Lecture 18: Policy Gradient and Monte Carlo Tree Search CS486/686 Intro to Artificial Intelligence

2026-3-12

Pascal Poupart
David R. Cheriton School of Computer Science
CIFAR AI Chair at Vector Institute

UNIVERSITY OF WATERLOO

Outline

- Stochastic policy gradient
  - REINFORCE algorithm
- AlphaGo
- Monte Carlo Tree Search

Model-free Policy-based Methods

- Q-learning
  - **Model-free value-based method**
  - No explicit policy representation

- Policy gradient
  - **Model-free policy-based method**
  - No explicit value function representation

Stochastic Policy

- Consider stochastic policy $\pi_\theta(a|s) = \Pr(a|s;\theta)$ parametrized by θ.

- Finitely many discrete actions

  Softmax: $\pi_\theta(a|s) = \frac{\exp(h(s,a;\theta))}{\sum_{a'} \exp(h(s,a';\theta))}$

  where $h(s,a;\theta)$ might be **linear** in θ: $h(s,a;\theta) = \sum_i \theta_i f_i(s,a)$

  or **non-linear** in θ: $h(s,a;\theta) = neuralNet(s,a;\theta)$

- Continuous actions:

  Gaussian: $\pi_\theta(a|s) = N(a|\mu(s;\theta), \Sigma(s;\theta))$

Supervised Learning

- Consider a stochastic policy $\pi_\theta(a|s)$
- Data: state-action pairs $\{(s_1, a_1^*), (s_2, a_2^*), \dots\}$
- Maximize log likelihood of the data

$$\theta^* = argmax_\theta \sum_n \log \pi_\theta(a_n^*|s_n)$$

- Gradient update

$$\theta_{n+1} \leftarrow \theta_n + \alpha_n \nabla_\theta \log \pi_\theta(a_n^*|s_n)$$

Reinforcement Learning

- Consider a stochastic policy $\pi_\theta(a|s)$

- Data: state-action-reward triples $\{(s_1, a_1, r_1), (s_2, a_2, r_2), \dots\}$

- Maximize discounted sum of rewards

$$\theta^* = argmax_\theta \sum_n \gamma^n E_\theta[r_n|s_n, a_n]$$

- Gradient update

$$\theta_{n+1} \leftarrow \theta_n + \alpha_n \boxed{\gamma^n G_n} \nabla_\theta \log \pi_\theta(a_n|s_n)$$

where $G_n = \sum_{t=0}^{\infty} \gamma^t r_{n+t}$

Stochastic Gradient Policy Theorem

- Stochastic Gradient Policy Theorem

$$\boxed{\nabla_\theta V_\theta(s_0) \propto \sum_s \mu_\theta(s) \sum_a \nabla_\theta\, \pi_\theta(a|s)\, Q_\theta(s,a)}$$

$\mu_\theta(s)$: stationary state distribution when executing policy parametrized by θ

$Q_\theta(s,a)$: discounted sum of rewards when starting in s, executing a and following the policy parametrized by θ thereafter.

Derivation

$$\nabla_\theta V_\theta(s_0) = \nabla_\theta\left[\sum_{a_0} \pi_\theta(a_0|s_0)\, Q_\theta(s_0,a_0)\right] \qquad \forall s_0 \in S$$

$$= \sum_{a_0}[\nabla\, \pi_\theta(a_0|s_0)\, Q_\theta(s_0,a_0) + \pi_\theta(a_0|s_0)\, \nabla Q_\theta(s_0,a_0)]$$

$$= \sum_{a_0}\left[\nabla\, \pi_\theta(a_0|s_0)\, Q_\theta(s_0,a_0) + \pi_\theta(a_0|s_0)\, \nabla \sum_{s_1,r_0} \Pr(s_1,r_0|s_0,a_0)\left(r_0 + \gamma V_\theta(s_1)\right)\right]$$

$$= \sum_{a_0}\left[\nabla\, \pi_\theta(a_0|s_0)\, Q_\theta(s_0,a_0) + \pi_\theta(a_0|s_0) \sum_{s_1} \gamma \Pr(s_1|s_0,a_0)\, \nabla V_\theta(s_1)\right]$$

$$= \sum_{a_0}[\nabla\, \pi_\theta(a_0|s_0)\, Q_\theta(s_0,a_0) + \pi_\theta(a_0|s_0) \sum_{s_1} \gamma \Pr(s_1|s_0,a_0)$$
$$\sum_{a_1}[\nabla\, \pi_\theta(a_1|s_1)\, Q_w(s_1,a_1) + \pi_\theta(a_1|s_1) \sum_{s_2} \gamma \Pr(s_2|s_1,a_1)\, \nabla V_\theta(s_2)]$$

$$= \sum_{s\in S} \sum_{n=0}^{\infty} \gamma^n \underbrace{\Pr(s_0 \to s; n, \theta)} \sum_a \nabla\, \pi_\theta(a|s)\, Q_\theta(s,a)$$

