Lecture 9: Planning and Learning

Chong Li

Outline

- Monte Carlo Tree Search
- Model-based Planning
- Integrating Learning and Planning

*materials are modified from David Silver's RL lecture notes

Outline

- Monte Carlo Tree Search
  - Motivation: Computer GO
  - Upper Confidence Bound
  - MC Tree search
- Model-based Planning
- Integrating Learning and Planning

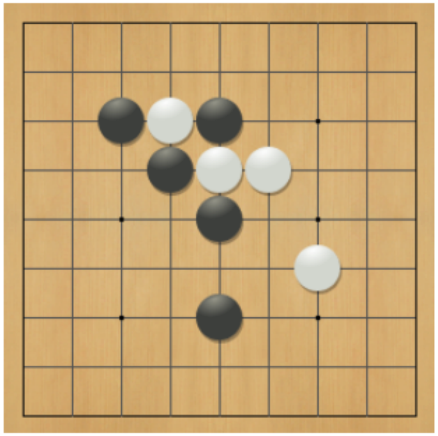
Model-Based RL Revisited

- Model is known, but curse of dimensionality
  - Each sweeping over states using DP is computationally costly.
- Game “GO” has high move and state complexity:
  - States: 10^171
- Studied for decades

- What can we do ….

Rules of GO

- Usually played on 19x19, also 13x13 or 9x9 board
- Simple rules, complex strategy
- Black and white place down stones alternately
- Surrounded stones are captured and removed
- The player with more territory wins the game



Position Evaluation in GO

- How good is a position s?
- Reward function (undiscounted):

$$R_t = 0 \text{ for all non-terminal steps } t < T$$

$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

$$v_\pi(s) = \mathbb{E}_\pi \left[R_T \mid S = s \right] = \mathbb{P}\left[\text{Black wins} \mid S = s \right]$$

$$v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

Outline

- Monte Carlo Tree Search
  - Motivation: Computer GO
  - Upper Confidence Bound
  - MC Tree search
- Model-based Planning
- Integrating Learning and Planning

Upper Confidence Bounds (UCB)

- One-step bandit problem – a row of slot machine, each with different payout probabilities and amounts.
- Fundamental tradeoff of exploration and exploitation
- What is the optimal exploration?
  - We have learnt non-optimal Ɛ-greedy and Softmax.

Definition of “Optimal”

We denote the average (or mean or expected) reward of the best action as μ^* and of any other action j as μ_j. There are a total of K actions. We write $T_j(n)$ for the number of times we have tried action j in a total of n action. Formally, the regret after n actions is defined as

$$regret(n) = \mu^* n - \sum_{j=1}^{K} \mathbb{E}[T_j(n)]$$

- The regret is the loss due to the fact that the globally optimal policy is not followed all the times
- Our goal is to minimize the regret
- Lai and Robbins showed that the regret for the multi-armed bandit problem has to grow at least logarithmically w.r.t. the number of plays n
- UCB is proved to grow logarithmically -> optimal policy

Upper Confidence Bounds 1 (UCB1)

- The simplest UCB policy is called UCB1 *

exploit

explore

$$\mathrm{UCB1} = \overline{X}_j + C\sqrt{\frac{2\ln n}{n_j}}$$

X_j is estimated reward of choice j

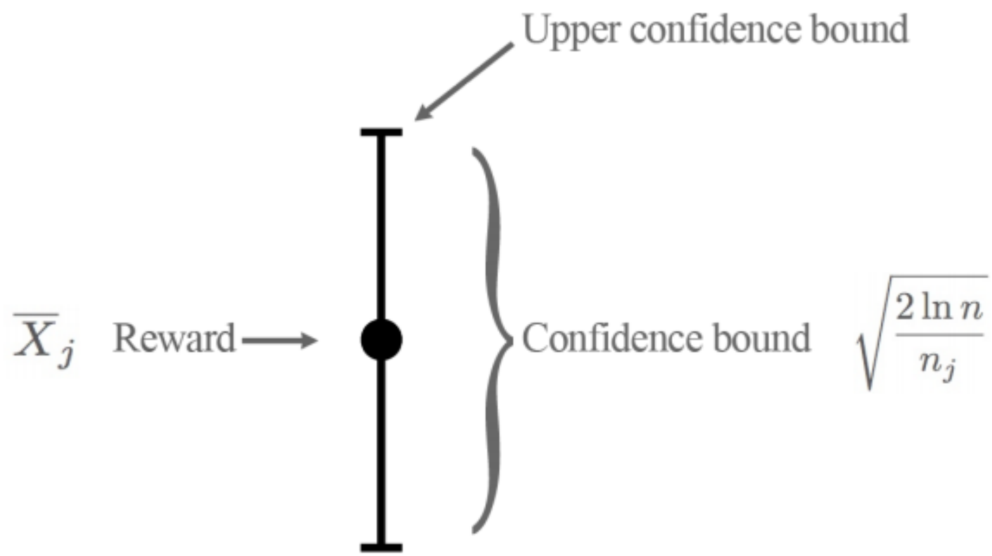
n is number of all plays done so far

n_j is number of times choice j has been tried

*P. Auer, et al. "Finite-time Analysis of the Multiarmed Bandit Problem", 2002

Upper Confidence Bounds 1 (UCB1)

