Mastering the Game of Go With Deep Neural Networks and Tree Search

Nabiha Asghar

27th May 2016

AlphaGo by Google DeepMind

- Go: ancient Chinese board game. Simple rules, but far more complicated than Chess
- Oct '15: defeated Fan Hui (2-dan European Go champion) 5 – 0

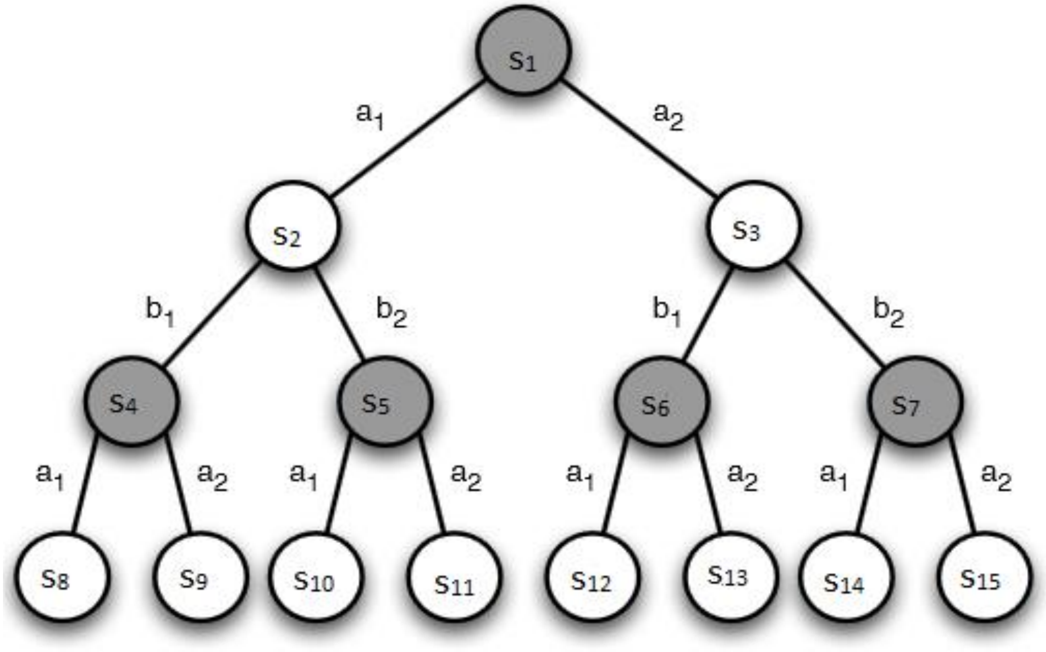
  (news delayed till January 2016 to coincide with the publication in Nature)
- Mar '16: defeated Lee Se-dol (9-dan South Korean Go player) 4 – 1
- "Last night was very gloomy… Many people drank alcohol": South Korean newspaper after Lee's first defeat

Before AlphaGo

The strongest previous Go programs were all based on Monte Carlo Tree Search (MCTS)

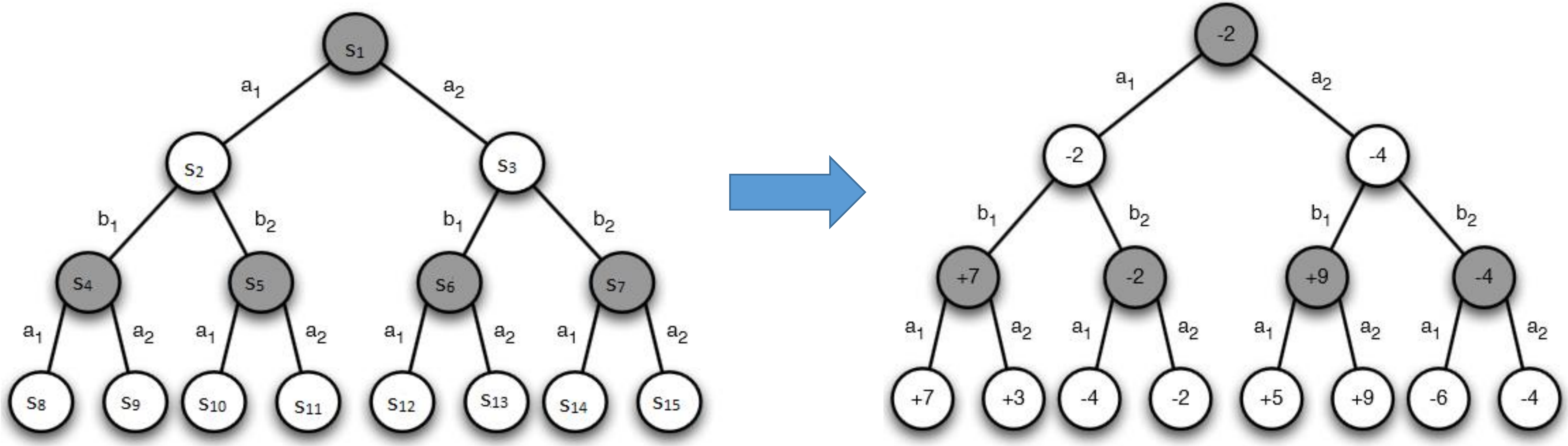
- Crazy Stone – 2006
- Mogo – 2007
- Fuego – 2010
- Pachi – 2012