Probability of reaching s from s_0 at time step n

Since $\mu_\theta(s) \propto \sum_{n=0}^{\infty} \gamma^n \Pr(s_0 \to s; n, \theta)$ then

$$\propto \sum_s \mu_\theta(s) \sum_a \nabla_\theta\, \pi_\theta(a|s)\, Q_\theta(s,a)$$

REINFORCE: Monte Carlo Policy Gradient

- $\nabla_\theta V_\theta(s_0) = \sum_{s \in S} \sum_{n=0}^{\infty} \gamma^n \Pr(s_0 \to s; n, \theta) \sum_a \nabla_\theta \pi_\theta(a|s)\, Q_\theta(s,a)$
  $= E_\theta[\sum_{n=0}^{\infty} \gamma^n \sum_a Q_\theta(S_n, a) \nabla_\theta \pi_\theta(a|S_n)]$
  $= E_\theta\left[\sum_{n=0}^{\infty} \gamma^n \sum_a \pi_\theta(a|S_n)\, Q_\theta(S_n, a) \frac{\nabla_\theta \pi_\theta(a|S_n)}{\pi_\theta(a|S_n)}\right]$
  $= E_\theta\left[\sum_{n=0}^{\infty} \gamma^n\, Q_\theta(S_n, A_n) \frac{\nabla_\theta \pi_\theta(A_n|S_n)}{\pi_\theta(A_n|S_n)}\right]$
  $= E_\theta\left[\sum_{n=0}^{\infty} \gamma^n\, G_n \frac{\nabla_\theta \pi_\theta(A_n|S_n)}{\pi_\theta(A_n|S_n)}\right]$
  $= E_\theta[\sum_{n=0}^{\infty} \gamma^n\, G_n \nabla_\theta \log \pi_\theta(A_n|S_n)]$

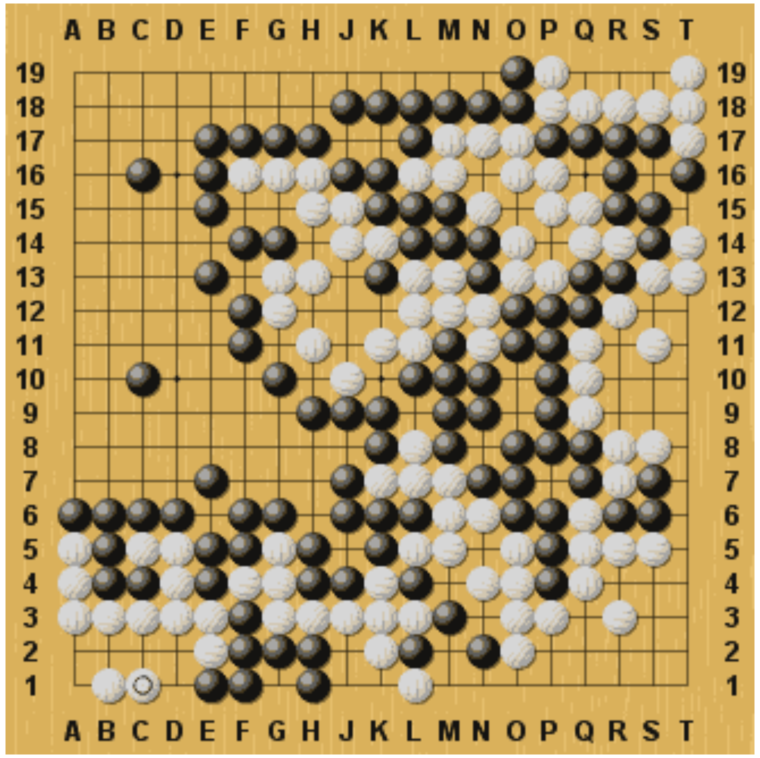
- Stochastic gradient at time step n
  $\nabla V_\theta \approx \gamma^n G_n \nabla_\theta \log \pi_\theta(a_n|s_n)$

REINFORCE Algorithm (stochastic policy)

```
REINFORCE($s_0$)
    Initialize $\pi_\theta$ to anything
    Loop forever (for each episode)
        Generate episode $s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T$ with $\pi_\theta$
        Loop for each step of the episode $n = 0, 1, \ldots, T$
            $G_n \leftarrow \sum_{t=0}^{T-n} \gamma^t r_{n+t}$
            Update policy: $\theta \leftarrow \theta + \alpha\, \gamma^n G_n \nabla_\theta \log \pi_\theta(a_n|s_n)$
    Return $\pi_\theta$
```

