \propto \frac{\partial \log P_\rho(a_t|s_t)}{\partial \rho} z_t$

RL Policy Network P_ρ: Evaluation

- P_ρ won 80% of the time against P_σ
- Won 85% of games vs Open Source Go player Pachi
  - Amateur 2-dan, uses 100,000 simulations per move.
- Previous State-of-the-art Supervised Learning network won 11% of the time against Pachi.

Dan & Elo scale

- +1 Dan ~ +230 Elo
- +230 Elo wins 79% of the time ⟹ Exponential Scale



Reinforcement Learning Value Network v_θ

- Goal: Estimate $v^P(s)$
  - The mean of the outcome from $\underline{s}$ by using the policy $\underline{P}$ for both players.
  - If P is optimal play, $v^P(s) = v^*(s)$.
    - $v^P(s) = \mathbb{E}[z_t | s_t = s, a_{t..T} \sim P]$
  - Approximate $v^P(s)$ with $v_\theta(s)$.

Reinforcement Learning Value Network v_θ

- Network structure similar to the Policy Network P_σ
  - But outputs a single scalar.
- Train by regression on state-outcome pairs, (s, z) using Stochastic Gradient Descent to minimize the mean-squared-error between $v_\theta(s_t)$ and z_T
  - $\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}\left(z - v_\theta(s)\right)$

Value Network v_θ: Training

- Naïve training leads to overfitting
  - Successive boards very similar, with same result.
  - Training: MSE = 0.19 Test: MSE = 0.37
- Solution: Train on larger set, generated by self-play.
  - 1 State sampled per game.
  - Training: MSE = 0.226 Test: MSE = 0.234
  - Not much overfitting

Value Network v_θ: Quality

- v_θ was consistently more accurate than rollouts with P_π
- A single evaluation of v_θ approached the accuracy of rollouts with P_ρ
  - But with 15,000 times less computation

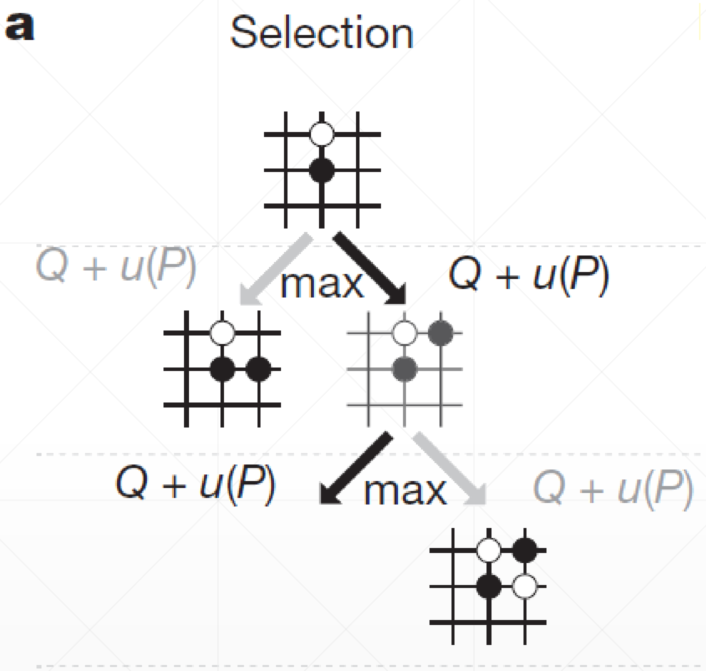
MCTS - Monte Carlo Tree Search

- Build an assymetric search tree
- Use more time on promising subtrees
- But explore too: maybe there is a hidden treasure somewhere.
- Proven to converge to true minimax value asymptotically