Confidence in the reward's accuracy



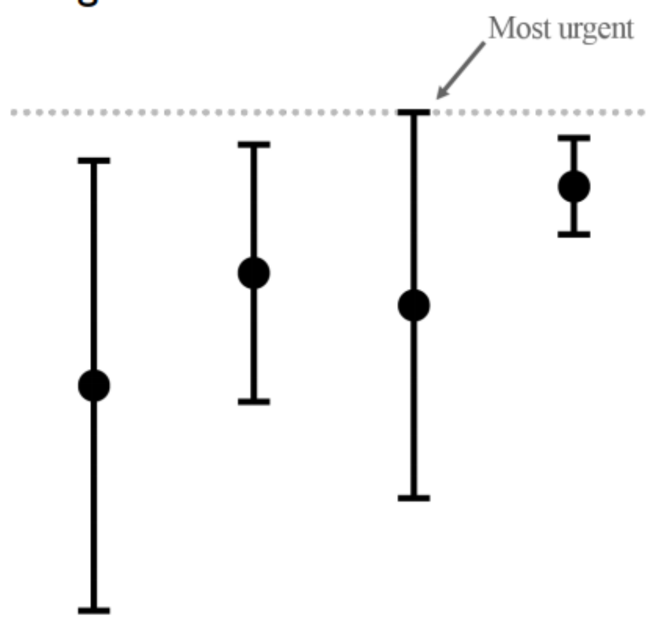
More visits = tighter bound

Upper Confidence Bounds 1 (UCB1)

Most urgent node has the highest UCB

Not highest reward

Not widest spread

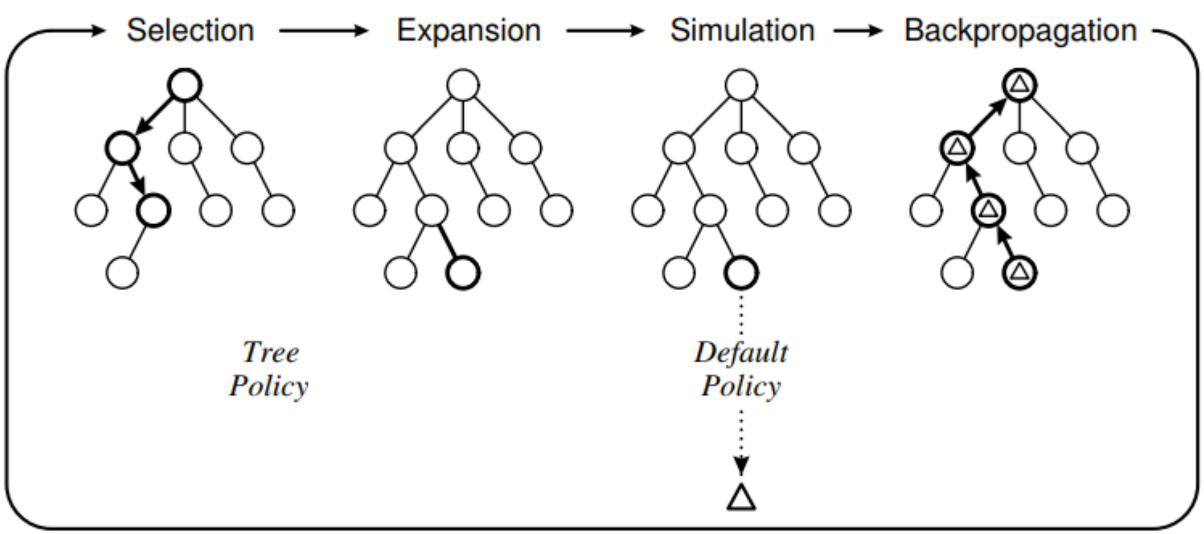


Outline

- Monte Carlo Tree Search
    - Motivation: Computer GO
    - Upper Confidence Bound
    - MC Tree search
- Model-based Planning
- Integrating Learning and Planning

Monte Carlo Tree Search (MCTS)