Game Tree



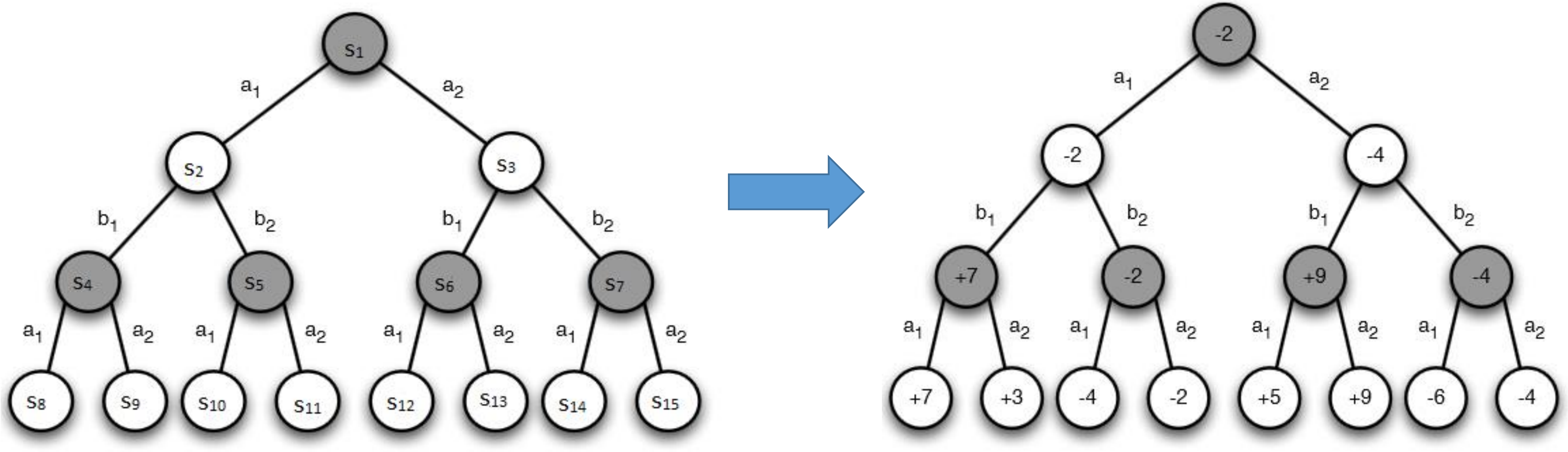
s1
a1
a2
s2
s3
b1
b2
b1
b2
s4
s5
s6
s7
a1
a2
a1
a2
a1
a2
a1
a2
s8
s9
s10
s11
s12
s13
s14
s15

Game Tree



s1
a1
a2
s2
s3
b1
b2
b1
b2
s4
s5
s6
s7
a1
a2
a1
a2
a1
a2
a1
a2
s8
s9
s10
s11
s12
s13
s14
s15
-2
a1
a2
-2
-4
b1
b2
b1
b2
+7
-2
+9
-4
a1
a2
a1
a2
a1
a2
a1
a2
+7
+3
-4
-2
+5
+9
-6
-4

Game Tree



- Optimal value of a node = best possible value the node's player can guarantee for himself
- Optimal value function: $f(node) \rightarrow optimal\ value$

Monte Carlo Simulations

Q: How do we estimate the value of a node?

Monte Carlo Simulations

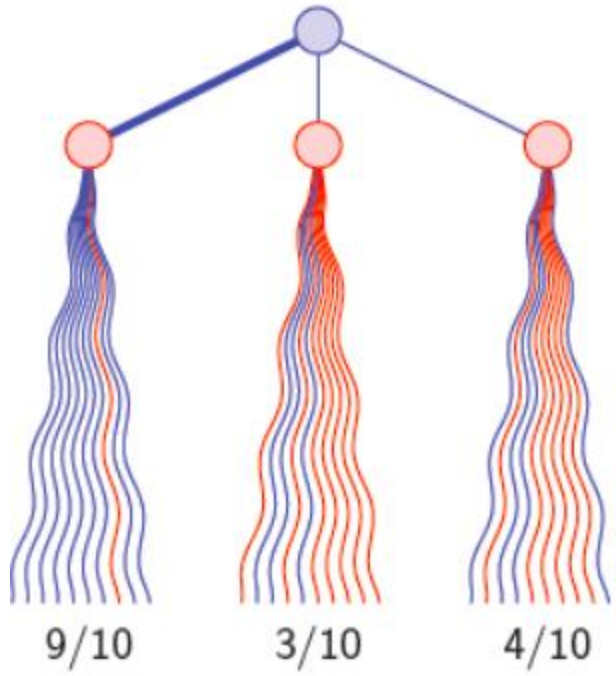
Q: How do we estimate the value of a node?

