Lecture 14: Model-Based RL
CS486/686 Intro to Artificial Intelligence

2024-6-25

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

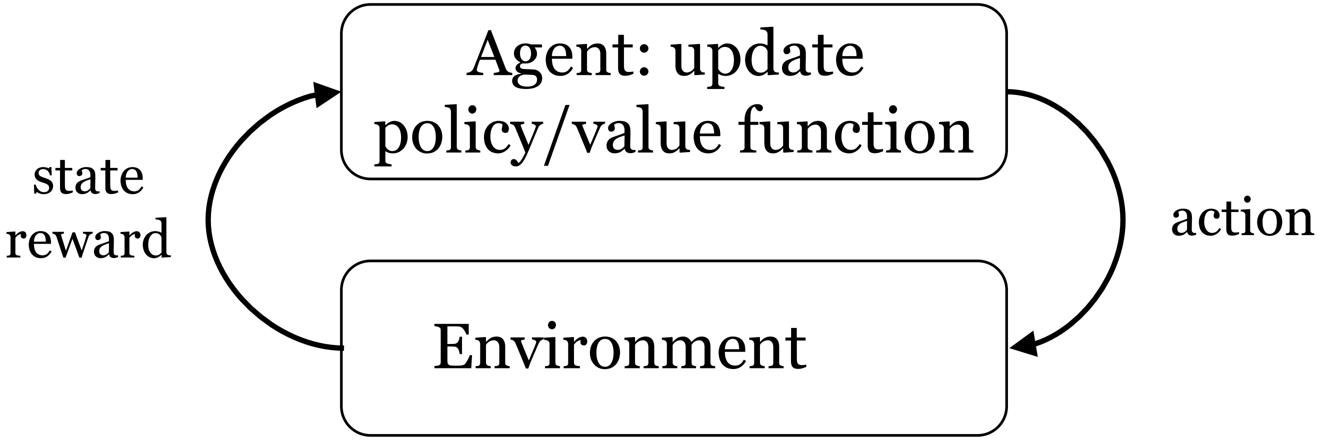
UNIVERSITY OF WATERLOO

Outline

- Model-based RL
- Dyna
- Monte-Carlo Tree Search

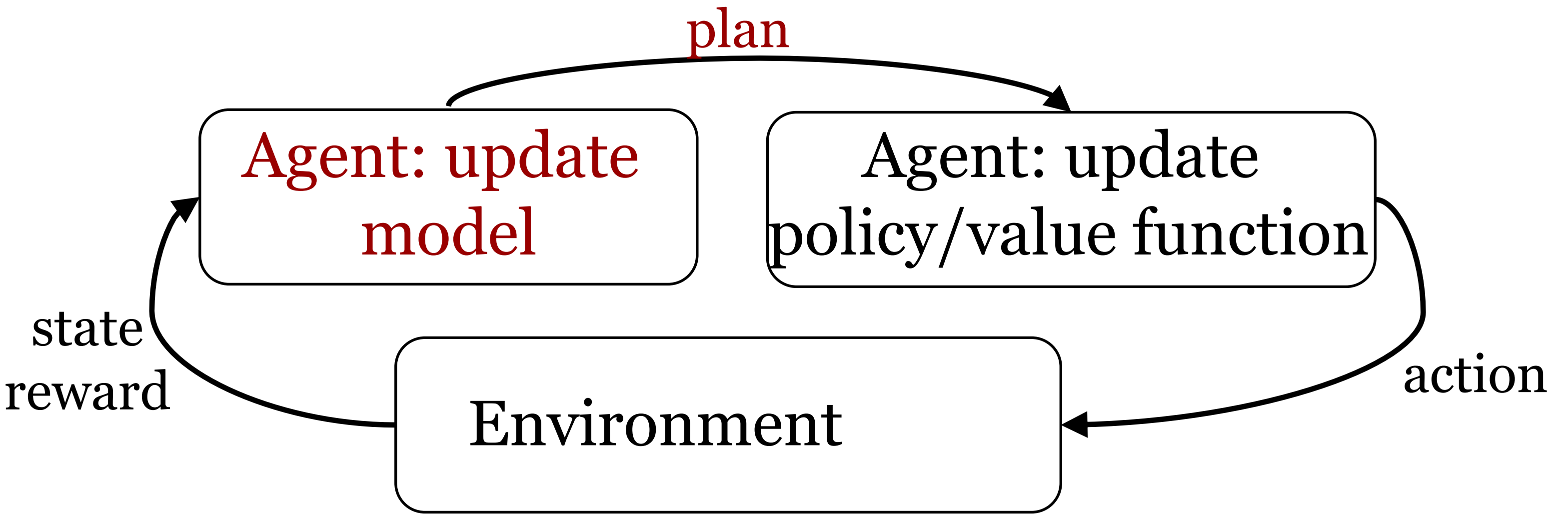
Model-free Online RL

- No explicit transition or reward models
  - Q-learning: **value-based method**
  - Policy gradient: **policy-based method**