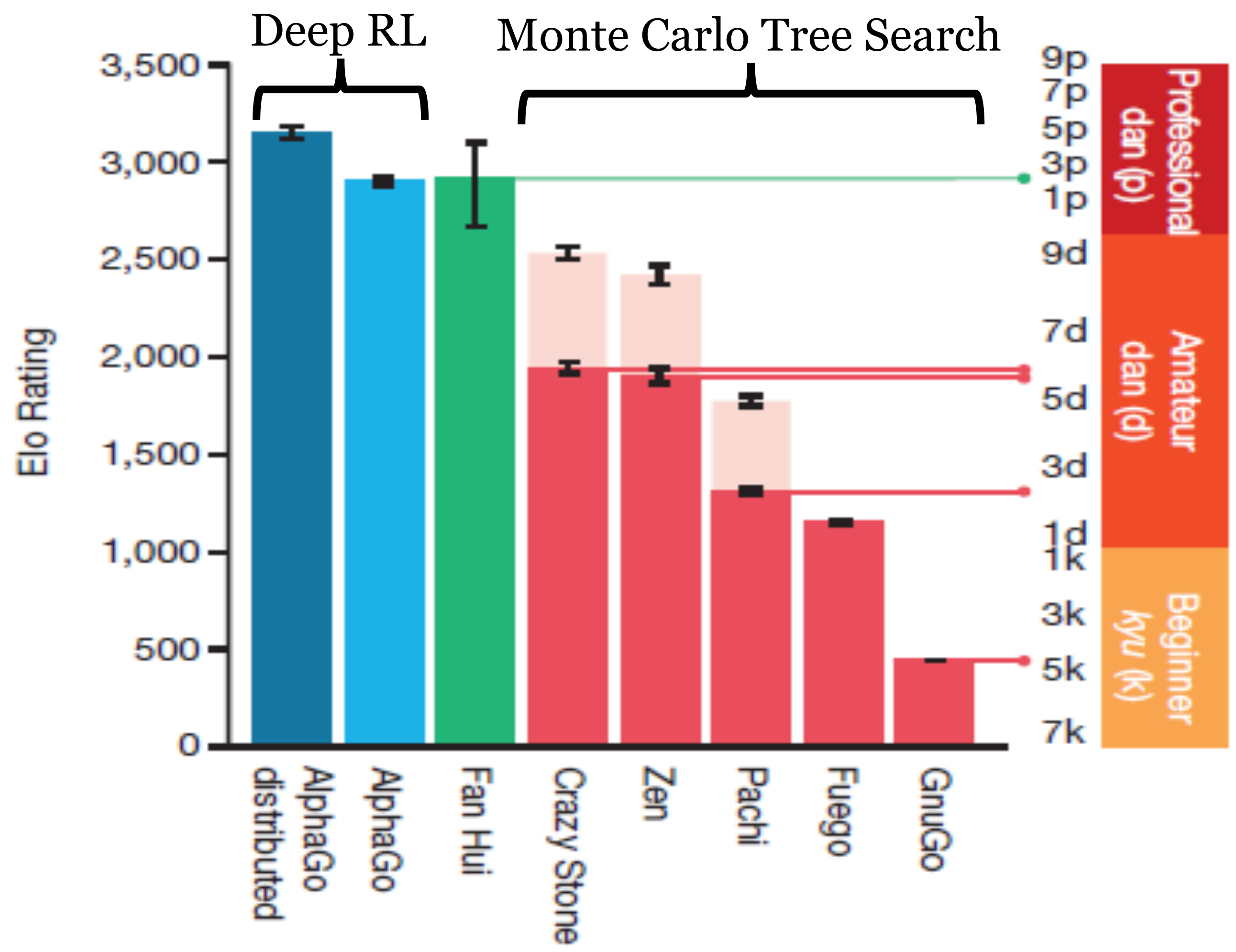
Example: Game of Go

- (simplified) rules:
  - Two players (black and white)
  - Players alternate to place a stone of their color on a vacant intersection.
  - Connected stones without any liberty (i.e., no adjacent vacant intersection) are captured and removed from the board
  - Winner: player that controls the largest number of intersections at the end of the game



Computer Go

October 2015:



Computer Go

- March 2016: AlphaGo defeats Lee Sedol (9-dan)

  “[AlphaGo] can’t beat me” Ke Jie (world champion)

- May 2017: AlphaGo defeats Ke Jie (world champion)

  “Last year, [AlphaGo] was still quite humanlike when it played.
  But this year, it became like a god of Go” Ke Jie (world champion)

