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

Qingqing Cao 110452148
Yingkai Li 110568795
Ping Hu 110560827

CSE 537 Artificial Intelligence
Professor Anita Wasilewska
Department of Computer Science
Stony Brook University

Reference

1. Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.
2. https://youtu.be/yCALyQRN3hw?list=PLqYmG7hTraZA7v9HpbpsOQNmJC4L1NE3S&t=11434

Game of Go

Go, literally 'encircling game', is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent.

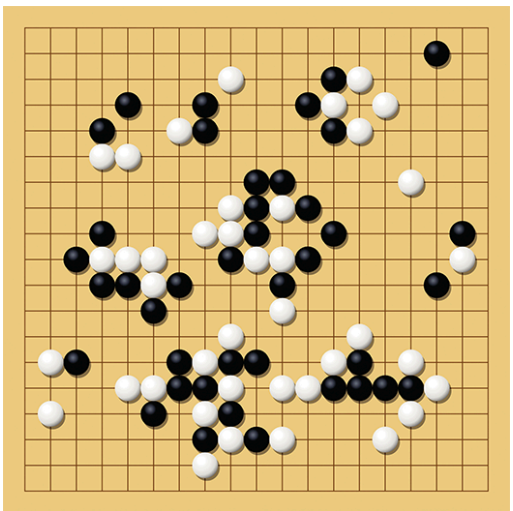


Figure: Game of Go

Game of Go

Artificial Intelligence characteristics:

1. enormous search space: $\sim 10^{172}$.
2. difficulty of evaluating board positions and moves: $\sim 10^{360}$ game-tree complexity.

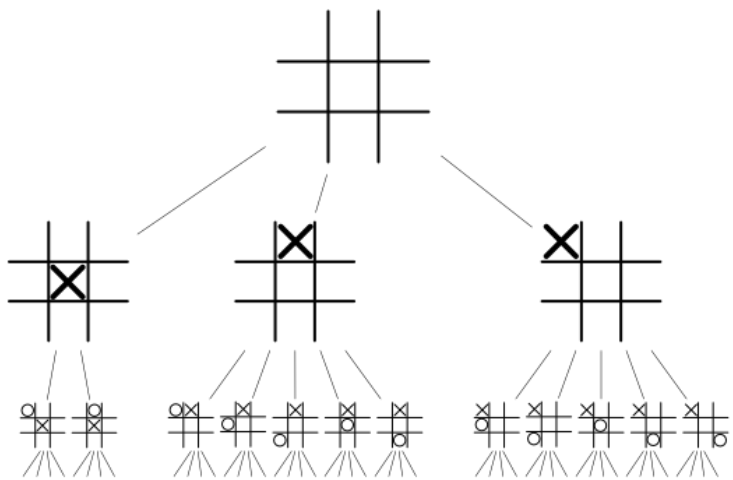


Figure: Game tree of tic-tac-toe

Related Works

General ideas:

1. All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game.

2. Recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves

Related Works

Effectively reducing search space:

1. Search depth may be reduced by position evaluation:
   * truncating the search tree at state s and
   * replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s.

Related Works

Effectively reducing search space:

1. Search depth may be reduced by position evaluation:
   * truncating the search tree at state s and
   * replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s.

   Still intractable in Go due to the complexity!

Related Works

Effectively reducing search space:

1. Search depth may be reduced by position evaluation:
   * truncating the search tree at state s and
   * replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s.

   Still intractable in Go due to the complexity!

2. Search breadth may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s.

Related Works

Effectively reducing search space:

1. Search depth may be reduced by position evaluation:
   * truncating the search tree at state s and
   * replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s.

   Still intractable in Go due to the complexity!

2. Search breadth may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s.
   Example Monte Carlo rollouts search still leads weak amateur level play in Go!