Model-based Online RL

- Learn explicit transition and/or reward model
  - Plan based on the model
  - Benefit: Increased sample efficiency
  - Drawback: Increased complexity



Maze Example

3	r	r	r	+1
2	u		u	-1
1	u	l	l	l
	1	2	3	4

$\gamma = 1$

Reward is -0.04 for non-terminal states

We need to learn all the transition probabilities!

(1,1)→ (1,2)→ **(1,3)**→ (1,2)→ **(1,3)**→ (2,3)→ (3,3)→ $(4,3)_{+1}$
(1,1)→ (1,2)→ **(1,3)**→ (2,3)→ (3,3)→ (3,2)→ (3,3)→ $(4,3)_{+1}$
(1,1)→ (2,1)→ (3,1)→ (3,2)→ $(4,2)_{-1}$

$$\left.\begin{array}{l} P((2,3)|(1,3),r) = 2/3 \\ P((1,2)|(1,3),r) = 1/3 \end{array}\right\} \text{Use this information in}$$

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)\, V^*(s')$$

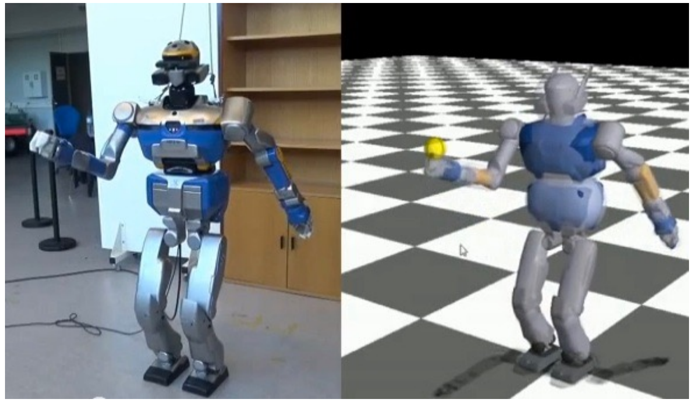
Model-based RL

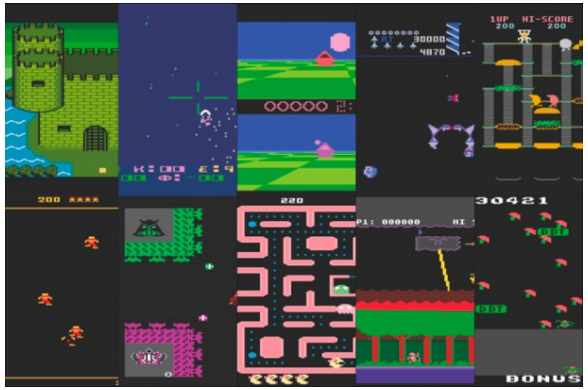
- Idea: at each step
  - Execute action
  - Observe resulting state and reward
  - Update transition and/or reward model
  - Update policy and/or value function

Model-based RL (with Value Iteration)

ModelBasedRL(s)
Repeat
Select and execute a
Observe s' and r
Update counts: $n(s,a) \leftarrow n(s,a)+1,$
$n(s,a,s') \leftarrow n(s,a,s')+1$
Update transition: $\Pr(s'|s,a) \leftarrow \frac{n(s,a,s')}{n(s,a)}\ \forall s'$
Update reward: $R(s,a) \leftarrow \frac{r+(n(s,a)-1)R(s,a)}{n(s,a)}$
Solve: $V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)\, V^*(s')\ \forall s$
$s \leftarrow s'$
Until convergence of V^*
Return V^*

Complex Models





- Use function approximation for transition and reward models
  - Linear model: $pdf(s'|s,a) = N(s'|w^T \begin{bmatrix} s \\ a \end{bmatrix}, \sigma^2 I)$
  - Non-linear models:
    - Stochastic (e.g., Gaussian process): $pdf(s'|s,a) = GP(s|w^T \begin{bmatrix} s \\ a \end{bmatrix}, \sigma^2 I)$
    - Deterministic (e.g., neural network): $s' = T(s,a) = NN(s,a)$

