AlphaGo Zero

Silver D., Schrittwieser J., Simonyan I., et al. 2017

Overview

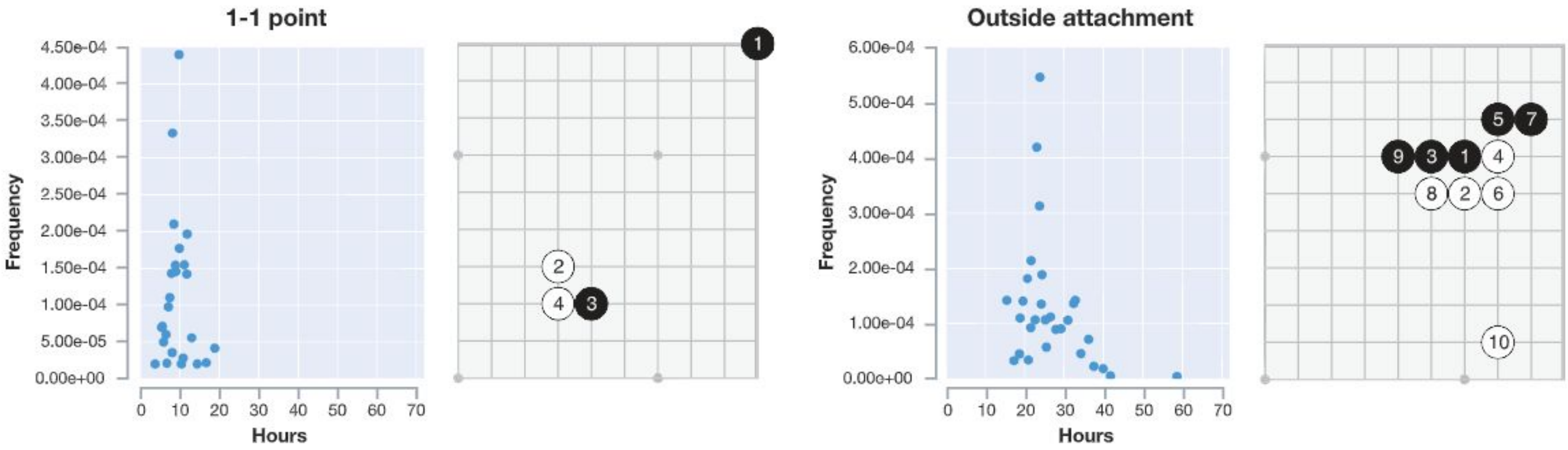
- Cool Findings
- Neural Network Architecture
- Search Algorithm
- Training Pipeline
- Analysis

Overview

- Cool Findings
- Neural Network Architecture
- Search Algorithm
- Training Pipeline
- Analysis

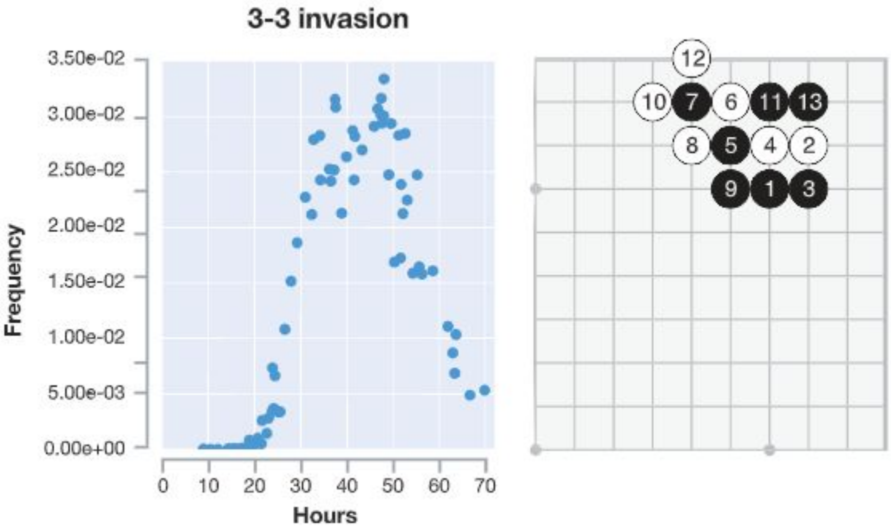
Cool Findings

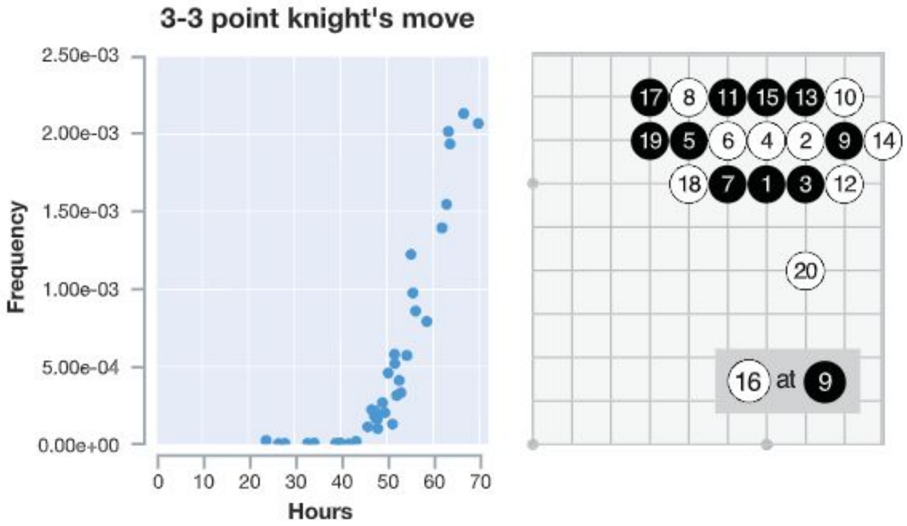
- Discovery of common human moves, then later discarding them for novel moves