Monto Carlo Tree Search

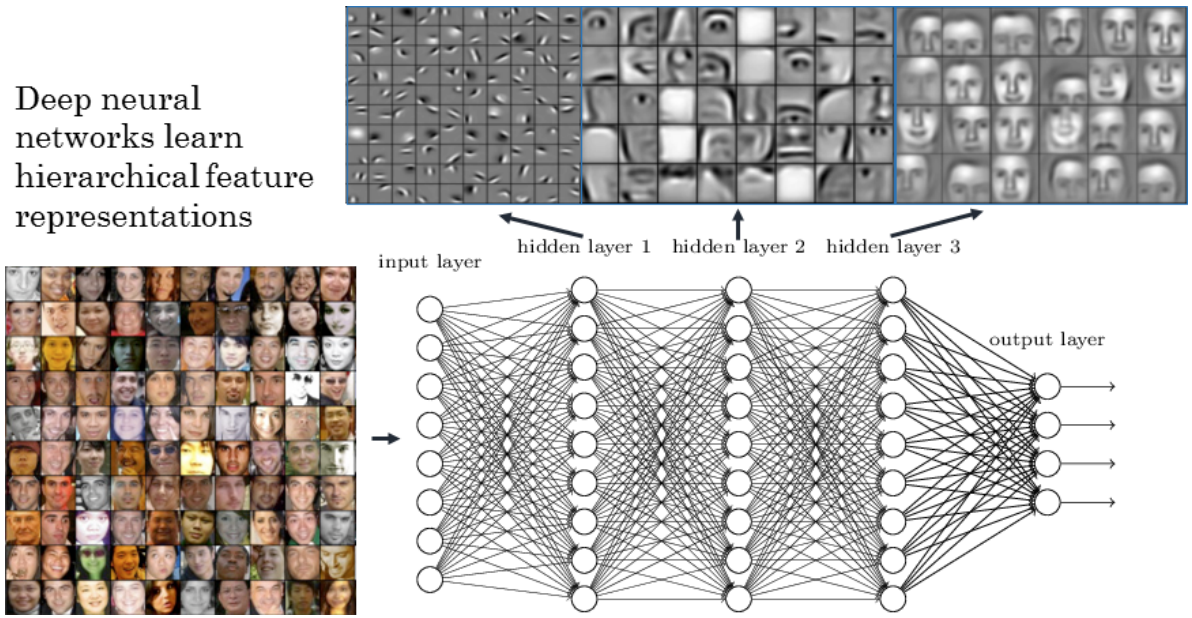
Monte Carlo tree search (MCTS) uses Monte Carlo rollouts to estimate the value of each state in a search tree.

1. More simulations, tree grows larger and relevant values become more accurate.
2. Policy to select actions improves over time.
3. Evaluations converge to the optimal value function.

Supervised Deep Learning

Deep convolutional neural networks:

* Use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image. (Image credit: `http://www.rsipvision.com/exploring-deep-learning/`)



Supervised Deep Learning

For the game of Go:

* Pass in the board position as a $19x19$ image and use convolutional layers to construct a representation of the position.
* Use these neural networks to reduce the search tree space
* evaluating positions using a value network, and sampling actions using a policy network.
* Training a supervised learning policy network directly from expert human moves.

Policy Network and Value Network

1. Train a supervised learning (SL) policy network p_σ directly from expert human moves.
2. Train a fast policy p_π that can rapidly sample actions during rollouts.
3. Train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play.
4. Train a value network v_θ that predicts the winner of games played by the RL policy network against itself.