Idea:

- Run several simulations from that node by sampling actions from a policy distribution

$$a_t \sim p(a|s)$$

- Average the rewards from the simulations to obtain a Monte Carlo value estimate of the node



Monte Carlo Tree Search (MCTS)

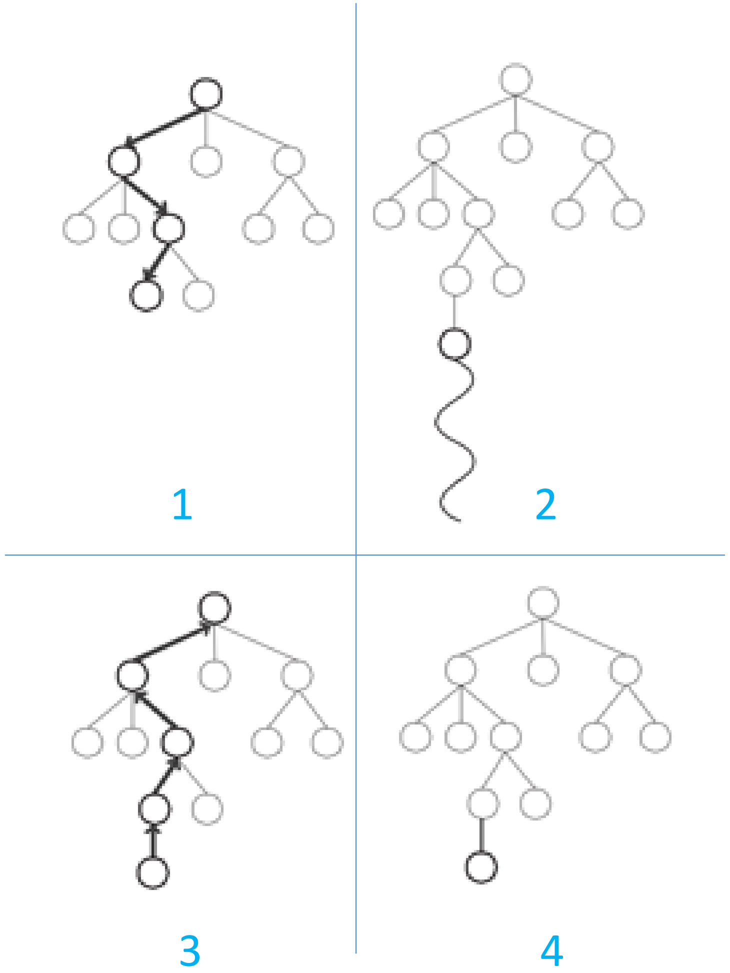
Combine Monte Carlo simulations with game tree search

1. Selection: Select the action leading to the node with highest value in the tree

2. Evaluation/Rollout: When a leaf is encountered in the tree, use a stochastic policy to select actions for both players, till the game terminates

3. Backup/Update: Update the statistics (# of visits, # of wins, prior probability) for each node of the tree visited during Selection phase

4. Growth: The first new node visited in the rollout phase is added to the tree, and its stats are initialized



MCTS: Advantages over Exhaustive Search

- The rollouts reduce the tree search breadth by sampling actions from a policy
- As more simulations are executed, the tree grows larger and the relevant values become more accurate, converging to optimal values
- The policy also improves over time (by selecting nodes with higher values), converging to optimal play

MCTS: Challenges

- Need to choose a good simulation policy that approximately chooses the optimal actions

- Need to estimate the value function based on the chosen policy

MCTS: Challenges

- In previous works, simulation policy has been chosen by training over human expert moves, or through reinforcement learning via self-play.

- Achieve superhuman performance in backgammon and scrabble, but only amateur level play in Go

- Reliance on a linear combination of input features

AlphaGo

Leverage the power of deep convolutional neural networks (CNNs) in MCTS

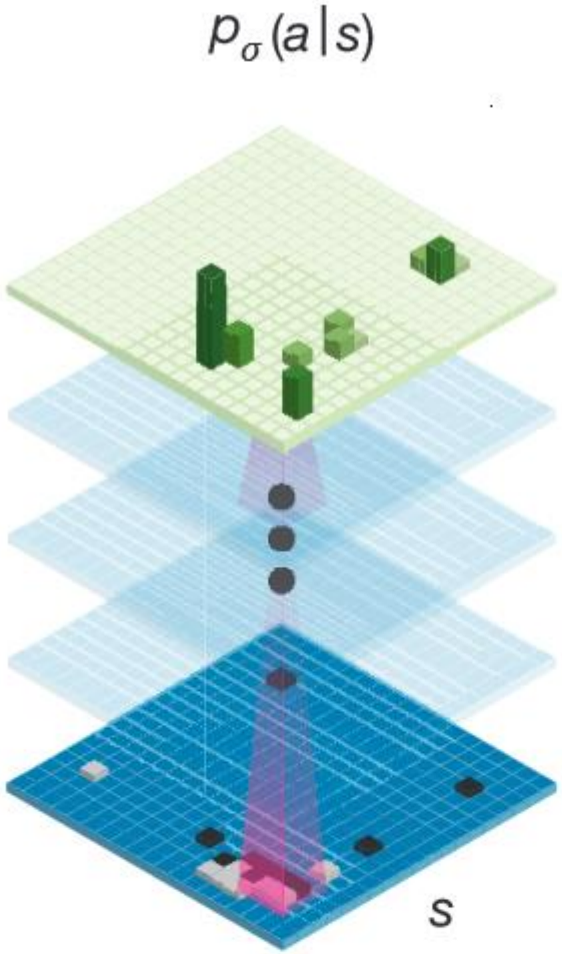
1. Policy network to compute a simulation policy $p(a|s)$
2. Value network to compute node values $v(s)$