Cool Findings

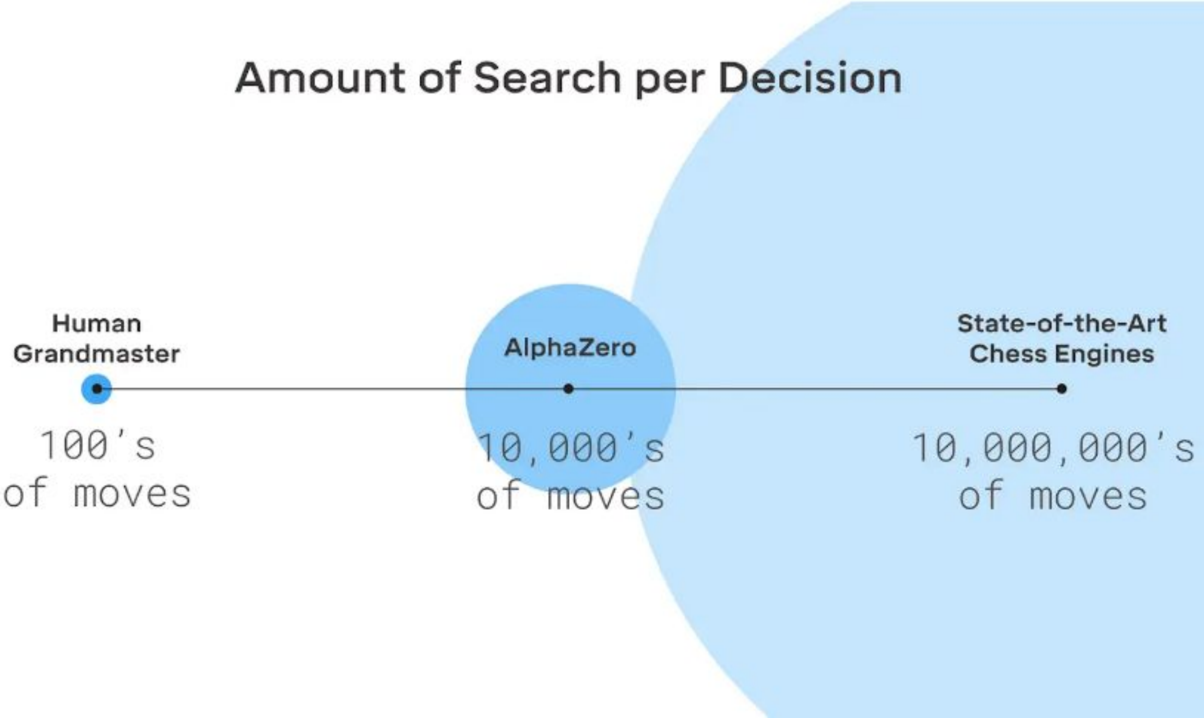
- ❑ Discovery of common human moves, then later discarding them for novel moves





Cool Findings

- Number of moves considered, human vs AlphaZero vs Stockfish



Overview

- Cool Findings
- Neural Network Architecture
- Search Algorithm
- Training Pipeline
- Analysis

Neural Network Architecture: Inputs

$$s_t = [X_t, Y_t, X_{t-1}, Y_{t-1}, \ldots, X_{t-7}, Y_{t-7}, C]$$

$[X_t, \ldots, X_{t-7}]$ 19 X 19 binary feature plane of current players stones over last 8 moves.

$[Y_t, \ldots, Y_{t-7}]$ 19 x 19 binary feature plane of opponents stones over last 8 moves.

$[C]$ Handicap points to indicate who's turn it is.

Neural Network Architecture: Outputs

p_t

Policy vector
Probability distribution over all moves including pass.