Policy Network and Value Network

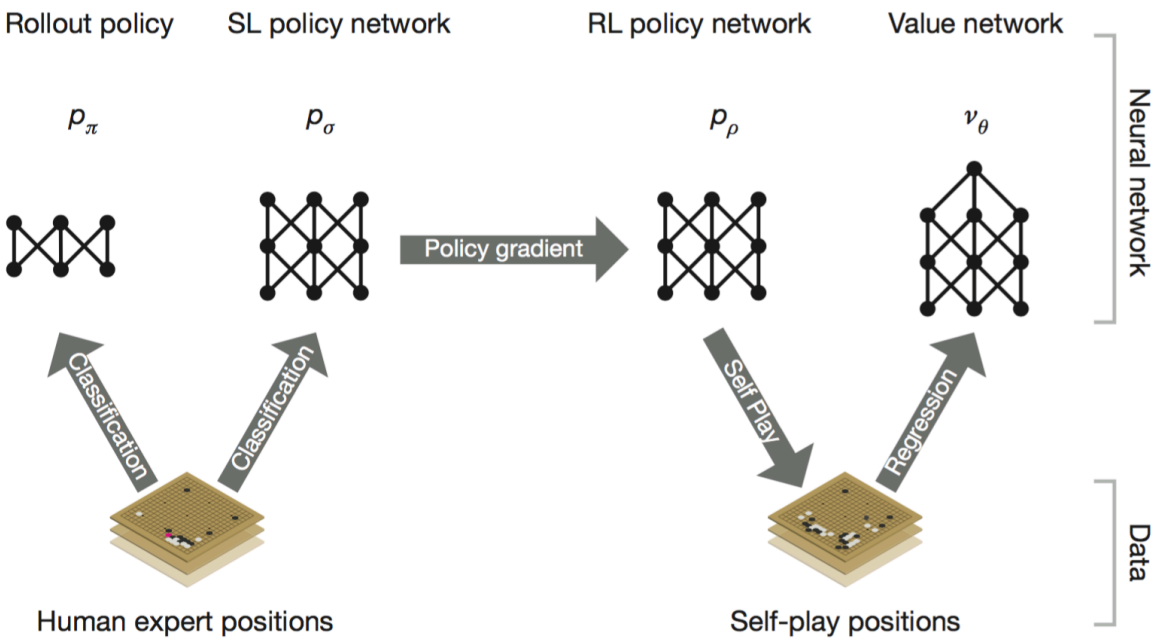


Figure: Neural Network Training Pipeline and Architecture

Policy Network and Value Network

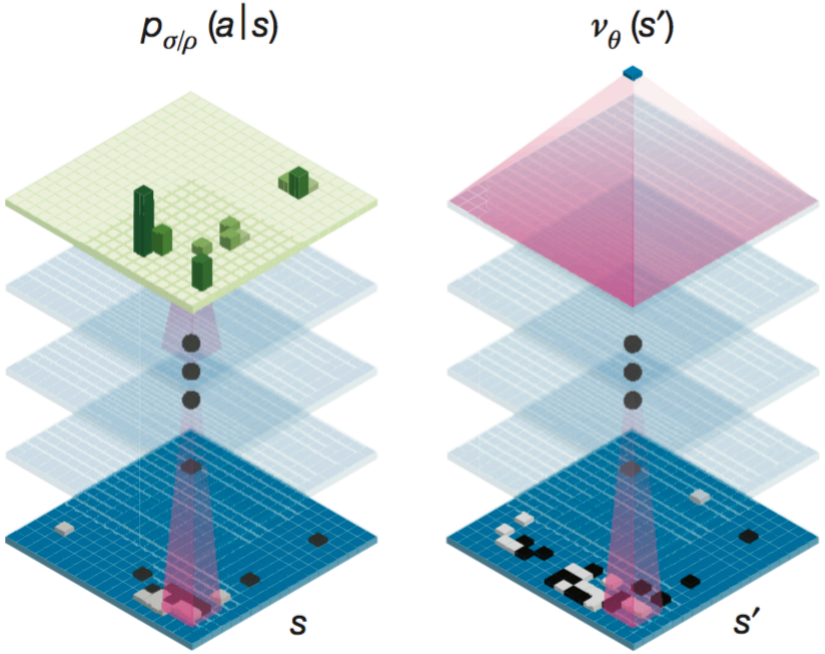


Figure: Left: Policy Network; Right: Value Network

Policy Network and Value Network

1. Accuracy of 57.0% using all input features, and 55.7% using only raw board position and move history as inputs for p_σ. [1]
2. Accuracy of 24.2% for p_π.
3. Time cost of $2\mu s$ to select an action for p_π, and time cost of $3ms$ for the policy network p_σ.