AlphaGo Training Architecture

Main Components:

1. A Supervised Learning (SL) policy network $p_\sigma(a|s)$ (as well as a fast but less accurate rollout policy $p_\pi(a|s)$)
2. A Reinforcement Learning (RL) policy network $p_\rho(a|s)$
3. A value network $v_\theta(s)$

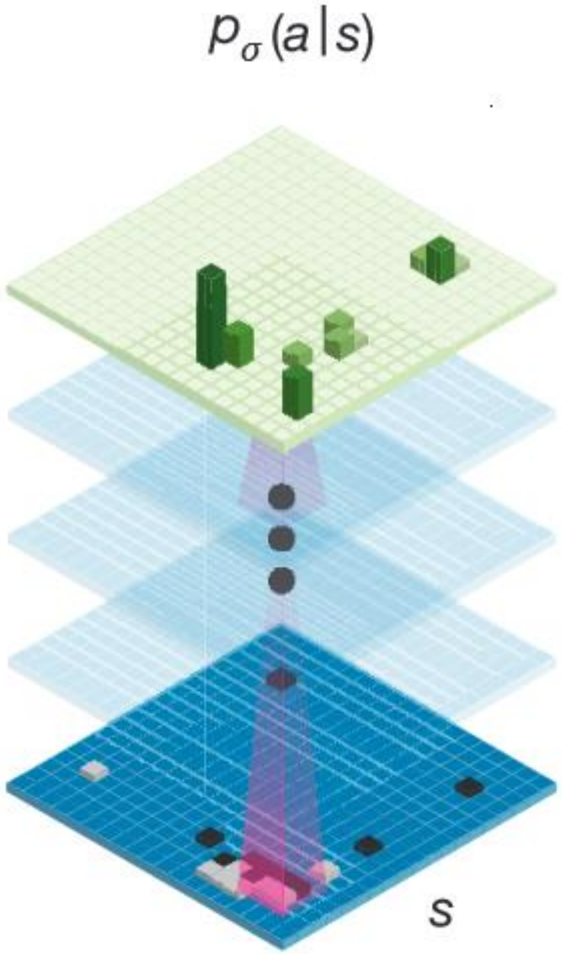
1. SL Policy Network



Goal: Predict the human expert's action at each step

Training Set: 30 million (s, a) pairs

1. SL Policy Network



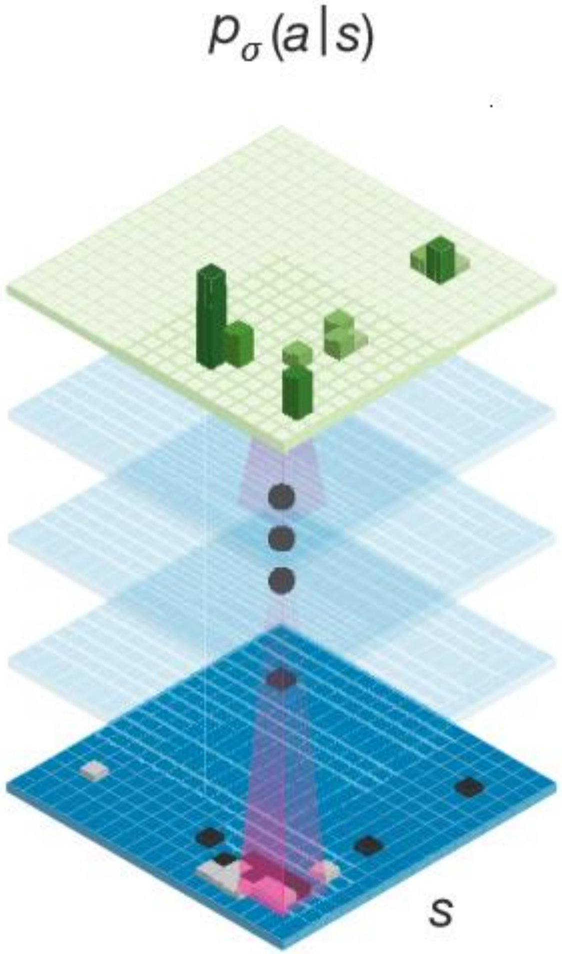
Goal: Predict the human expert's action at each step

Training Set: 30 million (s, a) pairs

Input: Simple features – stone color, #liberties, #turns, etc

Output: a probability distribution $p_\sigma(a|s)$ over all legal actions in state s

1. SL Policy Network



Goal: Predict the human expert's action at each step

Training Set: 30 million (s, a) pairs

Input: Simple features – stone color, #liberties, #turns, etc

Output: a probability distribution $p_\sigma(a|s)$ over all legal actions in state s

Architecture: 13 layers; alternating between convolutional layers with weights σ and layers containing rectifiers

Objective: Maximize the likelihood $p_\sigma(a|s)$ using stochastic gradient ascent:

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

1. Rollout Policy

- Architecture: A linear softmax of small pattern features with weights π

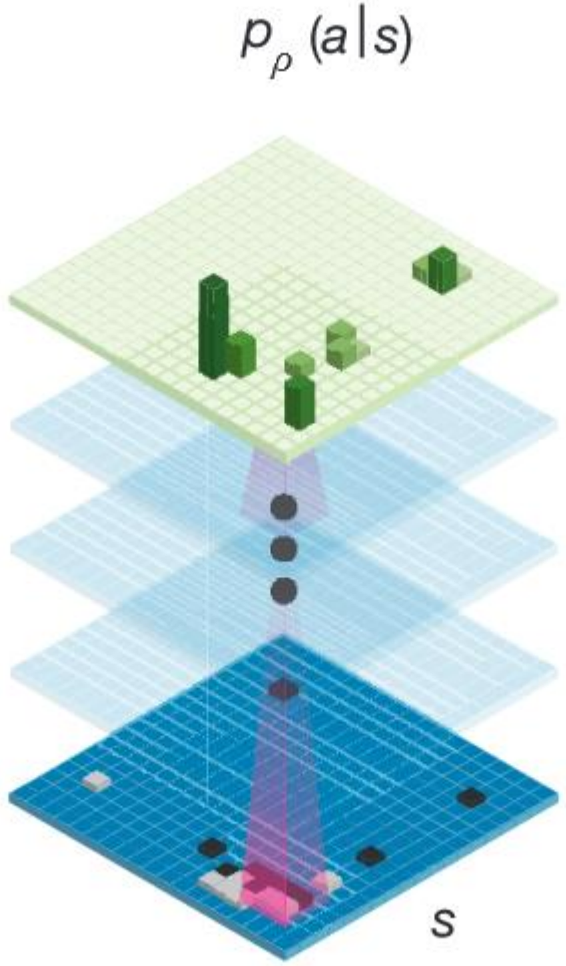
- Output: a probability distribution $p_\pi(a|s)$ over all legal actions available in state s

AlphaGo Training Architecture

Main Components:

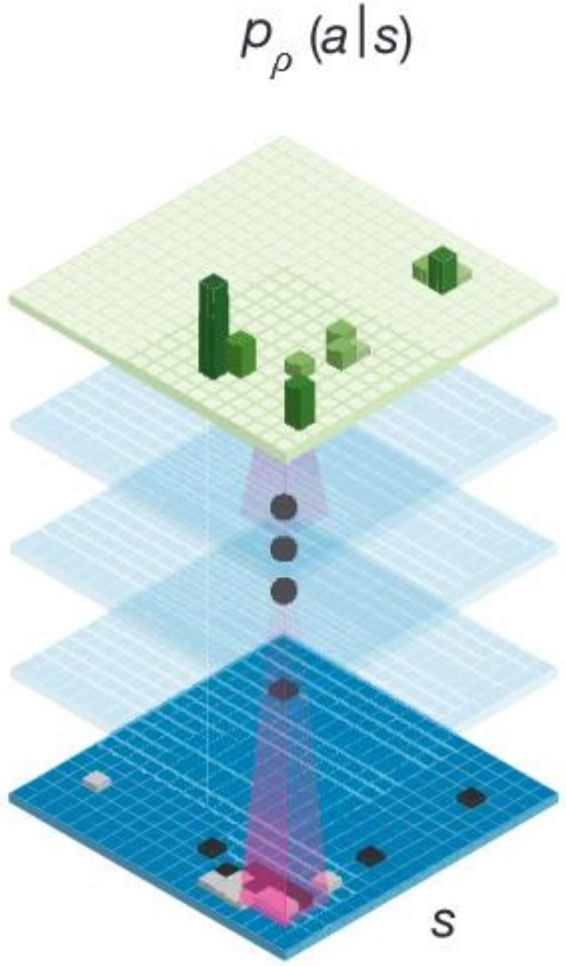
1. A Supervised Learning (SL) policy network $p_\sigma(a|s)$ (as well as a fast but less accurate rollout policy $p_\pi(a|s)$)
2. A Reinforcement Learning (RL) policy network $p_\rho(a|s)$
3. A value network $v_\theta(s)$

2. RL Policy Network



Structure: Same as SL policy network, with weights ρ initialized to σ

2. RL Policy Network



Structure: Same as SL policy network, with weights ρ initialized to σ

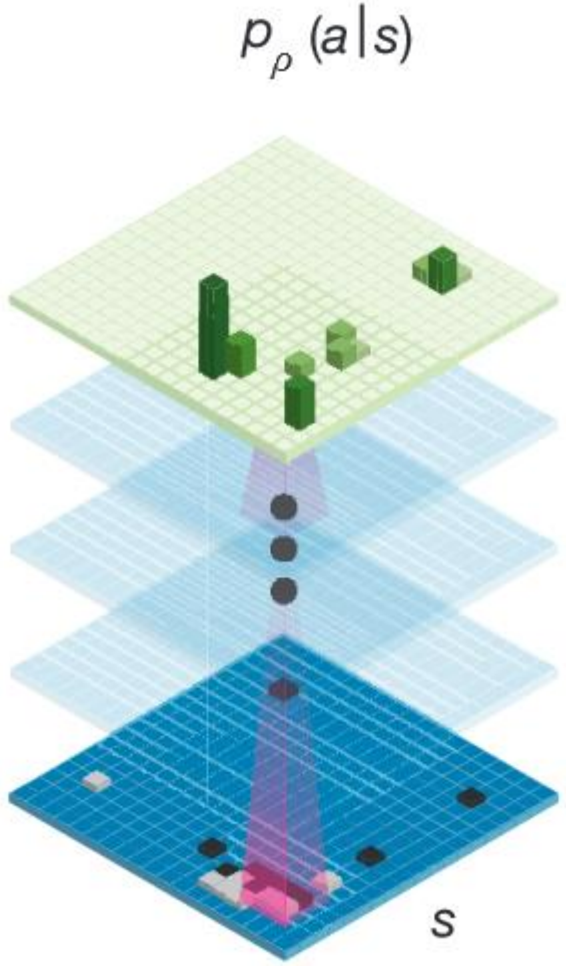
Goal: Improve the SL policy network through reinforcement learning