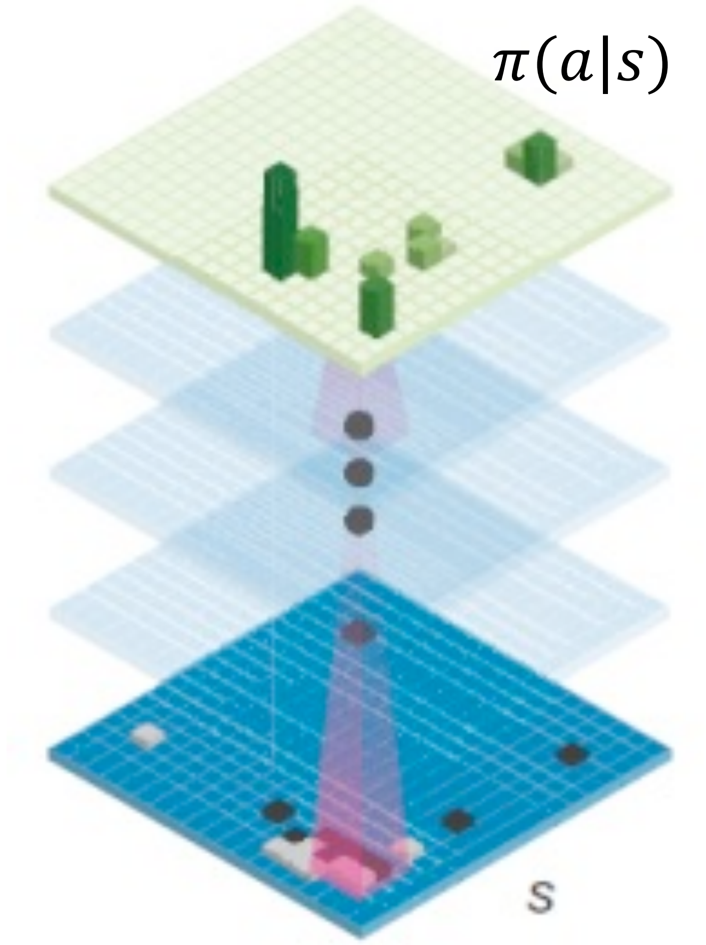
Winning Strategy

Four steps:

1. Supervised Learning of Policy Networks
2. Policy gradient with Policy Networks
3. Value gradient with Value Networks
4. Searching with Policy and Value Networks (Monte Carlo Tree Search variant)

Policy Network

- Train policy network to imitate Go experts based on a database of 30 million board configurations from the KGS Go Server.
- Policy network: $\pi(a|s)$
  - Input: state s (board configuration)
  - Output: distribution over actions a (intersection on which the next stone will be placed)



Supervised Learning of the Policy Network

- Let θ be the weights of the policy network

- Training:
  - Data: suppose a is optimal in s
  - Objective: maximize $\log \pi_\theta(\mathrm{a}|\mathrm{s})$
  - Gradient: $\nabla_\theta = \frac{\partial \log \pi_\theta(a|s)}{\partial \theta}$
  - Weight update: $\theta \leftarrow \theta + \alpha \nabla_\theta$

Policy Gradient for the Policy Network

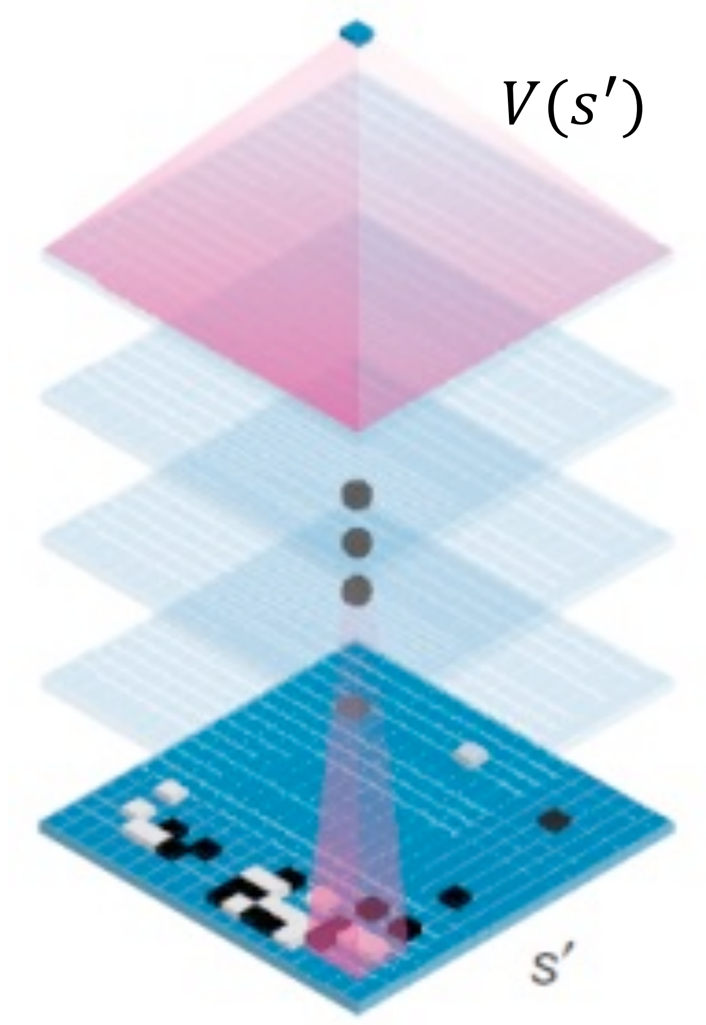
- How can we update a policy network based on reinforcements instead of the optimal action?
- Let $G_n = \sum_t \gamma^t r_{n+t}$ be the discounted sum of rewards in a trajectory that starts in s at time n by executing a.
- Gradient: $\nabla_\theta = \frac{\partial \, log \, \pi_\theta(a|s)}{\partial \theta} \gamma^n G_n$
  - Intuition: rescale supervised learning gradient by G_n
- Policy update: $\theta \leftarrow \theta + \alpha \nabla_\theta$