[1] Accuracy of 44.4% for others work. However, we can see that even a slight difference in prediction accuracy can cause huge difference in the probability of winning.

Policy Network and Value Network

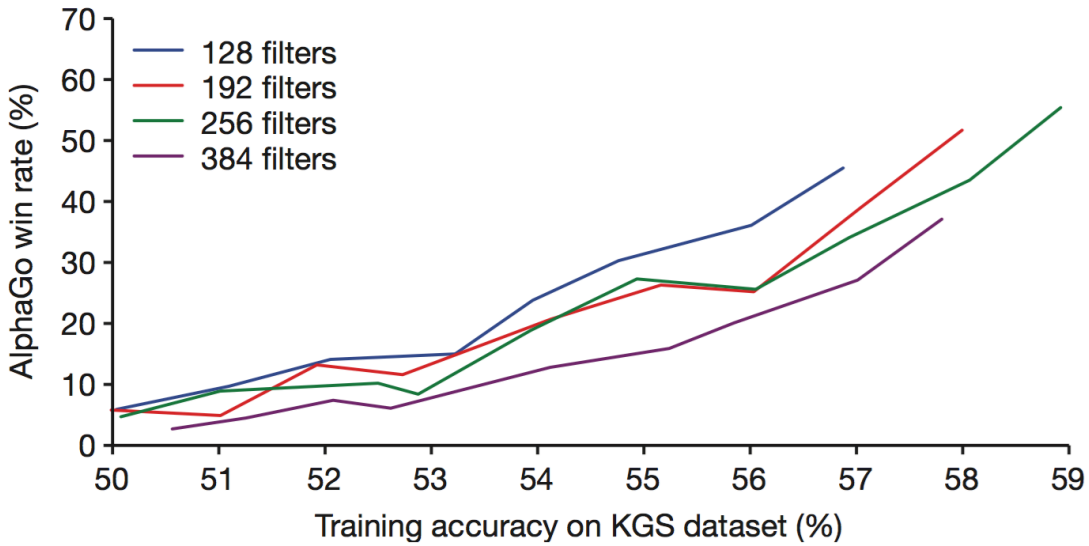


Figure: Left: Policy Network; Right: Value Network

Reinforcement Learning for Policy Network

1. Identical in structure to the SL policy network.
2. Weights ρ are initialized to the same values, $\rho = \sigma$.
3. Weights are then updated at each time step t by stochastic gradient ascent in the direction that maximizes expected outcome

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\rho} z_t$$

Here z_t is the terminal reward at the end of the game from the perspective of the current player at time step t: $+1$ for winning and -1 for losing.

Reinforcement Learning for Policy Network

1. Win more than 80% of games against the SL policy network.
2. Win 85% of games against Pachi, a sophisticated Monte Carlo search program, ranked at 2 amateur dan on KGS.

Reinforcement Learning for Value Network

1. Has a similar architecture to the policy network, but outputs a single prediction instead of a probability distribution.
2. Use stochastic gradient descent to minimize the mean squared error (MSE) between the predicted value $v_\theta(s)$, and the corresponding outcome z,

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z_t - v_\theta(s))$$

Monte Carlo Tree Search

1. At each time step t of each simulation, an action a_t is selected from state s_t

$$a_t = \arg\max(Q(s_t, a) + u(s_t, a))$$

2. The node is evaluated in two very different ways:
   - By the value network $v_\theta(s_L)$;
   - By the outcome z_L of a random rollout played out until terminal step T using the fast rollout policy p;

   These evaluations are combined, using a mixing parameter λ, into an evaluation $V(s_L)$.