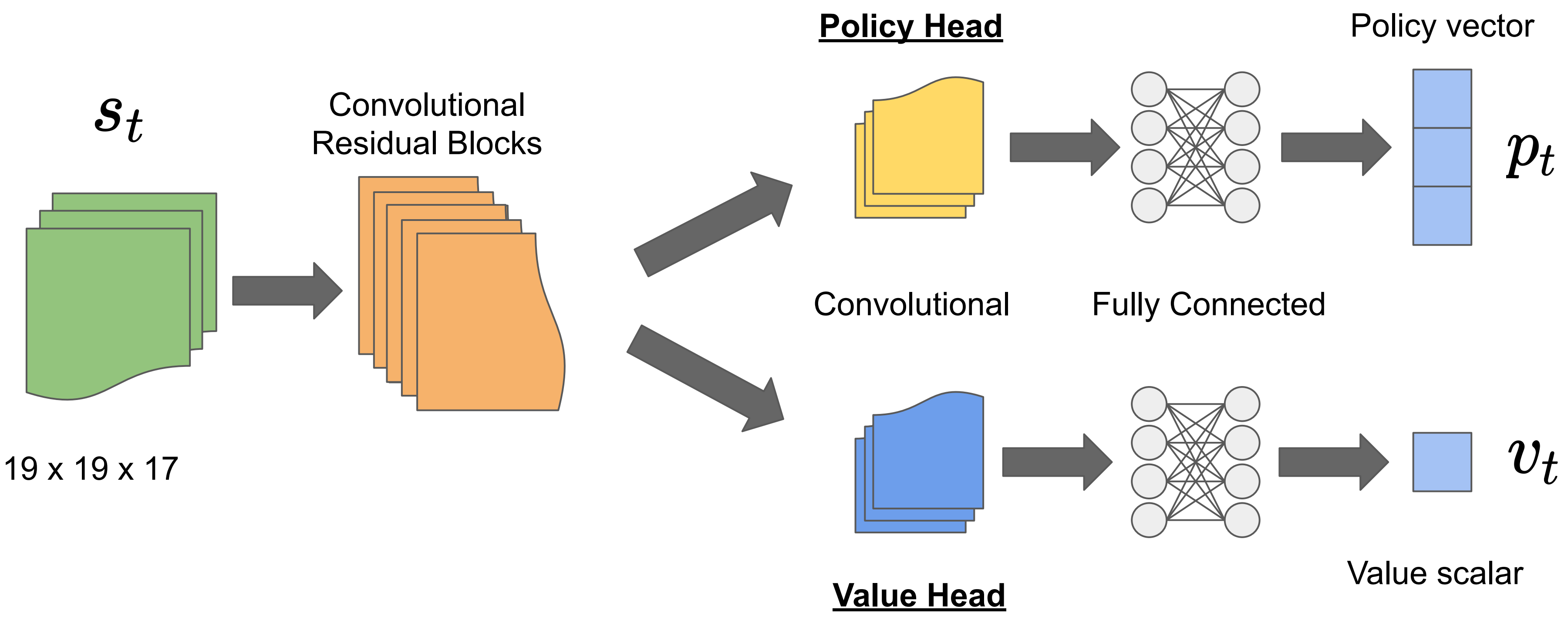
- Probability corresponds to how good the move is
- 19 x 19 + 1 = 362

v_t

Value scalar
Scalar indicating the likelihood of winning from current state.

- [-1,1]

Neural Network Architecture



Neural Network Architecture

$$f_\theta(s_t) = (p_t, v_t)$$

$$l = (z - v)^2 - \pi^T log(p) + c||\theta||^2$$

Overview

- Cool Findings
- Neural Network Architecture
- Search Algorithm
- Training Pipeline
- Evaluation

Search Algorithm

Vanilla Monte Carlo Tree Search (MCTS)

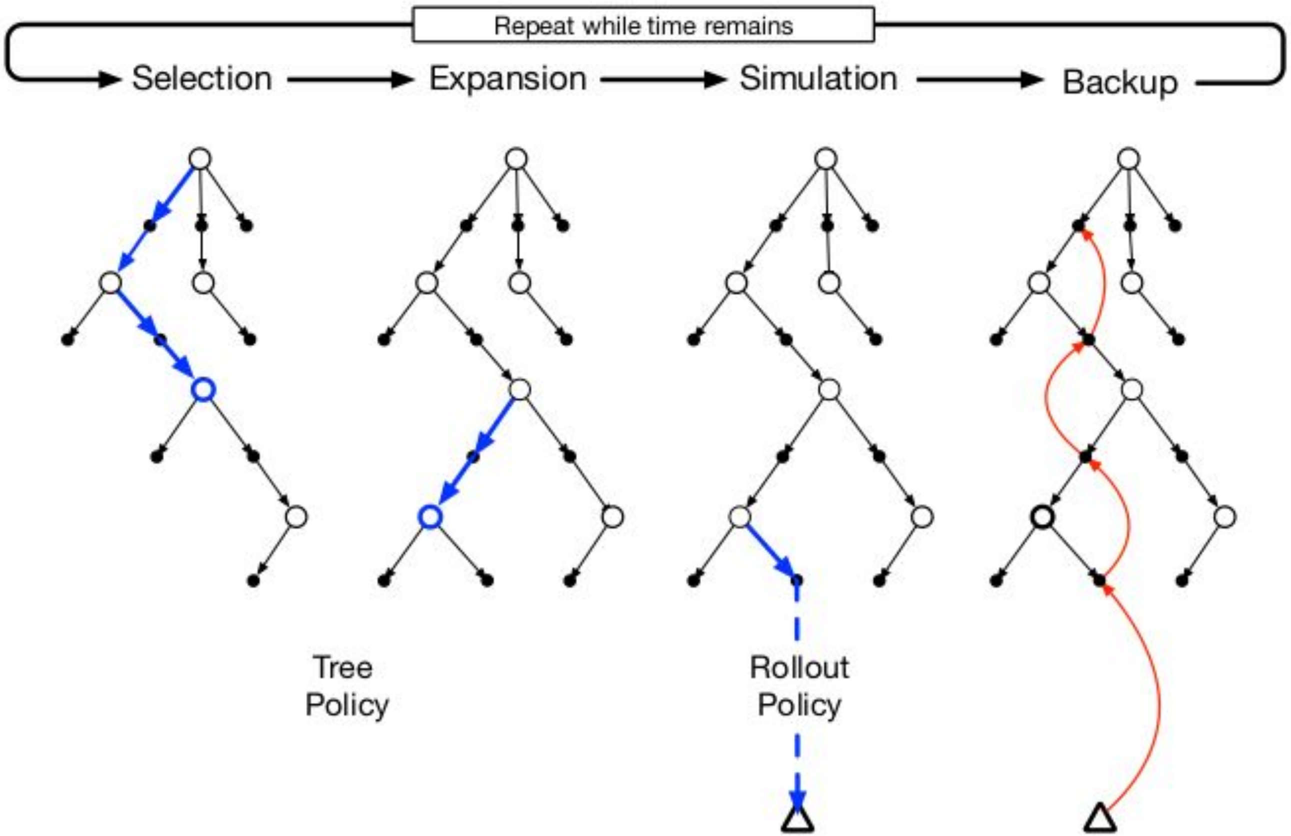
- ❑ Coulom, R. 2006, Kocsis, L., Szepesvari, C. 2006.
- ❑ Utilizes multiple simulated trajectories from current state to pick best action
- ❑ Focus successive simulations to extend trajectories of high value
- ❑ Effective when environment model is simple enough for fast simulation