Policy Gradient for the Policy Network

- In computer Go, program repeatedly plays against its former self.
- For each game $G_n = \begin{cases} 1 & win \\ -1 & lose \end{cases}$
- For each (s_n, a_n) at turn n of the game, assume $\gamma = 1$ and compute
  - Gradient: $\nabla_\theta = \frac{\partial\, log\, \pi_\theta(a|s)}{\partial\theta}\, \gamma^n G_n$
  - Policy update: $\theta \leftarrow \theta + \alpha\nabla_\theta$

Value Network

- Predict $V(s')$ (i.e., who will win game) in each state s' with a value network
  - Input: state s (board configuration)
  - Output: expected discounted sum of rewards $V(s')$



Gradient Value Learning with Value Networks

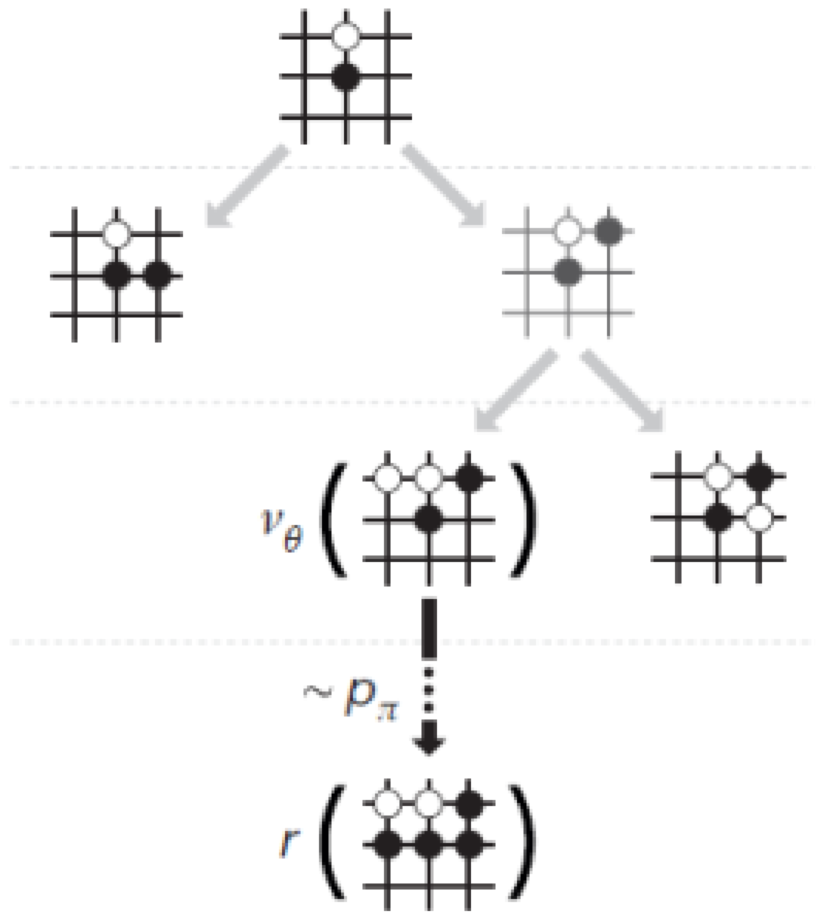
- Let $\boldsymbol{w}$ be the weights of the value network
- Training:
  - Data: (s, G) where $G = \begin{cases} 1 & win \\ -1 & lose \end{cases}$
  - Objective: minimize $\frac{1}{2}(V_{\boldsymbol{w}}(s) - G)^2$
  - Gradient: $\nabla_{\boldsymbol{w}} = \frac{\partial V_{\boldsymbol{w}}(s)}{\partial \boldsymbol{w}}(V_{\boldsymbol{w}}(s) - G)$
  - Weight update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}}$