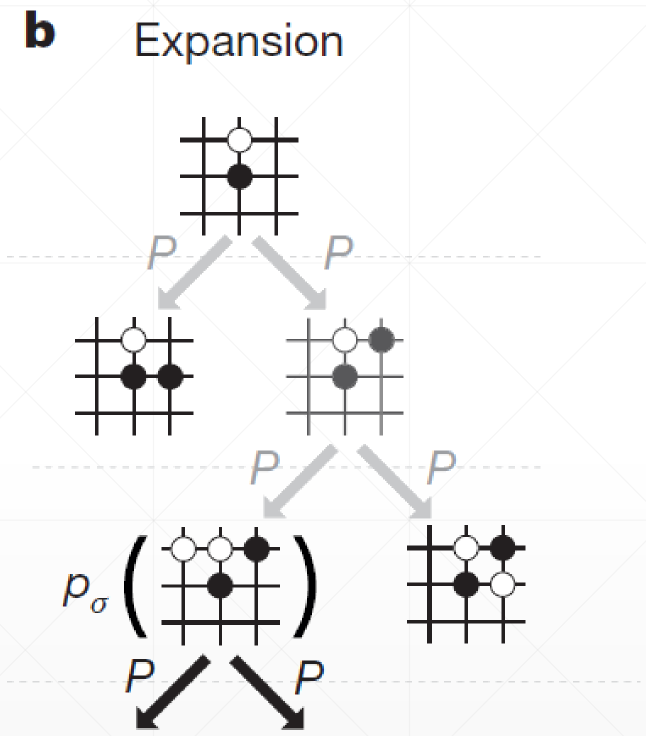
MCTS - Monte Carlo Tree Search

- Each edge contains:
  1. $N(s, a)$ = # of visits to the edge
  2. $Q(s, a)$ = Mean of $V\left(s_L^i\right)$ - values of leafs in the subtree
  3. $P(s, a)$ = Prior probability of choosing the move a
- $u(s, a)$ = bonus for exploration