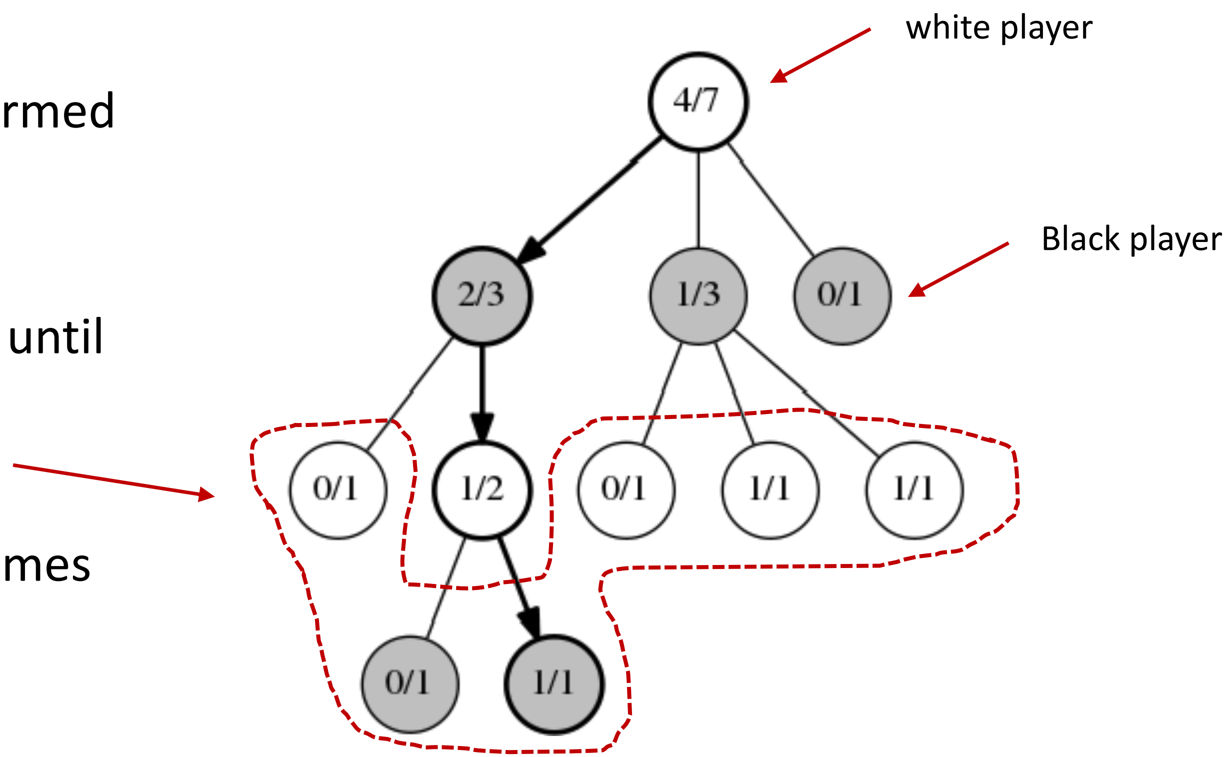
- A method for finding optimal decisions by taking random samples and building a search tree according to the results
- Profound impact on AI
- MCTS includes four steps:



Browne et al (2012)

Step 1: Selection (Tree Descent)

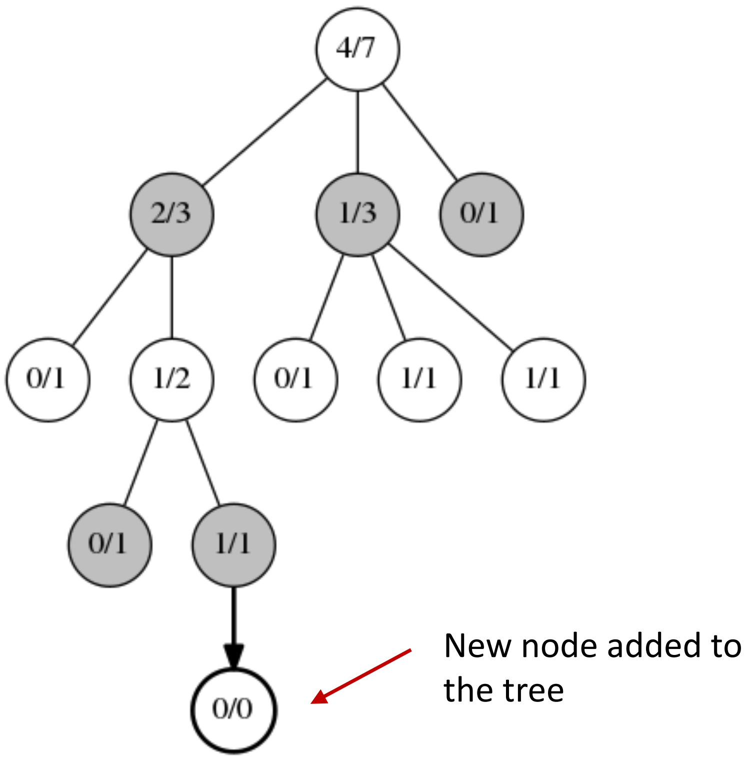
- Start at Root
- Select a child by an informed policy, e.g. UCB1
- Move to the child
- Repeat above step 2&3 until hitting tree boundary