Partial Planning

- In complex models, fully optimizing the policy or value function at each time step is intractable

- Consider partial planning
  - A few steps of Q-learning
  - A few steps of policy gradient

Model-based RL (with Q-learning)

ModelBasedRL(s)
Repeat
Select and execute a, observe s' and r
Update transition: $w_T \leftarrow w_T - \alpha_T(T(s,a) - s')\nabla_{w_T} T(s,a)$
Update reward: $w_R \leftarrow w_R - \alpha_R(R(s,a) - r)\nabla_{w_R} R(s,a)$
Repeat a few times:
sample $\hat{s}, \hat{a}$ arbitrarily
$\delta \leftarrow R(\hat{s},\hat{a}) + \gamma \max_{\hat{a}'} Q(T(\hat{s},\hat{a}),\hat{a}') - Q(\hat{s},\hat{a})$
Update Q: $w_Q \leftarrow w_Q - \alpha_Q \delta \nabla_{w_Q} Q(\hat{s},\hat{a})$
$s \leftarrow s'$
Until convergence of Q
Return Q

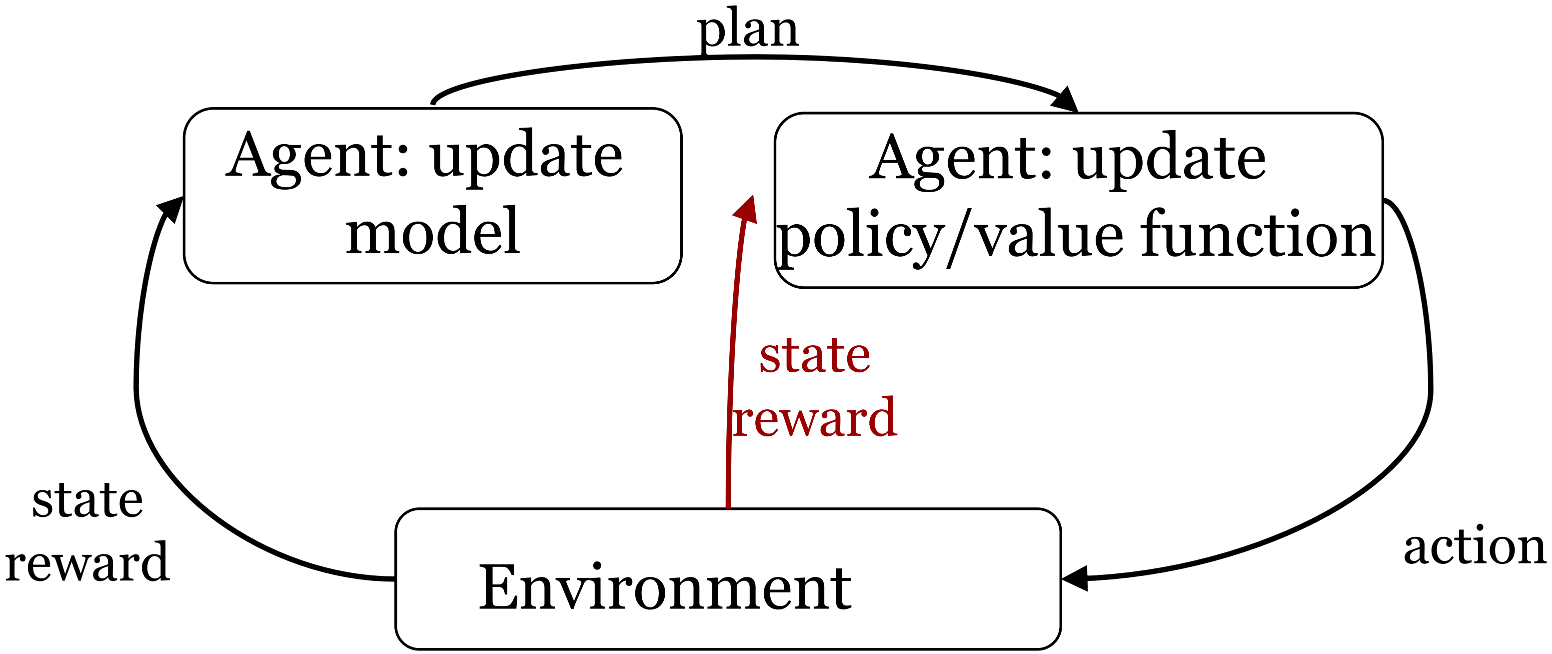
Partial Planning vs Replay Buffer

- Previous algorithm is very similar to Model-free Q-learning with a replay buffer
- Instead of updating Q-function based on samples from replay buffer, generate samples from model