UNIVERSITY OF WATERLOO

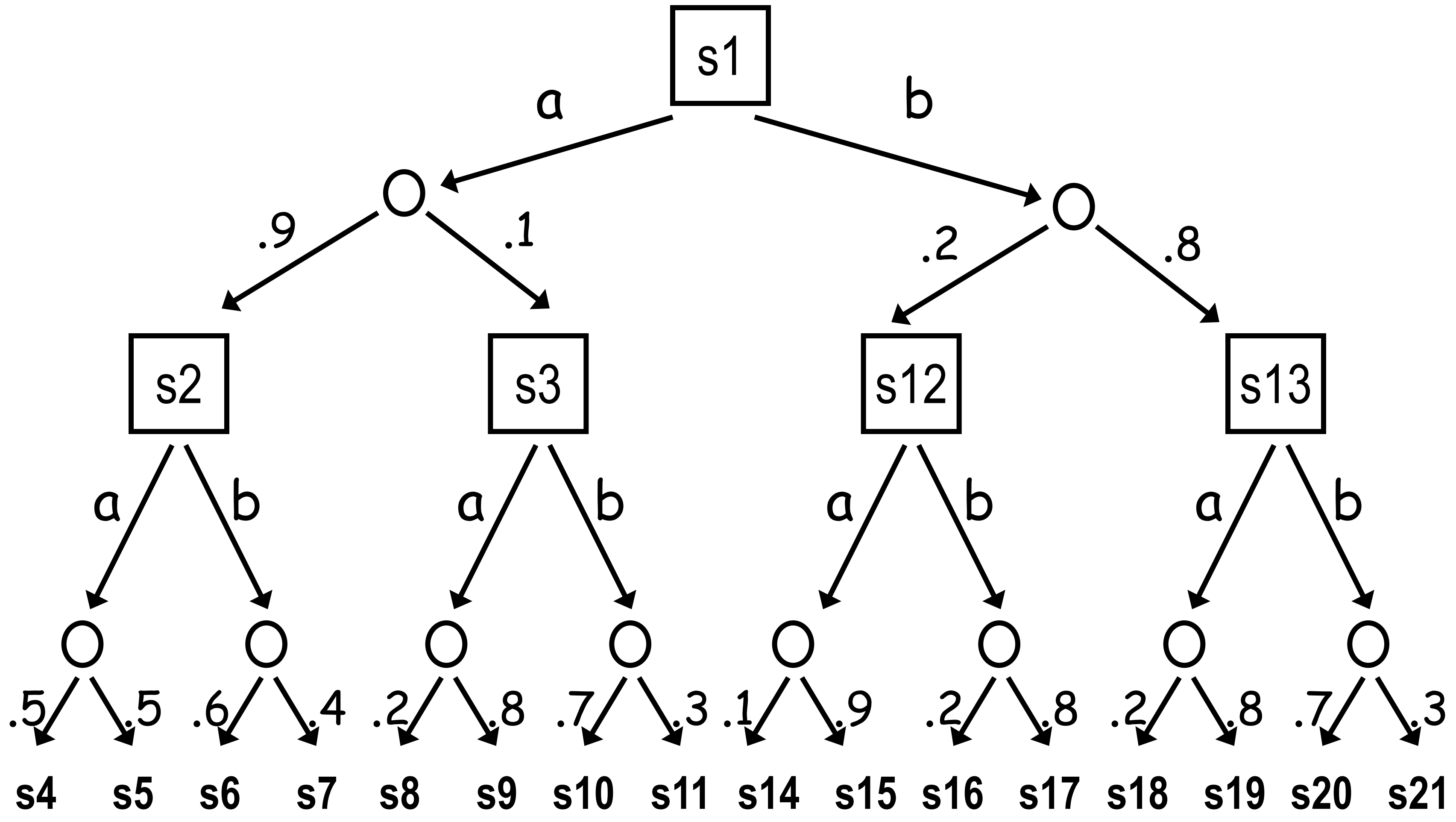
Searching with Policy and Value Networks

- AlphaGo combines policy and value networks into a **Monte Carlo Tree Search (MCTS)** algorithm

- Idea: construct a search tree
  - Node: s
  - Edge: a



Tree Search



s1
a
b
.9
.1
.2
.8
s2
s3
s12
s13
a
b
a
b
a
b
a
b
.5
.5
.6
.4
.2
.8
.7
.3
.1
.9
.2
.8
.2
.8
.7
.3
s4
s5
s6
s7
s8
s9
s10
s11
s14
s15
s16
s17
s18
s19
s20
s21

Tractable Tree Search

- Combine 3 ideas:
  - Leaf nodes: approximate leaf values with value of default policy π