2. RL Policy Network

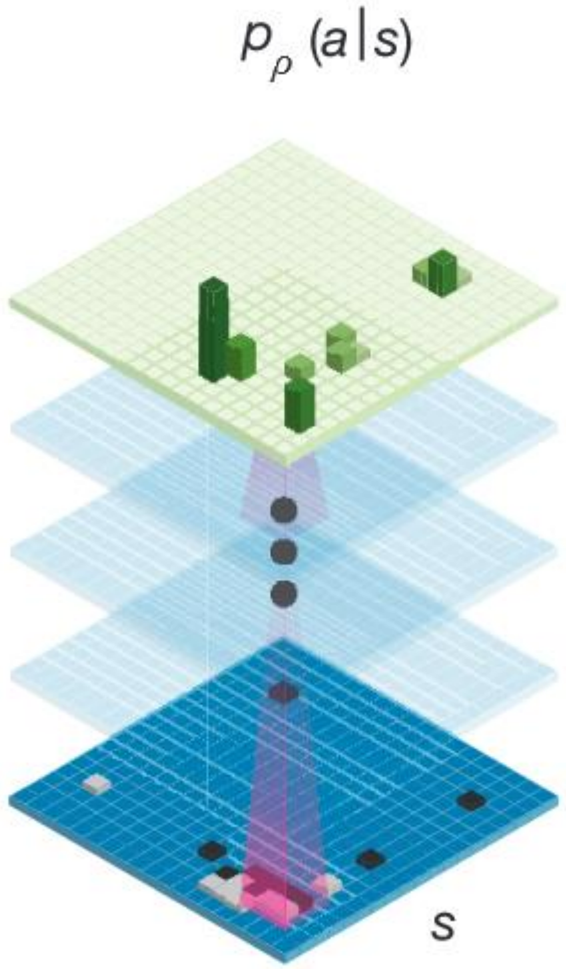
Structure: Same as SL policy network, with weights ρ initialized to σ

Goal: Improve the SL policy network through reinforcement learning

Output: a probability distribution $p_\rho(a|s)$ over all legal actions available in state s



2. RL Policy Network



Structure: Same as SL policy network, with weights ρ initialized to σ

Goal: Improve the SL policy network through reinforcement learning

Output: a probability distribution $p_\rho(a|s)$ over all legal actions available in state s

Objective: Play p_ρ against a randomly selected previous iteration. Update weights through stochastic gradient ascent to maximize expected outcome:

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

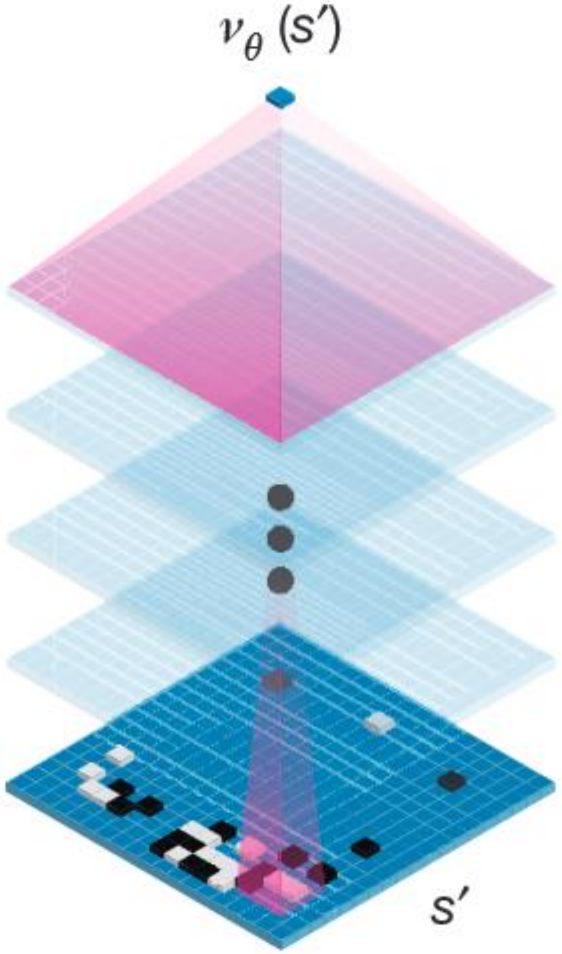
AlphaGo Training Architecture

Main Components:

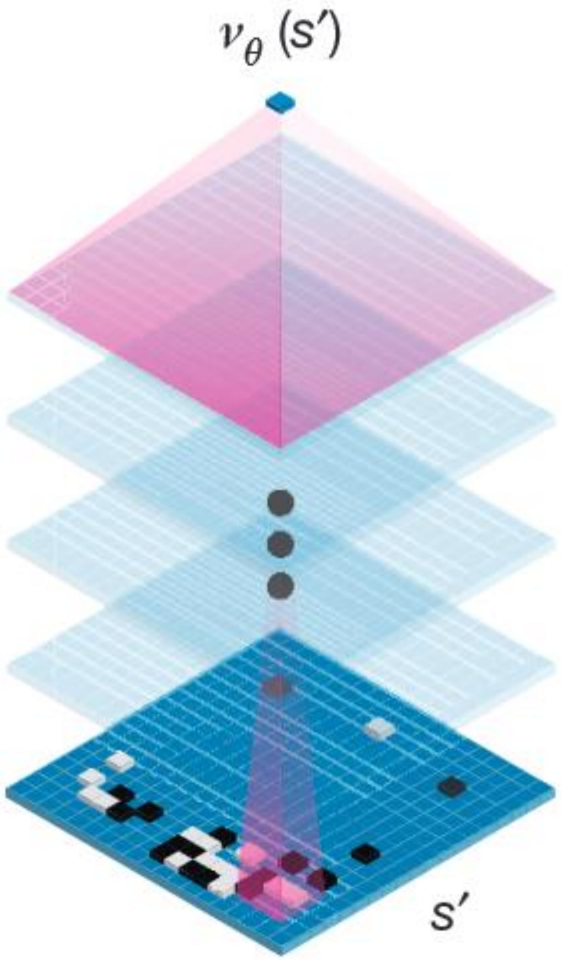
1. A Supervised Learning (SL) policy network $p_\sigma(a|s)$ (as well as a fast but less accurate rollout policy $p_\pi(a|s)$) ✓
2. A Reinforcement Learning (RL) policy network $p_\rho(a|s)$ ✓
3. A value network $v_\theta(s) \approx v^*(s)$

3. Value Network

Structure: Similar to SL/RL policy network with weights θ



3. Value Network



Structure: Similar to SL/RL policy network with weights θ

Goal: Estimate the value function $v^p(s)$ to predict outcome at state s, using policy p for both players:

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t...T} \sim p]$$

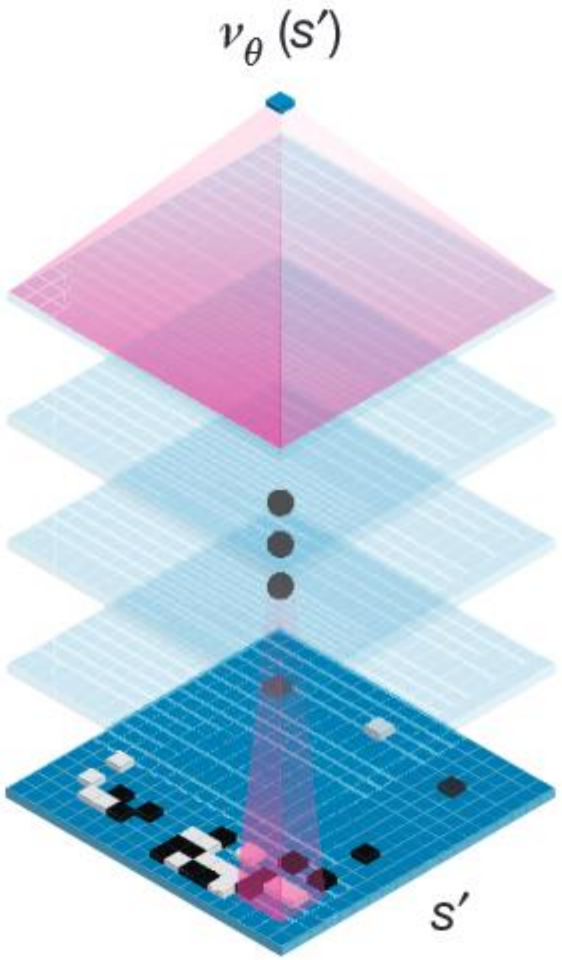
3. Value Network

Structure: Similar to SL/RL policy network with weights θ

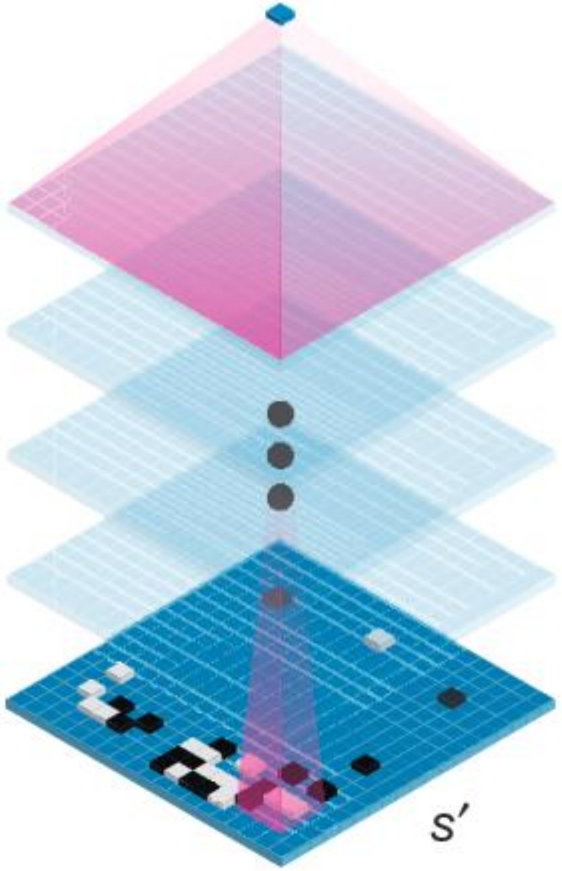
Goal: Estimate the value function $v^p(s)$ to predict outcome at state s, using policy p for both players: $v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \ldots T} \sim p]$