$$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$$

3. The algorithm chooses the most visited move from the root position.

Monte Carlo Tree Search

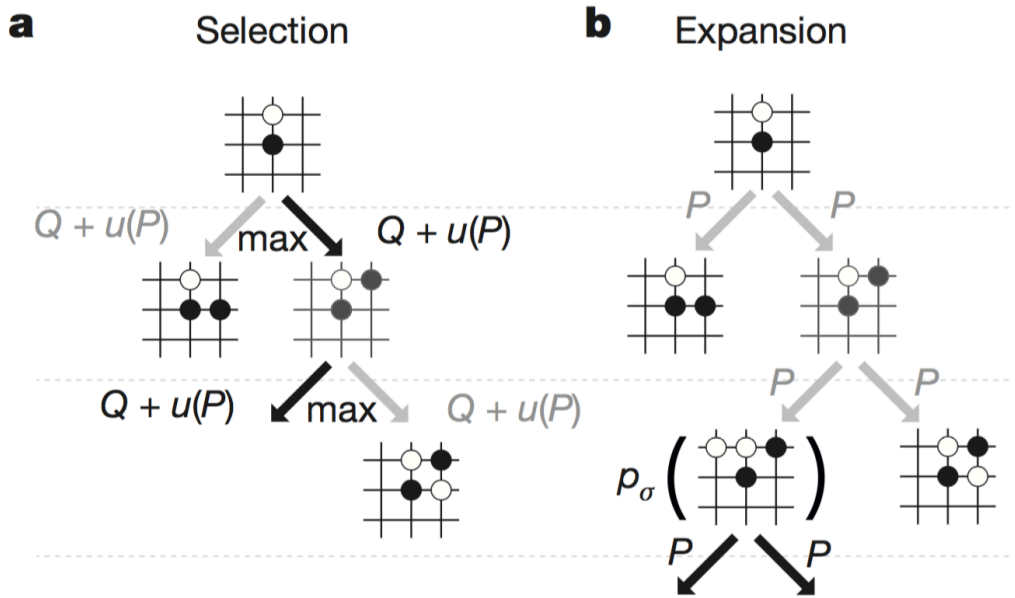


Figure: Monte Carlo Tree Search in AlphaGo

Monte Carlo Tree Search