$$Q^*(s,a) \approx Q^\pi(s,a) \approx \frac{1}{n(s,a)} \sum_{k=1}^{n} G_k$$

  - Chance nodes: approximate expectation by sampling from transition model

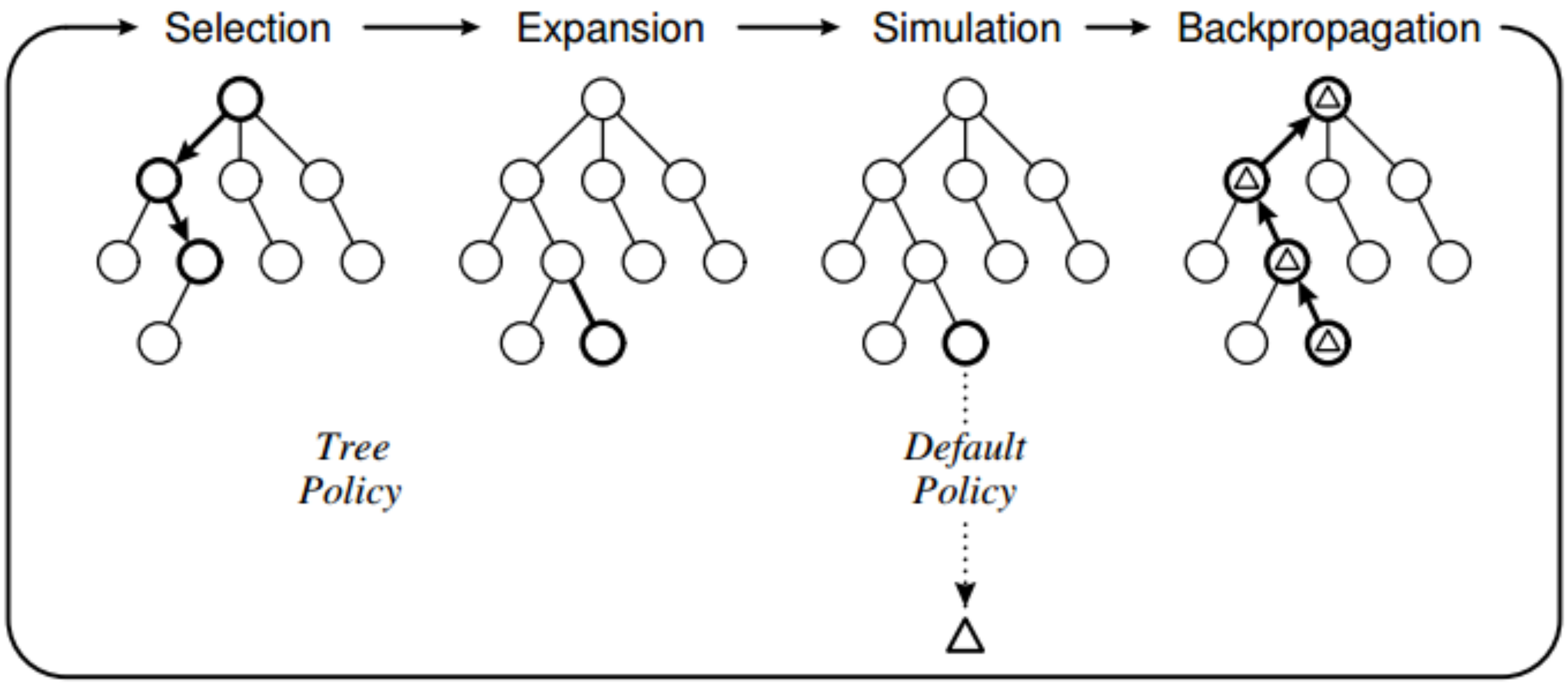
$$Q^*(s,a) \approx R(s,a) + \gamma \frac{1}{n(s,a)} \sum_{s' \sim \Pr(s'|s,a)} V(s')$$

  - Decision nodes: expand only most promising actions

$$a^* = argmax_a Q(s,a) + c\sqrt{\frac{2 \ln n(s)}{n(s,a)}} \quad \text{and} \quad V^*(s) = Q(s,a^*)$$

- Resulting algorithm: Monte Carlo Tree Search

Monte Carlo Tree Search



Monte Carlo Tree Search (with upper confidence bound)

```
UCT($s_0$)
    create root $node_0$ with state $state(node_0) \leftarrow s_0$
    while within computational budget do
        $node_l \leftarrow TreePolicy(node_0)$
        $value \leftarrow DefaultPolicy(node_l)$
        $Backpropagate(node_l, value)$
    return $action(SelectBestChild(node_0, 0))$
```

```
TreePolicy($node$)
    while $node$ is nonterminal do
        if $node$ is not fully expanded do
            return $Expand(node)$
        else
            $node \leftarrow SelectBestChild(node, C)$
    return $node$
```

Monte Carlo Tree Search (continued)

```
Expand($node$)
    choose $a \in$ untried actions of $A(state(node))$
    add a new child $node'$ to $node$
        with $state(node') \leftarrow T(state(node), a)$
    return $node'$
```

deterministic transition

```
SelectBestChild($node$,$c$)
    return $\arg\max_{node' \in children(node)} V(node') + c\sqrt{\frac{(2 \ln n(node))}{n(node')}}$
```

```
DefaultPolicy($node$)
    while $node$ is not terminal do
        sample $a \sim \pi(a|state(node))$
        $state(node') \leftarrow T(state(node), a)$
        $node \leftarrow node'$
    return $R(state(node), a)$
```

UNIVERSITY OF WATERLOO

Monte Carlo Tree Search (continued)