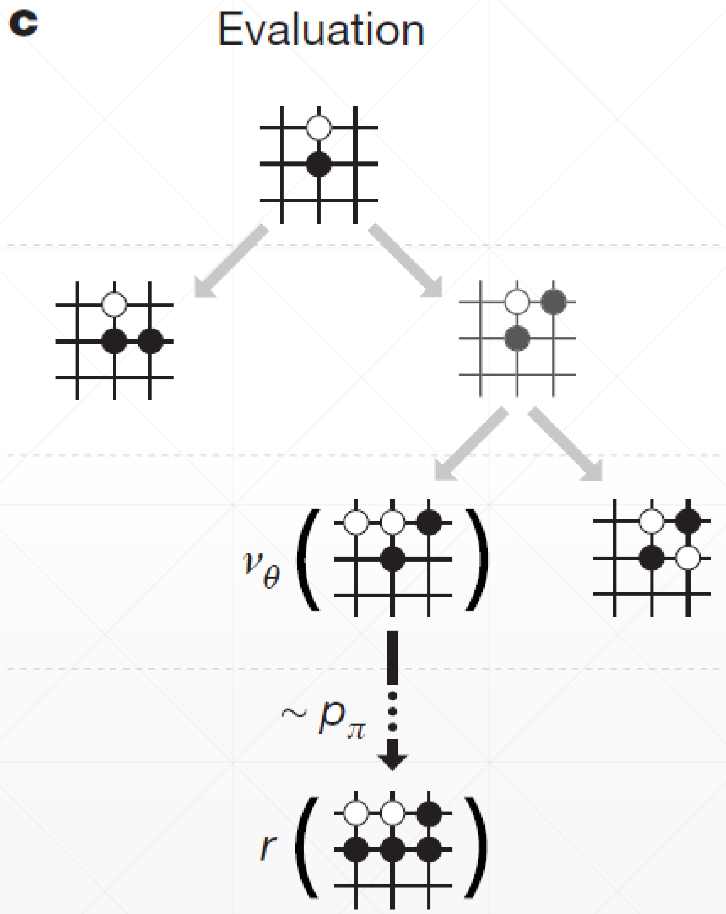
MCTS - Monte Carlo Tree Search



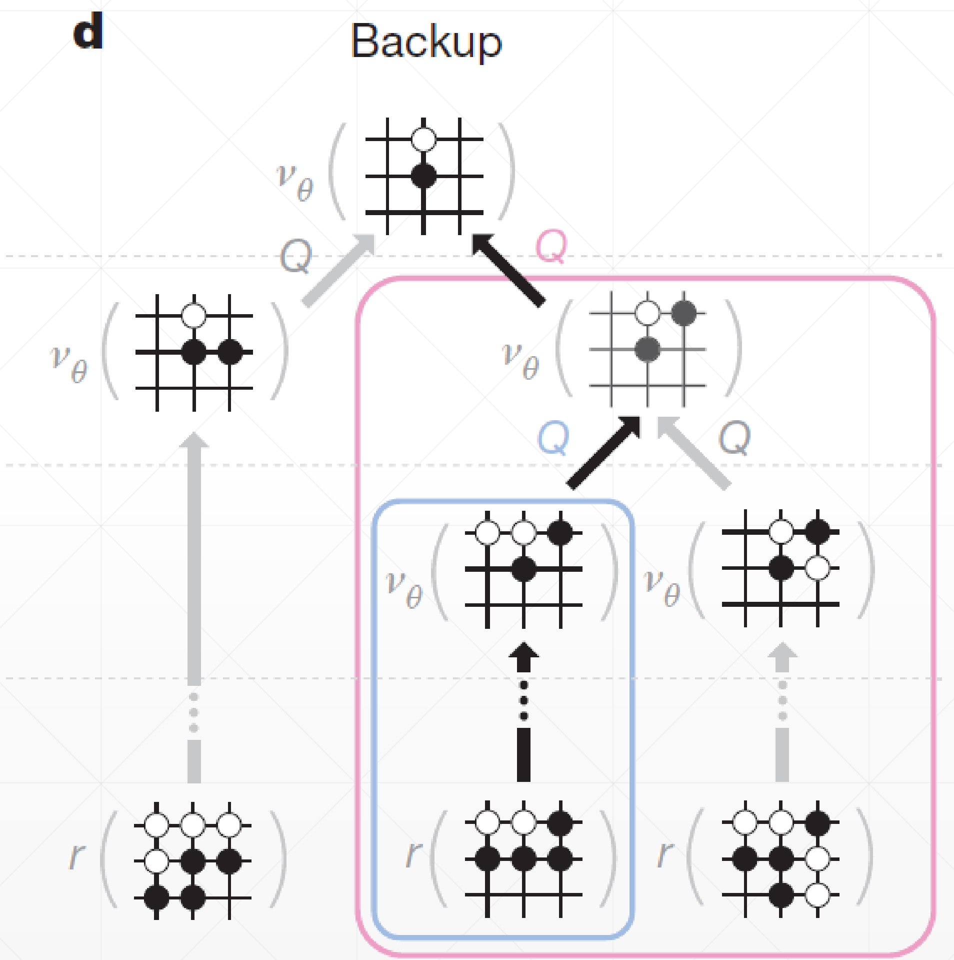
MCTS - Monte Carlo Tree Search



MCTS - Monte Carlo Tree Search



MCTS - Monte Carlo Tree Search



MCTS with Policy & Value Networks

1. Selection:
   - Run from the root to a leaf by:
     - $argmax_a\left(Q(s,a)+u(s,a)\right)$
   - Were $u(s,a)$ is a bonus, which starts with $P(s,a)$ and decays to 0 to encourage exploration.
     - $u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$

MCTS with Policy & Value Networks

2. Expansion:
   - Open valid moves a from a leaf s_L
   - Process the leaf s_L by the Supervised Learning Policy Network P_σ
   - Set the prior probabilities to the outputs:
     - $P(s_t, a) = P_\sigma(a|s_t)$

MCTS with Policy & Value Networks

3. Evaluation:
   1. By the Value Network $v_\theta(s_L)$
   2. By the outcome z_L of a rollout by P_π
   3. Mixed together linearly:
      - $V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$

- λ was chosen to be 0.5

MCTS with Policy & Value Networks

4. Backup / Backpropagation:
   - Update $N(s,a)$
   - Update $\mathrm{Q}(s,a)$
   - For all nodes in the path to the leaf

MCTS with Policy & Value Networks

5. Final selection:

- Select action with most visits from the root.
  - More stable than best Q
  - Allocate more time until they agree

MCTS with Policy & Value Networks: Notes

- Notes:
  - The SL Policy Network P_σ performed better for Prior Probabilities than the RL Network P_ρ
  - The RL Policy Network P_ρ was used in training the Value Network v_θ, which estimates the strong play of P_ρ.
  - This is because the SL Network gave more diverse probabilities, vs the RL Network, that was focused on the single best move.