- Replay buffer:
  - Simple, real samples, no generalization to other sate-action pairs
- Partial planning with a model
  - Complex, simulated samples, generalization to other state-action pairs (can help or hurt)

Dyna

- **Learn explicit transition and/or reward model**
  - Plan based on the model
- **Learn directly from real experience**



UNIVERSITY OF WATERLOO

Dyna-Q

Dyna-Q(s)

Repeat

Select and execute a, observe s' and r

Update transition: $w_T \leftarrow w_T - \alpha_T(T(s,a) - s')\nabla_{w_T} T(s,a)$

Update reward: $w_R \leftarrow w_R - \alpha_R(R(s,a) - r)\nabla_{w_R} R(s,a)$

$\delta \leftarrow r + \gamma \max_{a'} Q(s',a') - Q(s,a)$

Update Q: $w_Q \leftarrow w_Q - \alpha_Q \delta \nabla_{w_Q} Q(s,a)$

Repeat a few times:

sample $\hat{s}, \hat{a}$ arbitrarily

$\delta \leftarrow R(\hat{s},\hat{a}) + \gamma \max_{\hat{a}'} Q(T(\hat{s},\hat{a}),\hat{a}') - Q(\hat{s},\hat{a})$

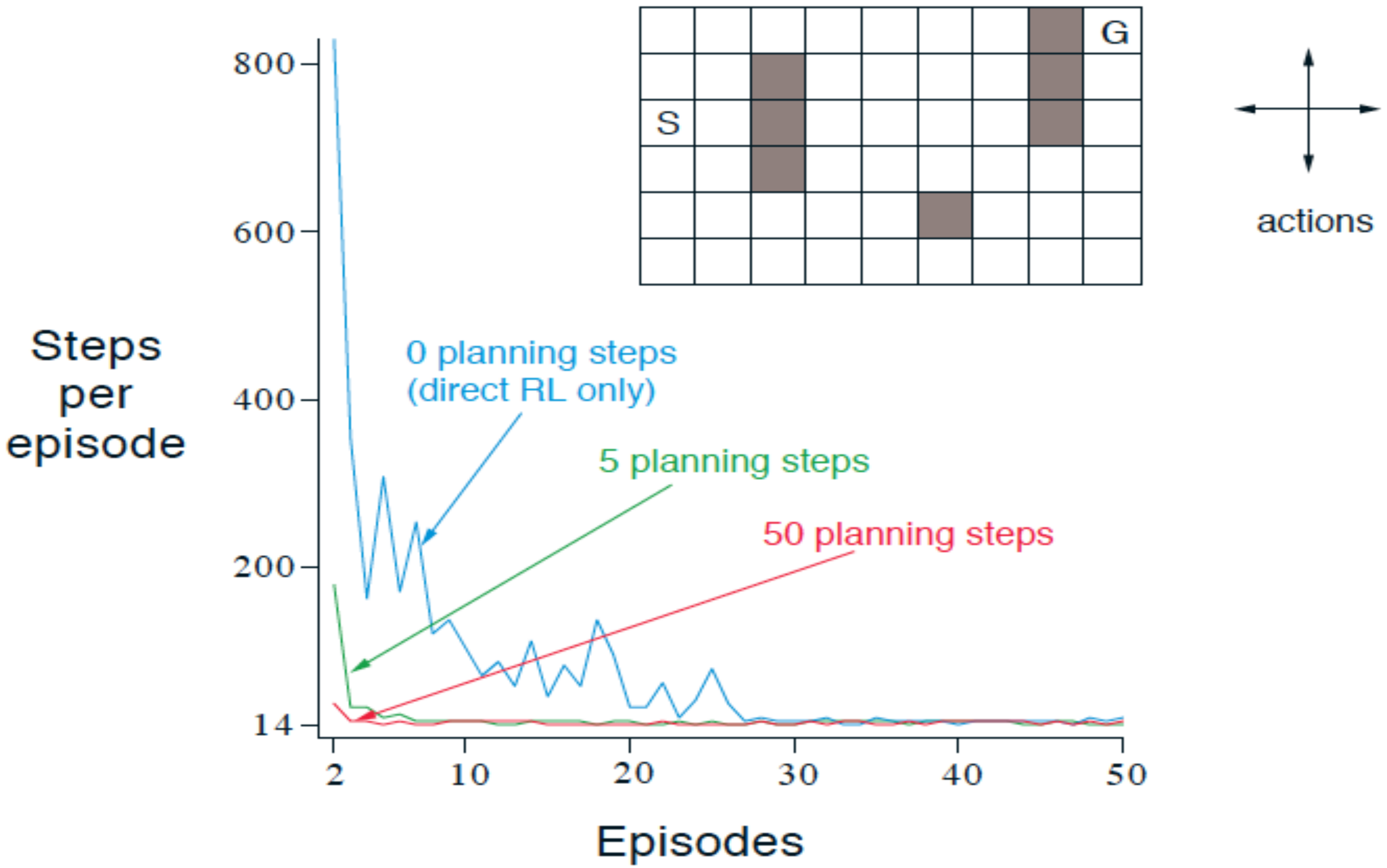
Update Q: $w_Q \leftarrow w_Q - \alpha_Q \delta \nabla_{w_Q} Q(\hat{s},\hat{a})$