- 4/7 = 4 wins / total 7 games

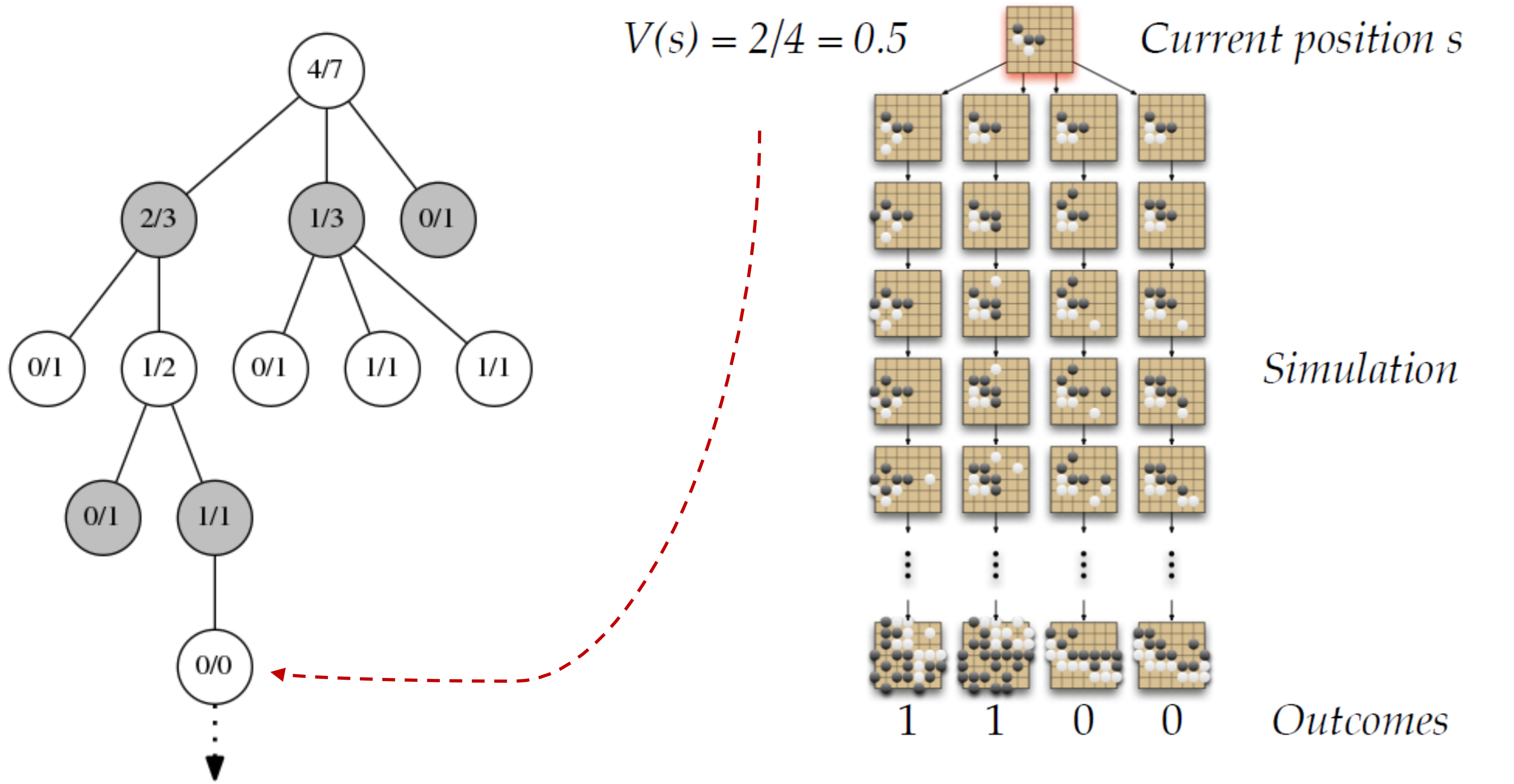
Step 2: Expansion

- At boundary, no loner apply UCB1
- Then, an unvisited child position is randomly chosen
- A new record node is added to the tree