Data: 30 million (s, z) pairs, from games played between RL network and itself

Output: a single prediction value $v_\theta(s) \approx v^p(s) \approx v^*(s)$



3. Value Network



Structure: Similar to SL/RL policy network with weights θ

Goal: Estimate the value function $v^p(s)$ to predict outcome at state s, using policy p for both players: $v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t...T} \sim p]$

Data: 30 million (s, z) pairs, from games played between RL network and itself

Output: a single prediction value $v_\theta(s) \approx v^p(s) \approx v^*(s)$

Objective: minimize MSE between $v_\theta(s)$ and outcome z through SGD:

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

AlphaGo Training Architecture

Main Components:

1. A Supervised Learning (SL) policy network $p_\sigma(a|s)$ (as well as a fast but less accurate rollout policy $p_\pi(a|s)$)
2. A Reinforcement Learning (RL) policy network $p_\rho(a|s)$
3. A value network $v_\theta(s) \approx v^*(s)$

AlphaGo Training Architecture

Main Components:

1. A Supervised Learning (SL) policy network $p_\sigma(a|s)$ (as well as a fast but less accurate rollout policy $p_\pi(a|s)$) ✓ 50 GPUs, 3 weeks
2. A Reinforcement Learning (RL) policy network $p_\rho(a|s)$ ✓ 50 GPUs, 1 day
3. A value network $v_\theta(s) \approx v^*(s)$ ✓ 50 GPUs, 1 week

AlphaGo Training Architecture

Main Components:

1. A Supervised Learning (SL) policy network $p_\sigma(a|s)$ (as well as a fast but less accurate rollout policy $p_\pi(a|s)$) ✓ 50 GPUs, 3 weeks
2. A Reinforcement Learning (RL) policy network $p_\rho(a|s)$ ✓ 50 GPUs, 1 day
3. A value network $v_\theta(s) \approx v^*(s)$ ✓ 50 GPUs, 1 week

PUT IT ALL TOGETHER USING MCTS

SETUP: MCTS in AlphaGo

Each edge (s, a) of the search tree stores:

- $Q(s, a)$**:** the action value
- $N(s, a)$**:** visit count
- $P(s, a)$**:** prior probability
- $u(s, a)$**:** exploration bonus

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

MCTS in AlphaGo

At time step t:

1. Selection: $\boxed{a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))}$

MCTS in AlphaGo

At time step t:

1. Selection: $\boxed{a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))}$

2. Evaluation: When a leaf s_L is encountered in the tree:
   - set $P(s_L, a) := p_{\sigma/\rho}(a|s_L)$ for each edge
   - evaluate the node $\boxed{V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L}$, where z_L = outcome of a random rollout using p_π

MCTS in AlphaGo

At time step t:

1. Selection: $\boxed{a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))}$

2. Evaluation: When a leaf s_L is encountered in the tree:
   - set $P(s,a) := p_{\sigma/\rho}(a|s)$ for each edge
   - evaluate the node $\boxed{V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L}$, where z_L = outcome of a random rollout using p_π

3. Update: Update the statistics of the visited edges:

$$\boxed{\begin{aligned} N(s,a) &= \sum_{i=1}^{n} 1(s,a,i) \\ Q(s,a) &= \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i) \end{aligned}}$$

MCTS in AlphaGo

At time step t:

1. Selection: $\boxed{a_t = \operatorname*{argmax}_a (Q(s_t, a) + u(s_t, a))}$

2. Evaluation: When a leaf s_L is encountered in the tree:

   - set $P(s,a) := p_{\sigma/\rho}(a|s)$ for each edge

   - evaluate the node $\boxed{V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L}$, where z_L = outcome of a random rollout using p_π

3. Update: Update the statistics of the visited edges:

$$\boxed{\begin{aligned} N(s,a) &= \sum_{i=1}^{n} 1(s,a,i) \\ Q(s,a) &= \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i) \end{aligned}}$$

4. Growth: When $N(s,a) > threshold$ for a node s', add the node to the tree, initialize it to all zeros and set

$$\boxed{P(s',a) := p_{\sigma/\rho}(a|s')}$$

Resource Usage

Final version of AlphaGo:

- 40 search threads, 48 CPUs (for simulation)
- 8 GPUs (to compute policy and value networks)

Distributed version:

- 40 search threads, 1202 CPUs
- 176 GPUs