Search Algorithm

Vanilla Monte Carlo Tree Search (MCTS)

1. **Selection** - start at root and choose the best path
2. **Expansion** - once at leaf node, expand one or more child nodes
3. **Simulation** - simulate trajectory until terminal state
4. **Backup** - use terminal value to update tree

Search Algorithm



Sutton, R., Barto, A., Reinforcement Learning 2018

Search Algorithm

AlphaGo Zero (MCTS)

- ❑ Utilizes the neural network to guide search $$f_\theta(s_t) = (p_t, v_t)$$
- ❑ Reduces the breath of the search with the policy vector p_t
- ❑ Reduces the depth of the search the value scalar v_t

Search Algorithm

AlphaGo Zero (MCTS)

- ❏ Each edge (s, a) stores statistics: $\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$
  - ❏ Visit count $N(s,a)$
  - ❏ Total action value $W(s,a)$
  - ❏ Mean action value $Q(s,a)$
  - ❏ Prior probability $P(s,a)$

Search Algorithm

AlphaGo Zero (MCTS)

1. **Selection** - Choose edges according to edge statistics

$$a_t = \underset{a}{argmax}(Q(s_t, a) + U(s_t, a))$$

- Q is the **exploitative** term
- U is the **exploration** term

$$U = c_{punct} P(s, a) \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

Search Algorithm

AlphaGo Zero (MCTS)

1. **Selection**

$$U = c_{puanct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

- Exploration constant c_{puanct}
- Prior probability $P(s,a)$
- Parent visit count $\sum_b N(s,b)$
- Edge visit count $N(s,a)$

Search Algorithm

AlphaGo Zero (MCTS)

2. **Expand and evaluate** - once a leaf node is reached, expand node with NN

$$f_\theta(s_t) = (p_t, v_t)$$

- ❑ Each edge is initialized:

$$\{N(s,a) = 0, W(s,a) = 0, Q(s,a) = 0, P(s,a) = p_t\}$$

- ❑ The value is used for back up: v_t

Search Algorithm

AlphaGo Zero (MCTS)