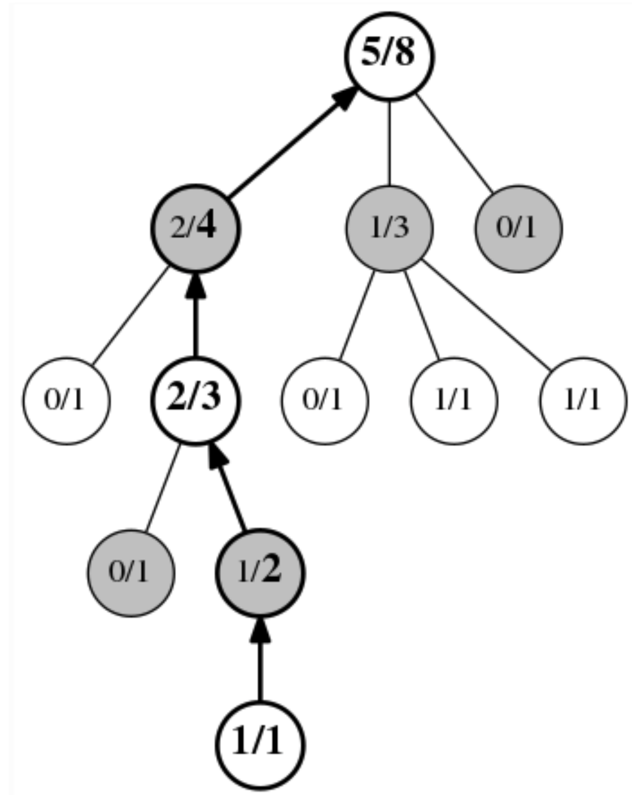
Step 3: Simulation

- Run K times typical MC simulation



Step 4: Back-Propagation

- All of the records of nodes in the path/branch taken are updated
- Each has its play count incremented by one
- Each that matches the winner has its win count increased by one



Pro and Cons

- Pros:
  - Tree growth focuses on more promising areas
  - Can stop algorithm anytime to get search results
  - Convergence to minimax solution
  - Java/Python implementation

- Cons:
  - For complex problems, enhancement needed for good performance
  - Memory intensive: entire tree in memory

Outline

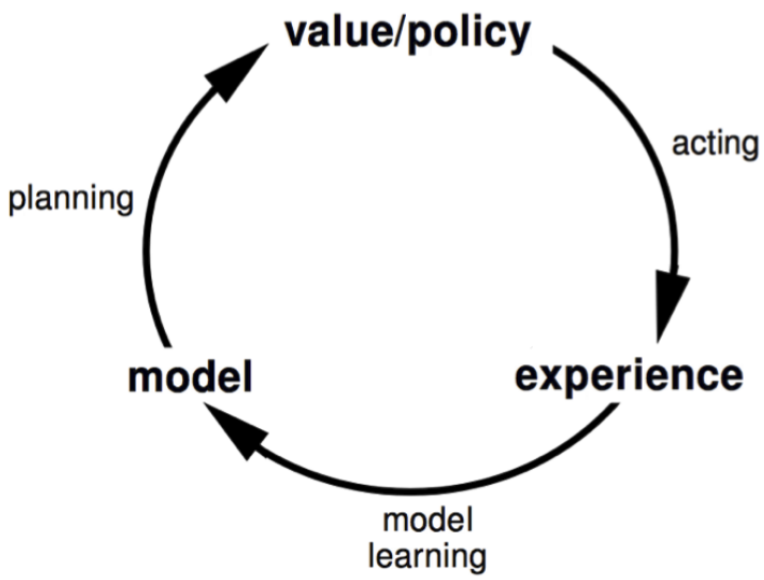
- Monte Carlo Tree Search
- Model-based Planning
- Integrating Learning and Planning

We have learnt…

- Model-Free RL
  - No Model
  - Learn value functions or policy from experience
- Model-based RL
  - Model is known
  - Plan value function or policy from model: DP algorithms, MC tree search

Question:

- If model is unknown, can we first learn model and then use the model to plan?



What is a Model?

- A *model* $\mathcal{M}$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, parametrized by η
- We will assume state space $\mathcal{S}$ and action space $\mathcal{A}$ are known
- So a model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ represents state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

- Typically assume conditional independence between state transitions and rewards

$$\mathbb{P}\left[S_{t+1}, R_{t+1} \mid S_t, A_t\right] = \mathbb{P}\left[S_{t+1} \mid S_t, A_t\right] \mathbb{P}\left[R_{t+1} \mid S_t, A_t\right]$$

Model Learning

- Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, ..., S_T\}$
- This is a supervised learning problem

$$\begin{gathered} S_1, A_1 \rightarrow R_2, S_2 \\ S_2, A_2 \rightarrow R_3, S_3 \\ \vdots \\ S_{T-1}, A_{T-1} \rightarrow R_T, S_T \end{gathered}$$

- Learning $s, a \rightarrow r$ is a *regression* problem
- Learning $s, a \rightarrow s'$ is a *density estimation* problem
- Pick loss function, e.g. mean-squared error, KL divergence, ...
- Find parameters η that minimise empirical loss