$s \leftarrow s'$

Return Q

Dyna-Q

Task:
reach G
from S

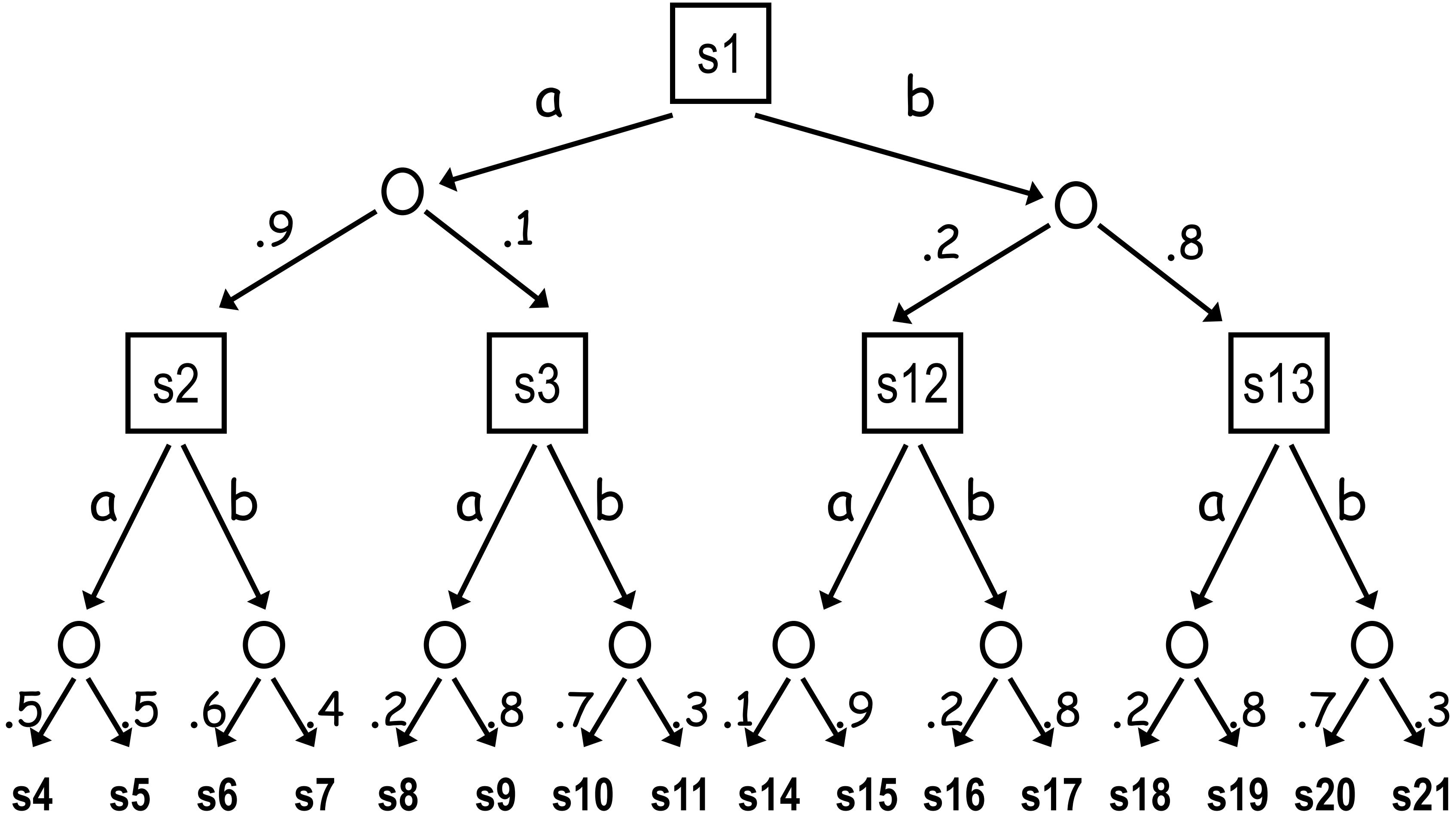


Planning from Current State

- Instead of planning at arbitrary states, plan from the current state
  - This helps improve next action