AlphaGo Execution

- Distributed Execution:

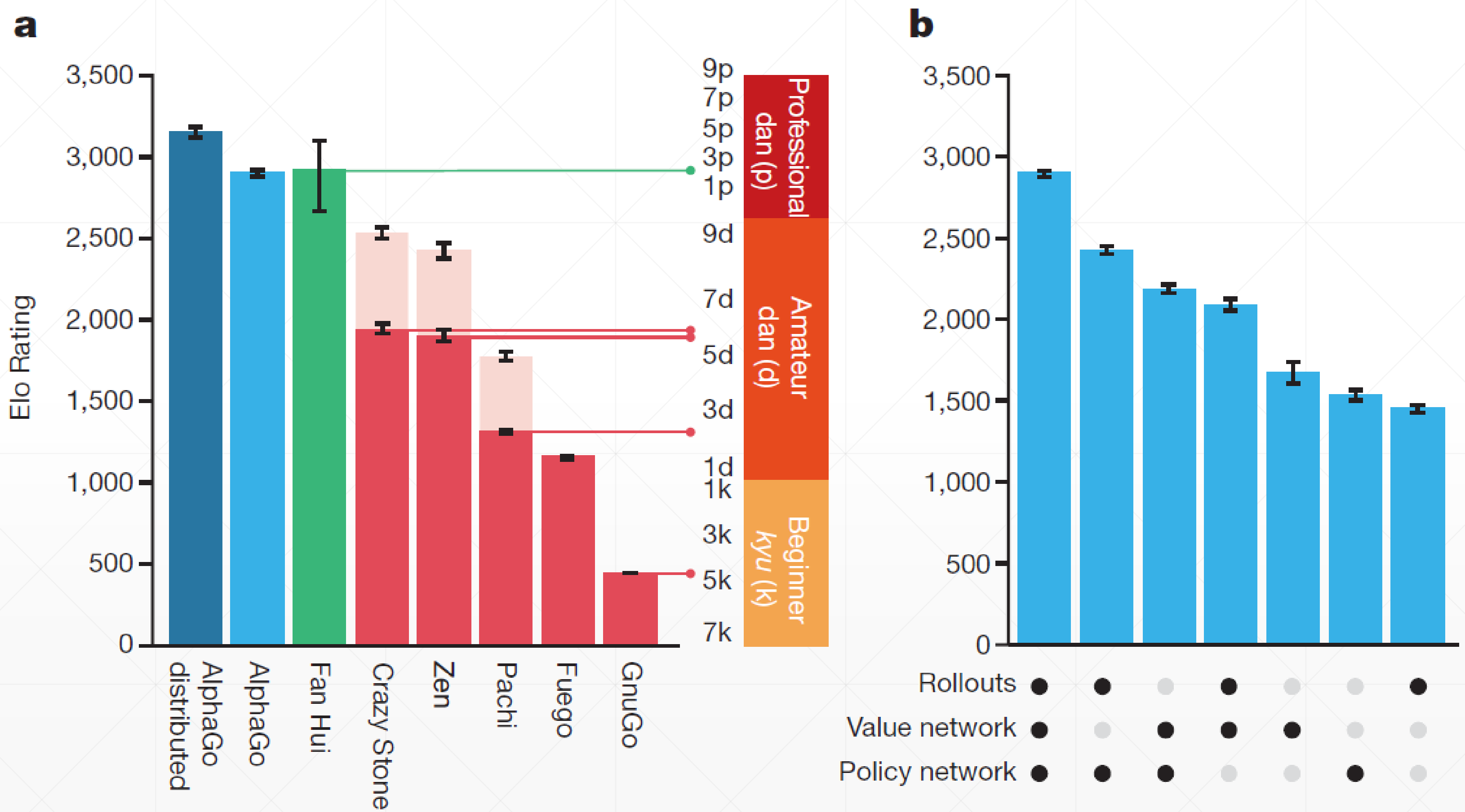
1. Master CPU: The search tree
2. Slave CPUs: Rollouts
3. Slave GPUs: Value & Policy Networks

- Virtual loss: To prevent different threads traversing the same paths

AlphaGo Execution

- Time management:
  - Give more time for mid-game
- Ethics:
  - Resign if winning probability low (<20%)

AlphaGo performance



Discussion

- Beat Lee Sedol, 9-Dan Go player, world champion
  - 4/5 games
- Evaluated thousands of times less positions than DeepBlue
- No hand-crafted evaluation function

Discussion

- MCTS led to many advances in:
  - General Game Playing (GGP)
  - Planning
  - Scheduling
  - Constraint Satisfaction
  - Feature Selection
  - & Many more!

Discussion

- AI Grand Challenge Completed
- Starcraft next up!
  - Partial information
  - Continuous
  - Real-time

Resources

- Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.
- Browne, Cameron B., et al. "A survey of monte carlo tree search methods." Computational Intelligence and AI in Games, IEEE Transactions on 4.1 (2012): 1-43.
- Artificial Intelligence and The Future

Thank you!