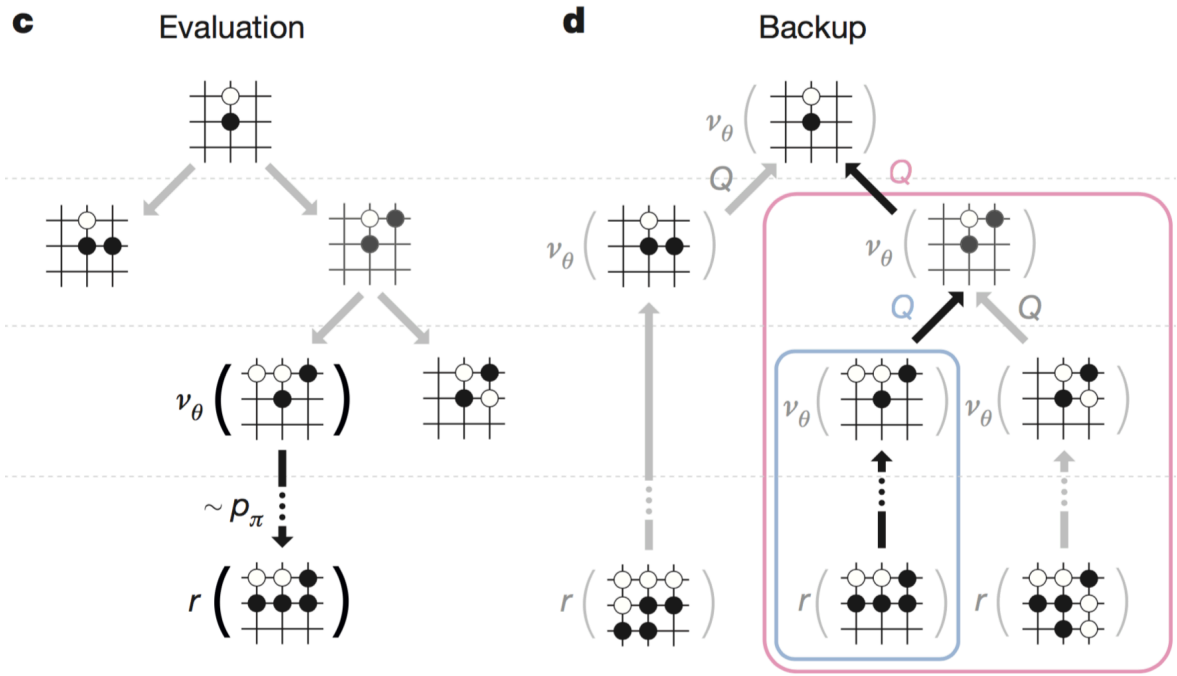


Figure: Monte Carlo Tree Search in AlphaGo

Tournament evaluation of AlphaGo

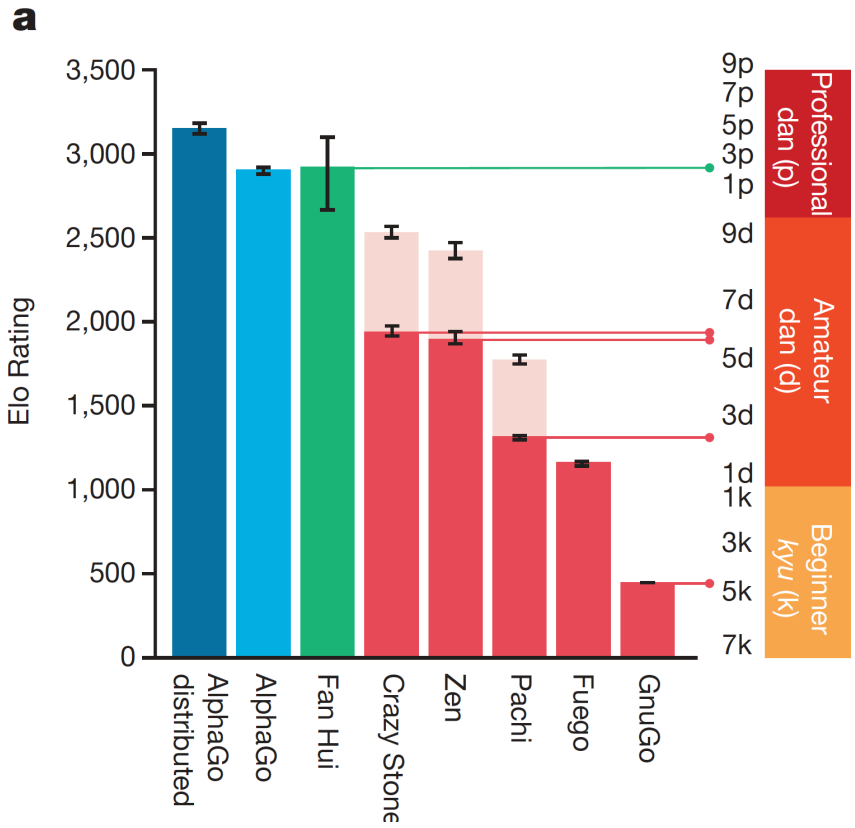


Figure: Results of a tournament between different Go programs

Tournament evaluation of AlphaGo

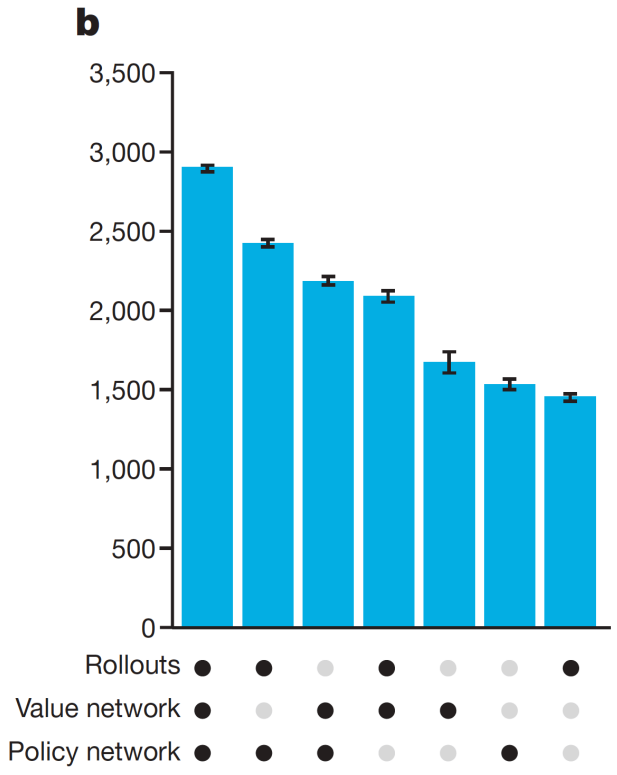