- Monte Carlo Tree Search

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

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

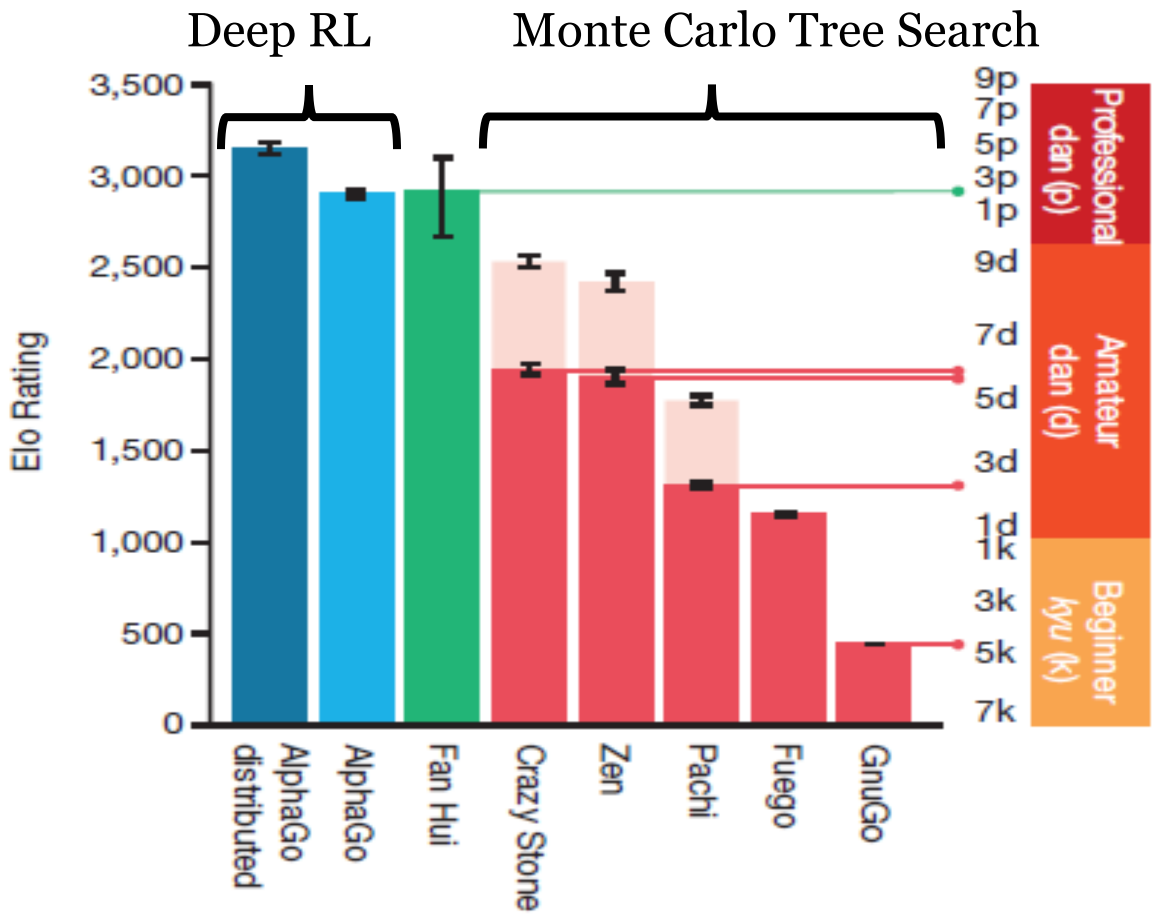
  - Decision nodes: expand only most promising actions