Example: Table Lookup Model

- Model is an explicit MDP, $\hat{\mathcal{P}}, \hat{\mathcal{R}}$
- Count visits $N(s,a)$ to each state action pair

$$\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, S_{t+1} = s, a, s')$$

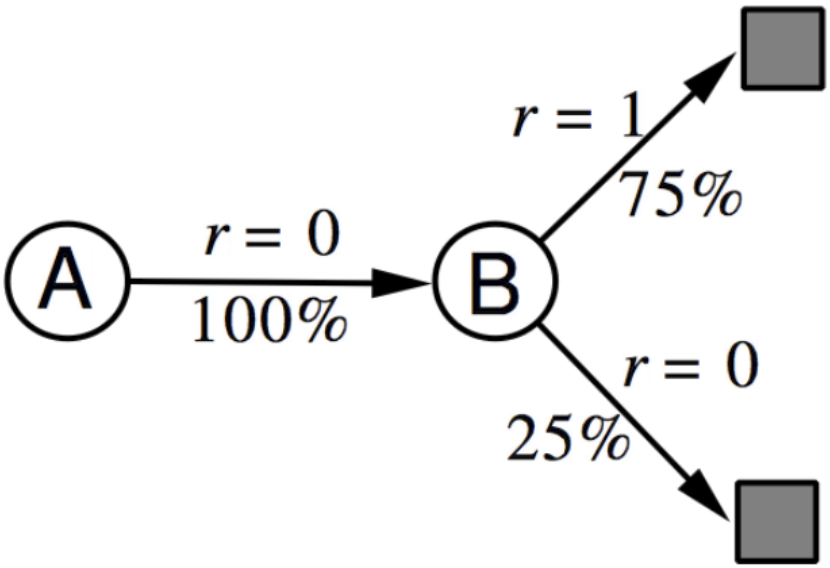
$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t = s, a) R_t$$

- Alternatively
  - At each time-step t, record experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$
  - To sample model, randomly pick tuple matching $\langle s, a, \cdot, \cdot \rangle$

A-B Example

- Two states A, B; no discounting; 8 episodes of experience

$\mathrm{A}, 0, \mathrm{B}, 0$
$\mathrm{B}, 1$
$\mathrm{B}, 1$
$\mathrm{B}, 1$
$\mathrm{B}, 1$
$\mathrm{B}, 1$
$\mathrm{B}, 1$
$\mathrm{B}, 0$



- We have constructed a table lookup model from the experience

Planning with a Model

- Model is known
- Need to solve MDP
- Using planning algorithms we have learnt:
  - Value iteration
  - Policy iteration
  - Monte Carlo tree search
  - …

Sample-based Planning

- A simple but powerful approach to planning
- Use the model only to generate samples
- Sample experience from model

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

- Apply model-free RL to samples, e.g.:

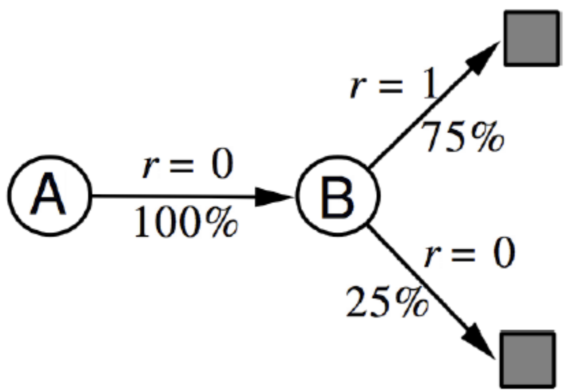
  Monte-Carlo control
  Sarsa
  Q-learning
- Sample-based planning methods are often more efficient

AB Example

- Construct a table-lookup model from real experience
- Apply model-free RL to sampled experience

Real experience

A,	0,	B,	0
B,	1		
B,	1		
B,	1		
B,	1		
B,	1		
B,	1		
B,	0		



Sampled experience

B,	1		
B,	0		
B,	1		
A,	0,	B,	1
B,	1		
A,	0,	B,	1
B,	1		
B,	0		

e.g. Monte-Carlo learning: $V(A) = 1, V(B) = 0.75$

Outline

- Monte Carlo Tree Search
- Model-based Planning
- Integrating Learning and Planning