Figure: Performance of AlphaGo (on a single machine) for different combinations of components

Evaluation of the value network and rollouts

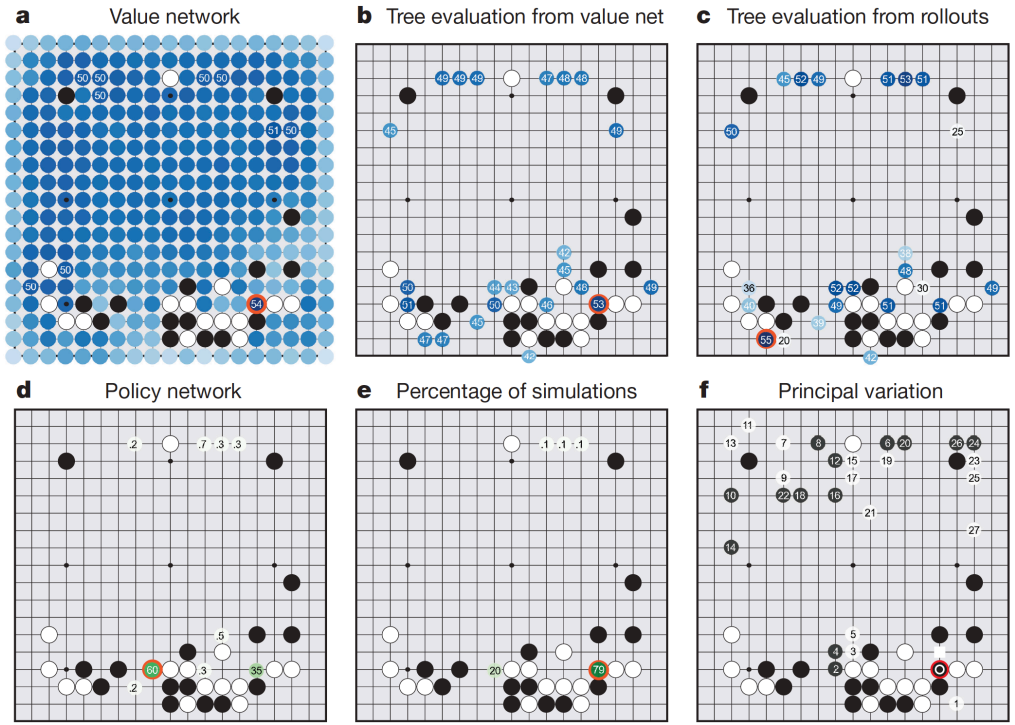


Figure: How AlphaGo (black, to play) selected its move in an informal game against Fan Hui