3. **Backup** - edge statistics are update in backward pass to the root

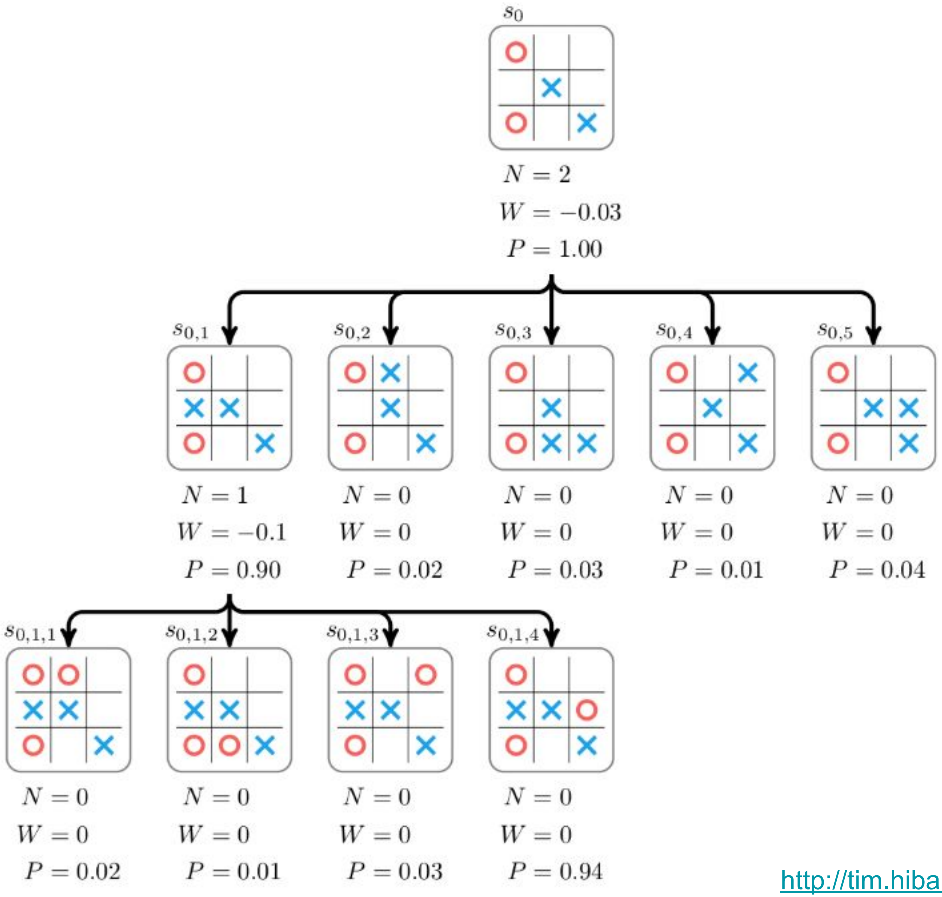
- ❏ Node visit counts are incremented:

$$N(s_t, a_t) = N(s_t, a_t) + 1$$

- ❏ Action value is updated to the mean value:

$$W(s_t, a_t) = W(s_t, a_t) + v, \quad Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$$

Search Algorithm



http://tim.hibal.org/blog/alpha-zero-how-and-why-it-works/

Search Algorithm

AlphaGo Zero (MCTS)

4. **Play** - choose action a to play in root position s_0

$$\pi(a \mid s_0) = \frac{N(s_0,a)^{1/\tau}}{\sum_b N(s_0,b)^{1/\tau}}$$

- ❑ Temperature parameter, controls exploration: τ
- ❑ Edge visit count: $N(s_0, a)$
- ❑ Root visit count: $\sum_b (s_0, b)$

Search Algorithm

AlphaGo Zero (MCTS)

4. **Play** - save MCTS policy, state, and value placeholder for NN training

$$\left(s_t\,, \pi_t\,, z_t\right)$$

- At end of game, the value placeholder is updated with final reward:

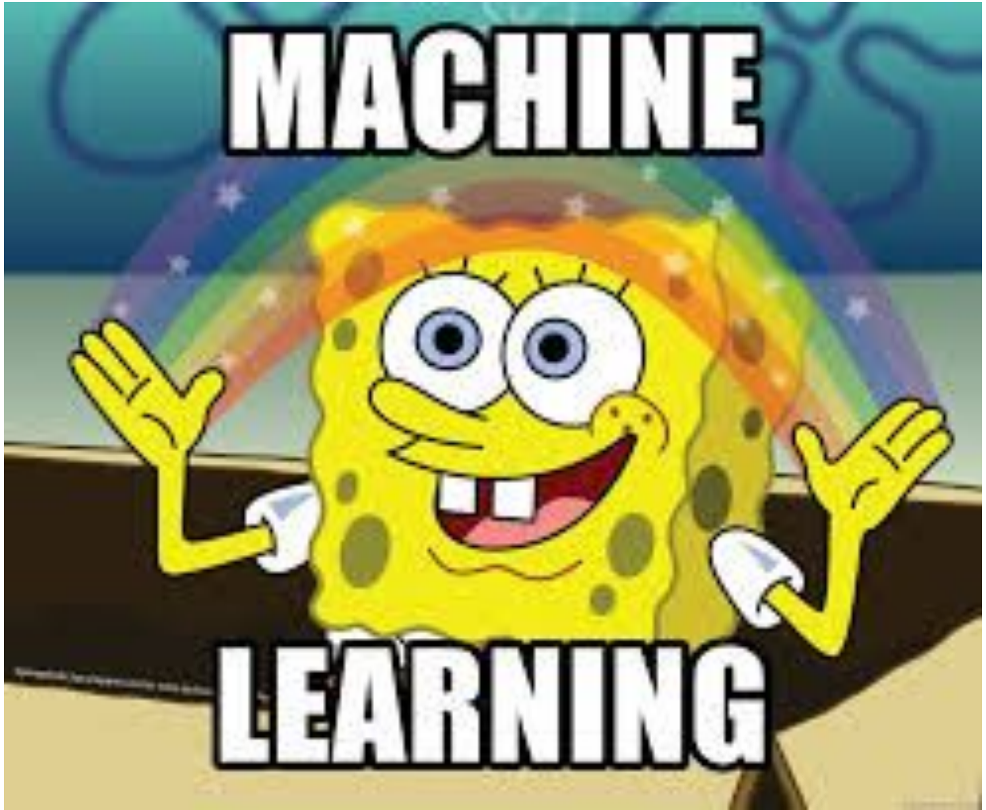
$$z_t = r_T = \{-1, +1\}$$

Overview

- Cool Findings
- Neural Network Architecture
- Search Algorithm
- Training Pipeline
- Analysis

Training Pipeline

We have a NN and we can use it to play games. How do we make it better?