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

- Resulting algorithm: Monte Carlo Tree Search

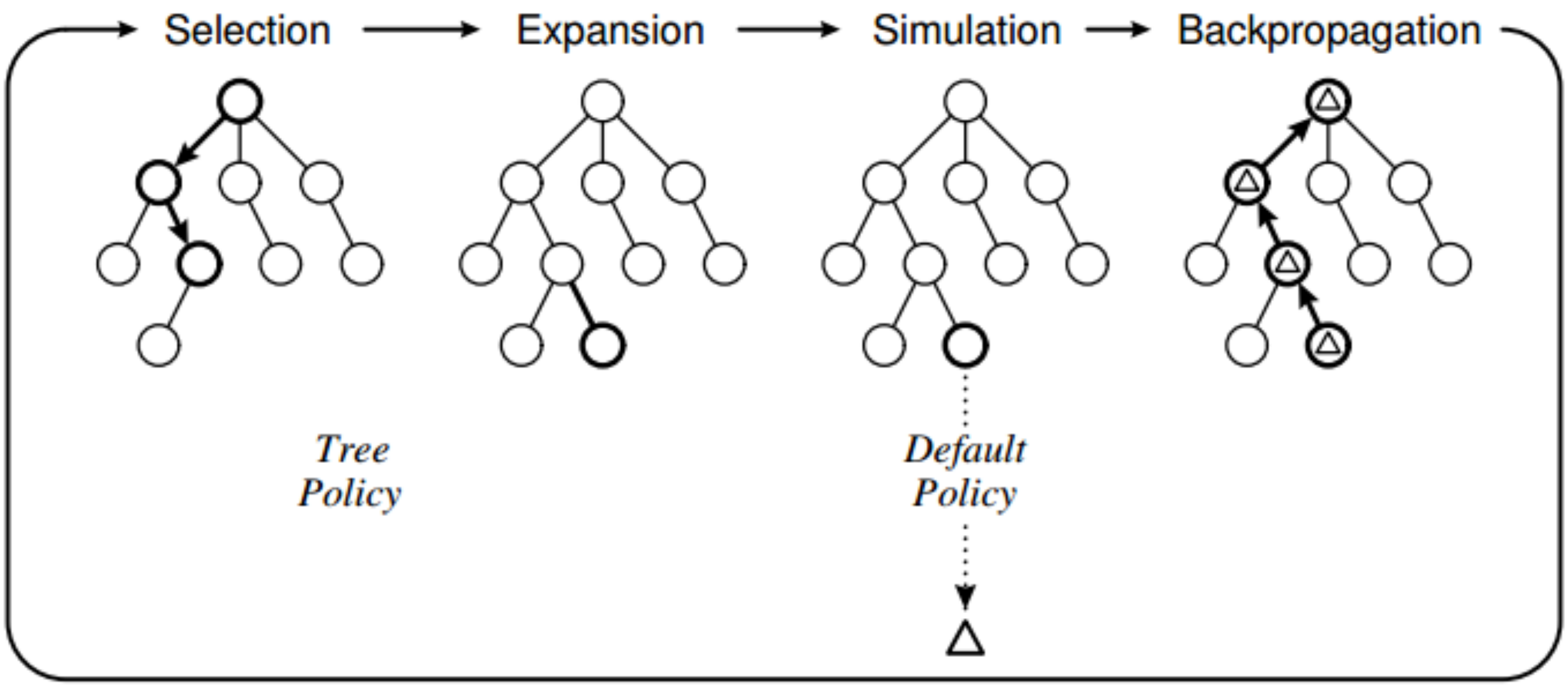
Computer Go

- Oct 2015:



UNIVERSITY OF WATERLOO

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
    $node_l \leftarrow TreePolicy(node_0)$
    $value \leftarrow DefaultPolicy\big(state(node_l)\big)$
    $Backpropagate(node_l, value)$
  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 $a \sim \pi(a|state(node))$
$s' \leftarrow T(state(node), a)$
return $R(s, a)$

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
    $V(node) \leftarrow \frac{n(node)V(node)+value}{n(node)+1}$
    $n(node) \leftarrow n(node) + 1$
    $value \leftarrow -value$
    $node \leftarrow parent(node)$

AlphaGo

Four steps:

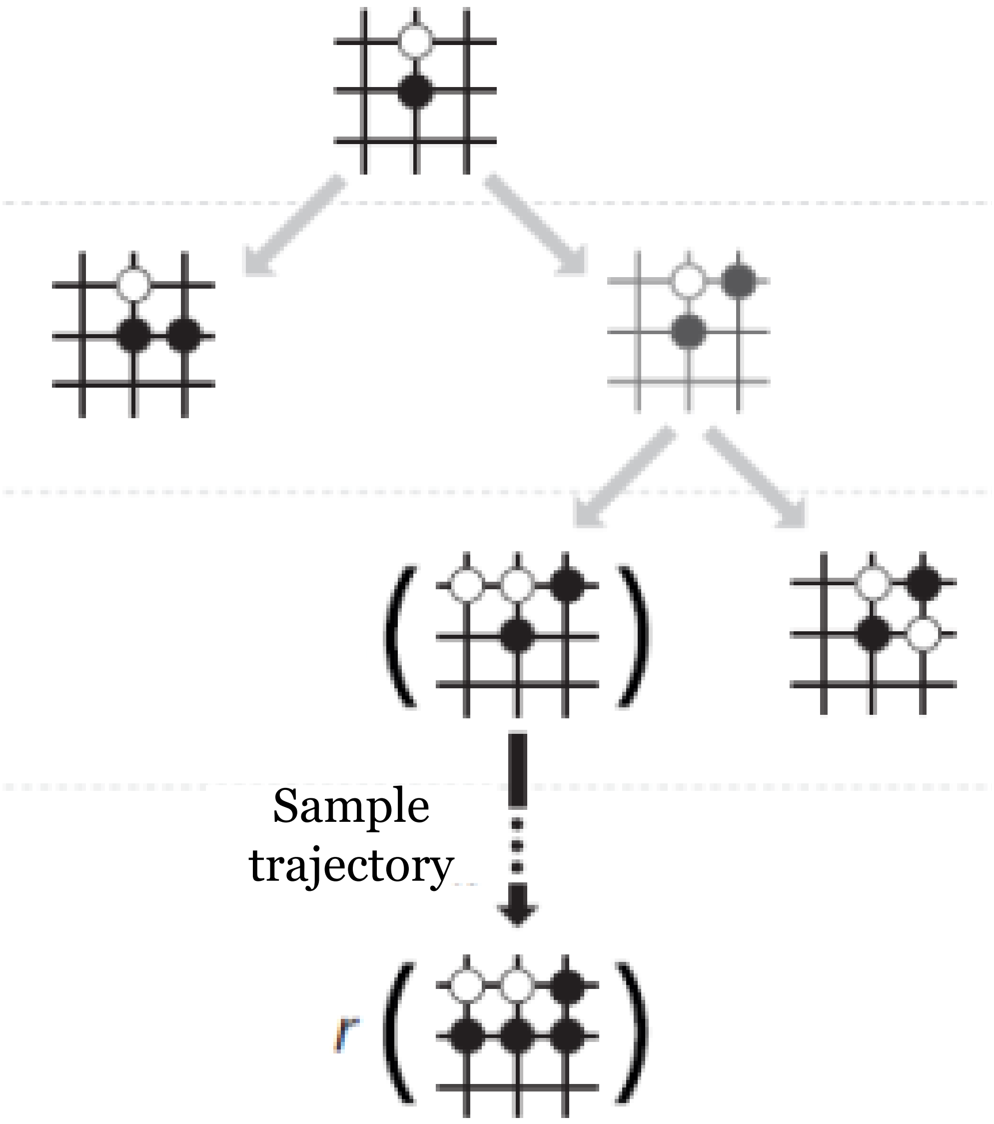
1. Supervised Learning of Policy Networks
2. Policy gradient with Policy Networks
3. Value gradient with Value Networks
4. Searching with Policy and Value Networks
    - Monte Carlo Tree Search variant

Search Tree

- At each edge store

  $Q(s,a), \pi(a|s), n(s,a)$

- Where $n(s,a)$ is the visit count of (s,a)



Simulation

- At each node, select edge a^* that maximizes

$$a^* = argmax_a\, Q(s,a) + u(s,a)$$

- where $u(s,a) \propto \frac{\pi(a|s)}{1+n(s,a)}$ is an exploration bonus

$$Q(s,a) = \frac{1}{n(s,a)} \sum_i 1_i(s,a)\,[\lambda V_{\mathbf{w}}(s) + (1-\lambda) G_i]$$

$$1_i(s,a) = \begin{cases} 1 & if\ (s,a)\ was\ visited\ at\ iteration\ i \\ 0 & otherwise \end{cases}$$