Weakness

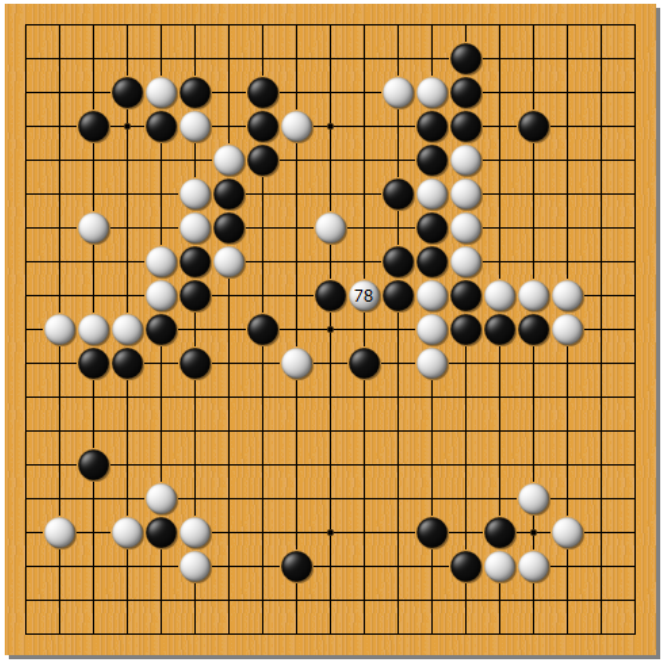


Figure: Match 4 of AlphaGo vs Lee Sedol: Lee played a brilliant move 78.

Weakness

In AlphaGo's search space, when the probability of the opponent's moves which leads to AlphaGo's losing is quite small, it is possible that AlphaGo may not find these moves, resulting in that AlphaGo mistakes B_k as the next move with the "highest" value in the policy network. However, in fact B_k leads to a severer losing in the subsequent moves.

Weakness

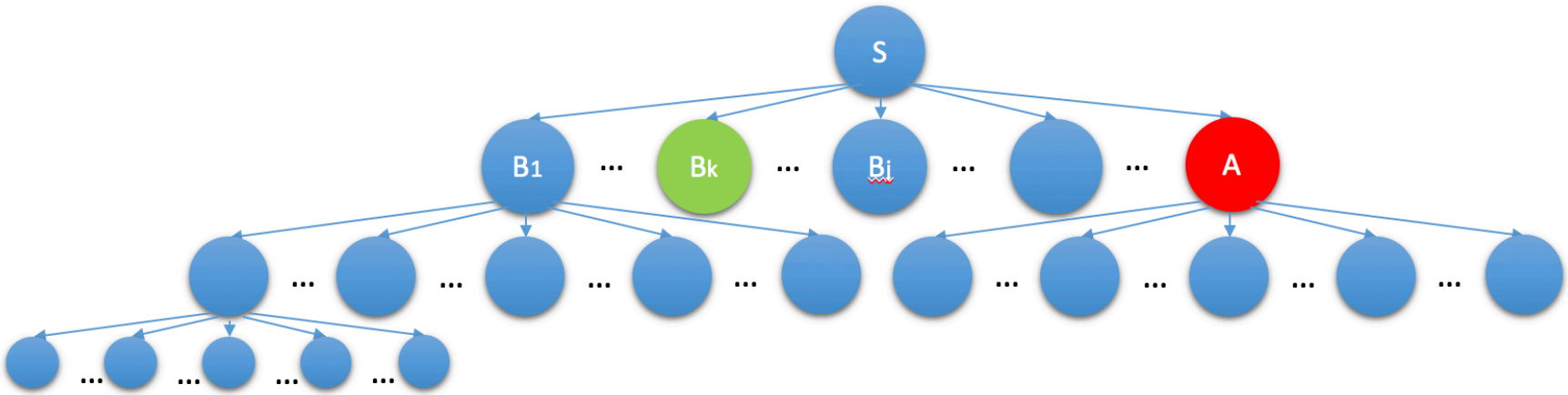


Figure: S: current state of the match. A: optimal move with the highest value in value network. B_1, ..., B_j are moves with lower values in the value network. B_k is AlphaGo's actual next move.

Summary

1. A Go game at the highest level of human players is developed based on a combination of **deep neural networks** and **tree search**.
2. For the first time, **effective move selection** and **position evaluation functions** for Go are developed based on deep neural networks.

Thank You!