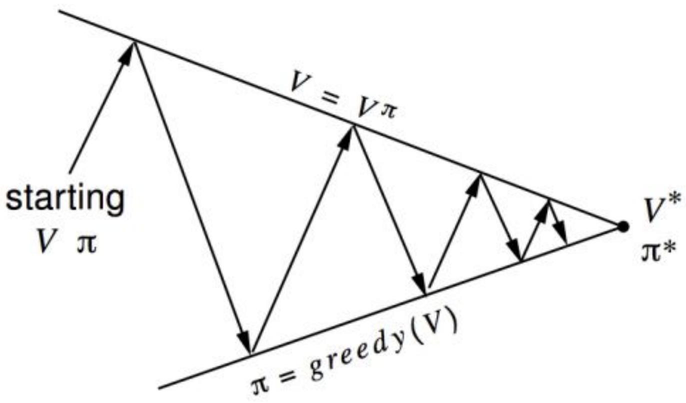
Training Pipeline: Review: Iterative Policy Evaluation

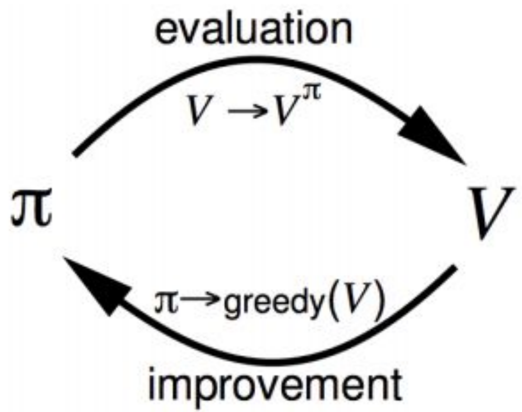
1. Policy Iteration:

a. **Policy Evaluation**: Estimate value following policy $V = V_\pi$

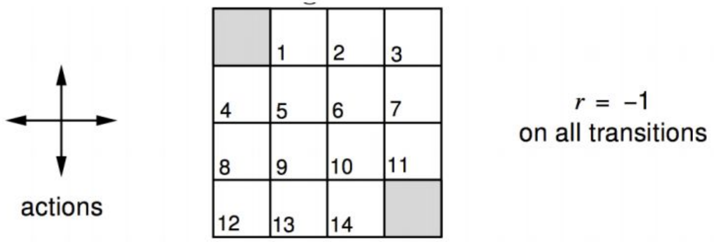
b. **Policy Improvement**: Generate new policy $\pi' \geq \pi$

c. **Repeat**: Continue until convergence V^*, π^*





Training Pipeline : Review: Iterative Policy Evaluation



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \ldots, 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

*These slides are horribly ripped off from David Silver's excellent "Introduction To Reinforcement Learning" course https://www.youtube.com/playlist?list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ

Training Pipeline : Review: Bellman Equation

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

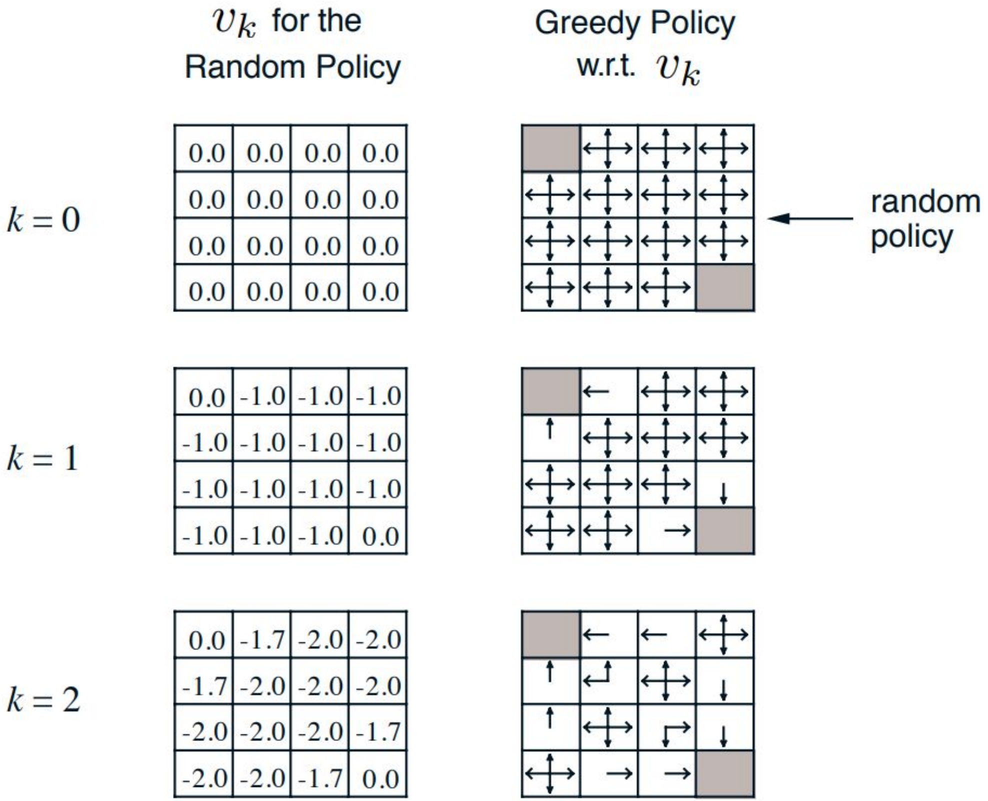
Definition

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}$$

Training Pipeline : Review: Iterative Policy Evaluation



Training Pipeline : Review: Iterative Policy Evaluation

$k = 3$

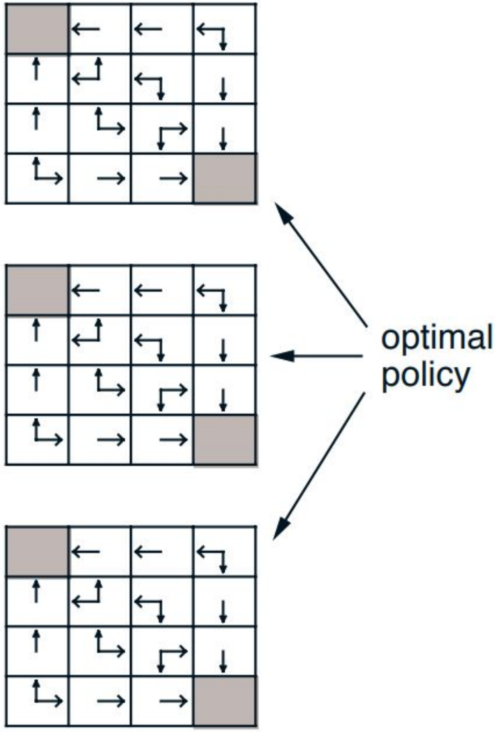
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Training Pipeline

Training is comprised of three components:

1. **Optimization:** Neural Network optimized on recent self-play data
2. **Evaluator:** AlphaGo players are evaluated against current best player
3. **Self-play:** Best current player is used to generate self-play data

Training Pipeline

1. Policy Iteration:

a. **Policy Evaluation**: $\left(s_t, \pi_t, z_t\right)$

- MCTS takes a sampling based approach (using priors to guide search) to finding the true value of your sub-MDP then uses that information to make an improved move

b. **Policy Improvement**:

- “Policy improvement starts with a neural network policy, executes an MCTS based on that policy’s recommendations, and then projects the (much stronger) search policy back into the function space of the neural network.”

c. **Repeat:**

- Self play, train

$$f_\theta(s_t) = (p_t, v_t)$$

$$l = (z - v)^2 - \pi^T log(p) + c||\theta||^2$$

Training Pipeline

2. Evaluator:

Training Pipeline

2. Evaluator:

- ❑ Each NN checkpoint is evaluated against current best player: f_{θ_i} vs α_{θ_*}

- ❑ MCTS is used to pick moves with: $\tau \rightarrow 0$

- ❑ If new player wins > 55% of games then: $\alpha_{\theta_*} \leftarrow f_{\theta_i}$

Training Pipeline

3. <u>Self-play:</u>

Training Pipeline

3. Self-play:

- Current best player is used to generate data

- For the first 30 moves: $\tau \longrightarrow 1$

- After that: $\tau \longrightarrow 0$

$$\pi(a \mid s_0) = \frac{N(s_0,a)^{1/\tau}}{\sum_b N(s_0,b)^{1/\tau}}$$

- Dirichlet noise is added to prior probabilities in MCTS:

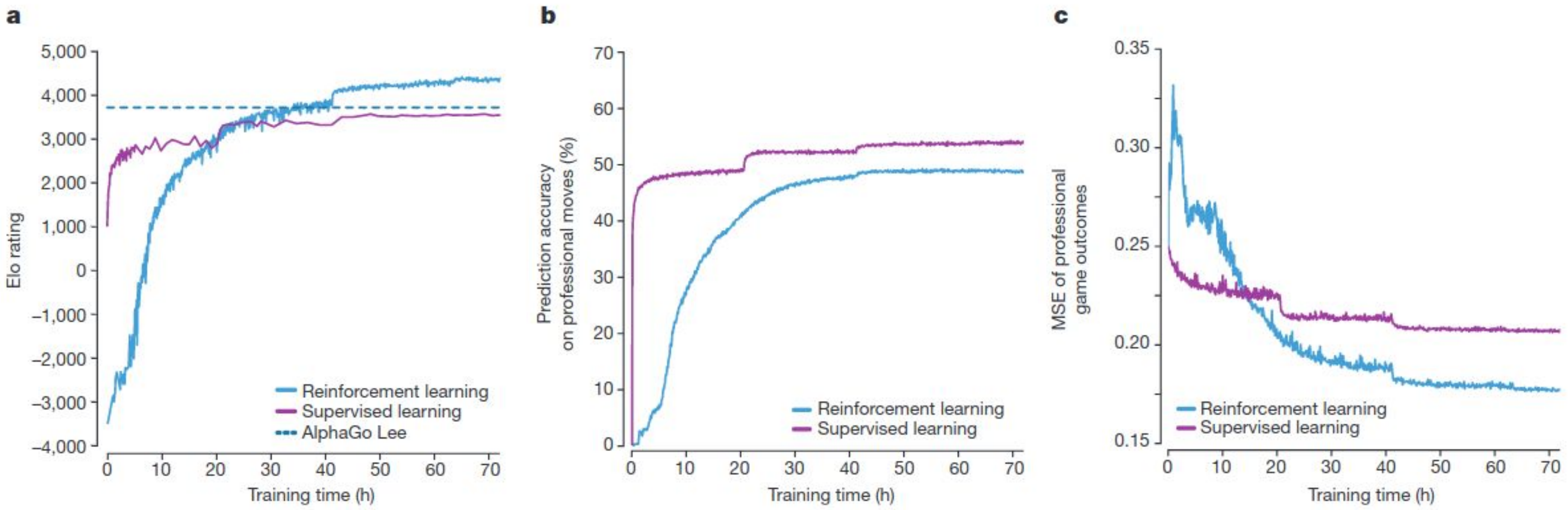
$$P(s,a) = (1-\varepsilon)p_a + \varepsilon\eta \qquad \varepsilon = 0.25, \eta \approx Dir(0.03)$$