Real and Simulated Experience

- We consider two sources of experience
- Real experience Sampled from environment (true MDP)

$$S' \sim \mathcal{P}^a_{s,s'}$$
$$R = \mathcal{R}^a_s$$

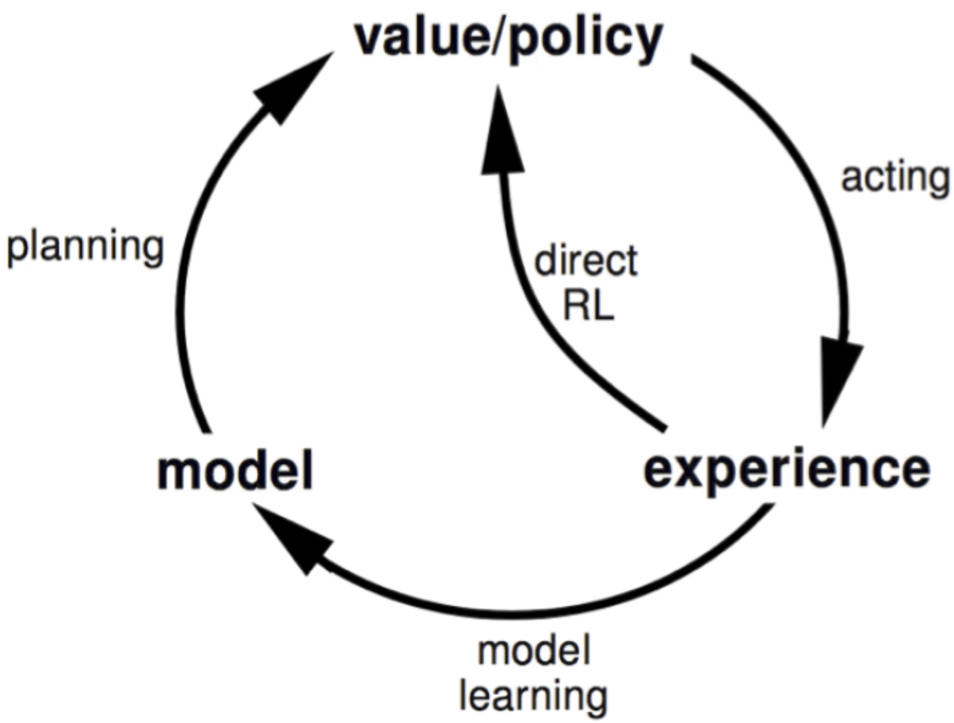
- Simulated experience Sampled from model (approximate MDP)

$$S' \sim \mathcal{P}_\eta(S' \mid S, A)$$
$$R = \mathcal{R}_\eta(R \mid S, A)$$

Integrating Learning and Planning

- Model Free RL
  - No model
  - Learn value function and/or policy from real experience
- Model based RL (using sample-based planning)
  - Learn a model from real experience
  - Plan value function and/or policy from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn and plan value function and/or policy from real and simulated experience

Dyna Architecture



Dyna-Q Algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S, Q)

(c) Execute action A; observe resultant reward, R, and state, S'

(d) $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$

(e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Repeat n times:

$S \leftarrow$ random previously observed state

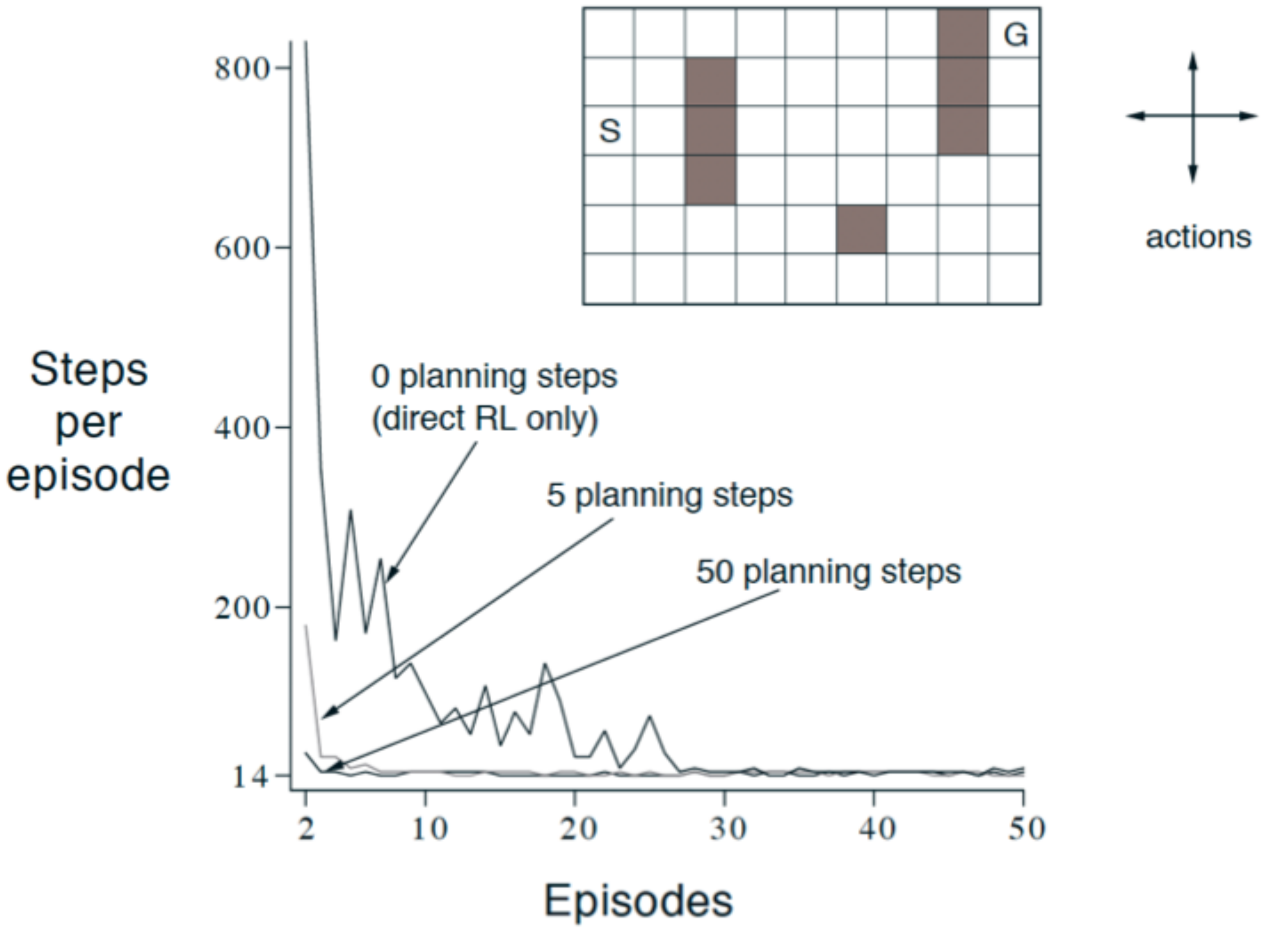
$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S, A)$