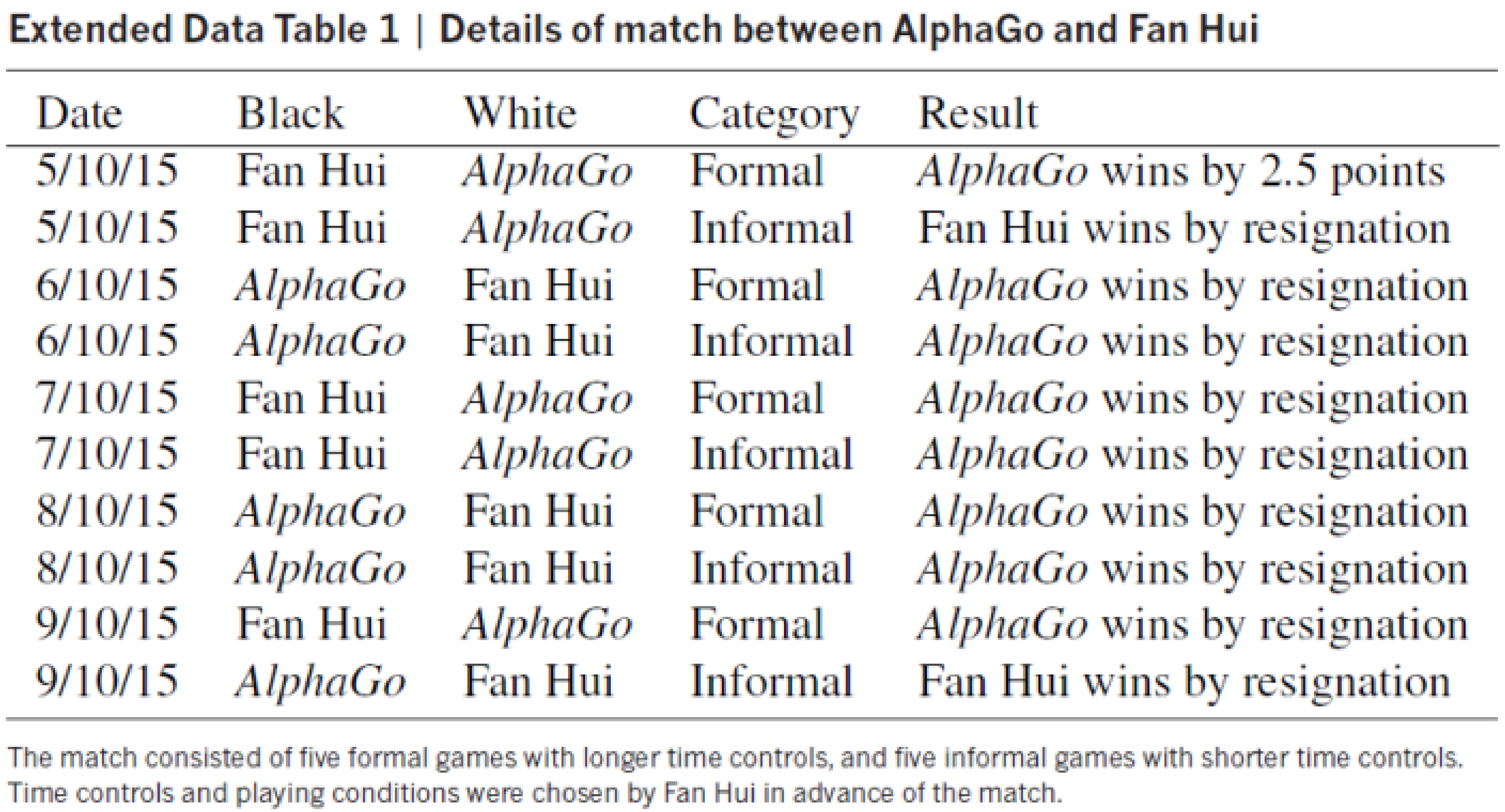
Single Player

```
Backpropagate($node$,$value$)
    while $node$ is not null do
        $V(node) \leftarrow \frac{n(node)V(node)+value}{n(node)+1}$
        $n(node) \leftarrow n(node) + 1$
        $node \leftarrow parent(node)$
```

Two Players (adversarial)

```
BackpropagateMinMax($node$,$value$)
    while $node$ is not null do
        $V(node) \leftarrow \frac{n(node)V(node)+value}{n(node)+1}$
        $n(node) \leftarrow n(node) + 1$
        $value \leftarrow -value$
        $node \leftarrow parent(node)$
```

Competition



Extended Data Table 1 | Details of match between AlphaGo and Fan Hui

Date	Black	White	Category	Result
5/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by 2.5 points
5/10/15	Fan Hui	*AlphaGo*	Informal	Fan Hui wins by resignation
6/10/15	*AlphaGo*	Fan Hui	Formal	*AlphaGo* wins by resignation
6/10/15	*AlphaGo*	Fan Hui	Informal	*AlphaGo* wins by resignation
7/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by resignation
7/10/15	Fan Hui	*AlphaGo*	Informal	*AlphaGo* wins by resignation
8/10/15	*AlphaGo*	Fan Hui	Formal	*AlphaGo* wins by resignation
8/10/15	*AlphaGo*	Fan Hui	Informal	*AlphaGo* wins by resignation
9/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by resignation
9/10/15	*AlphaGo*	Fan Hui	Informal	Fan Hui wins by resignation

The match consisted of five formal games with longer time controls, and five informal games with shorter time controls. Time controls and playing conditions were chosen by Fan Hui in advance of the match.