Overview

- ❏ Cool Findings
- ❏ Neural Network Architecture
- ❏ Search Algorithm
- ❏ Training Pipeline
- ❏ Analysis

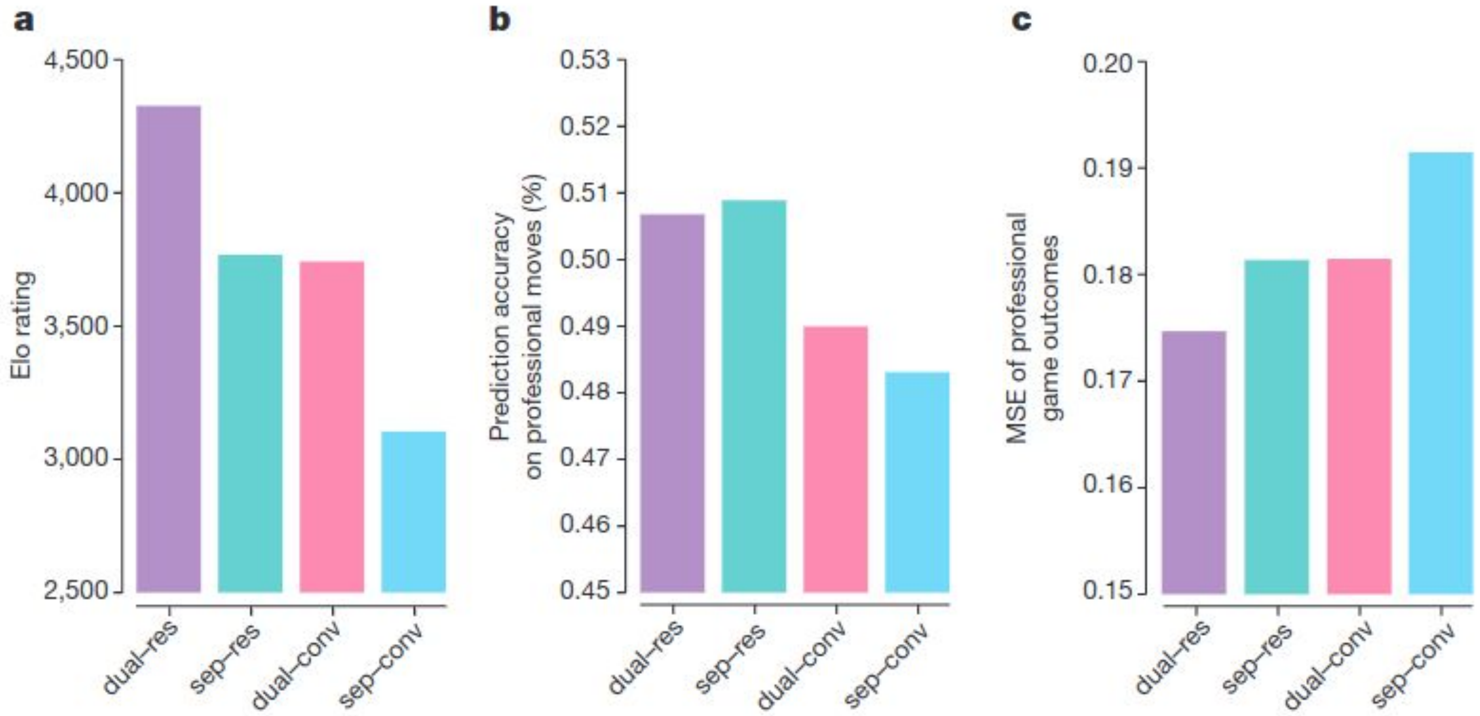
Analysis

Supervised Learning vs Self-play



Analysis

Architecture Comparison



Analysis

Performance Comparison