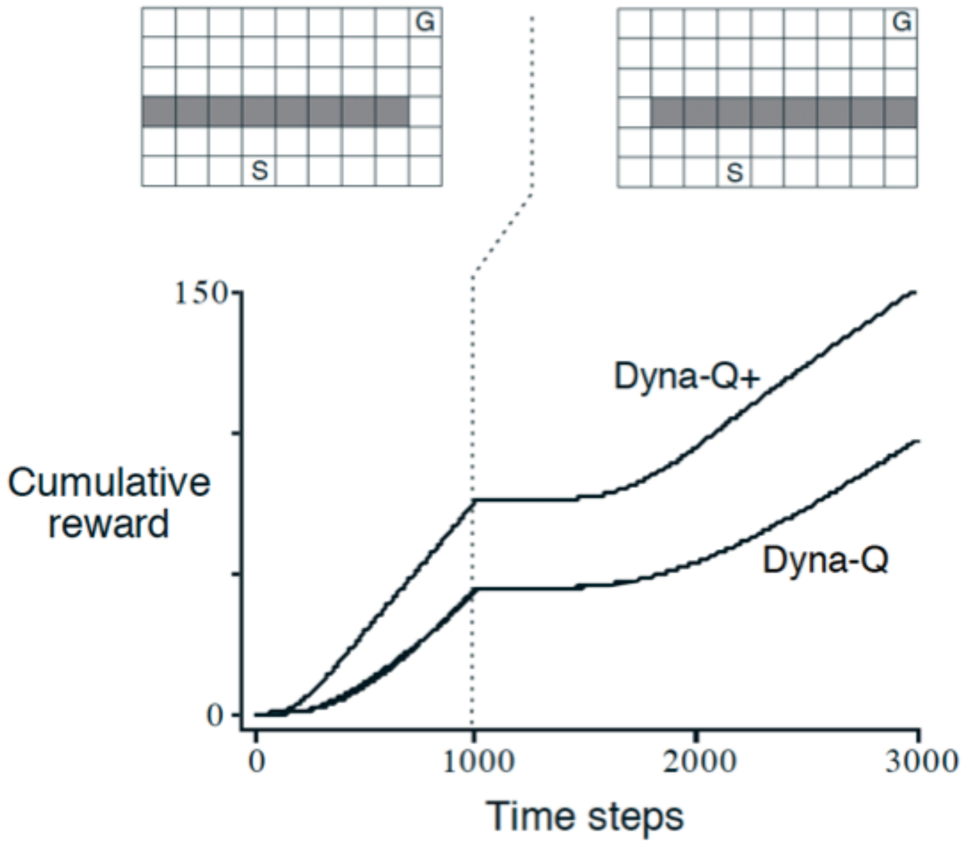
$Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$

Model-based planning

Dyna-Q on a Simple Maze

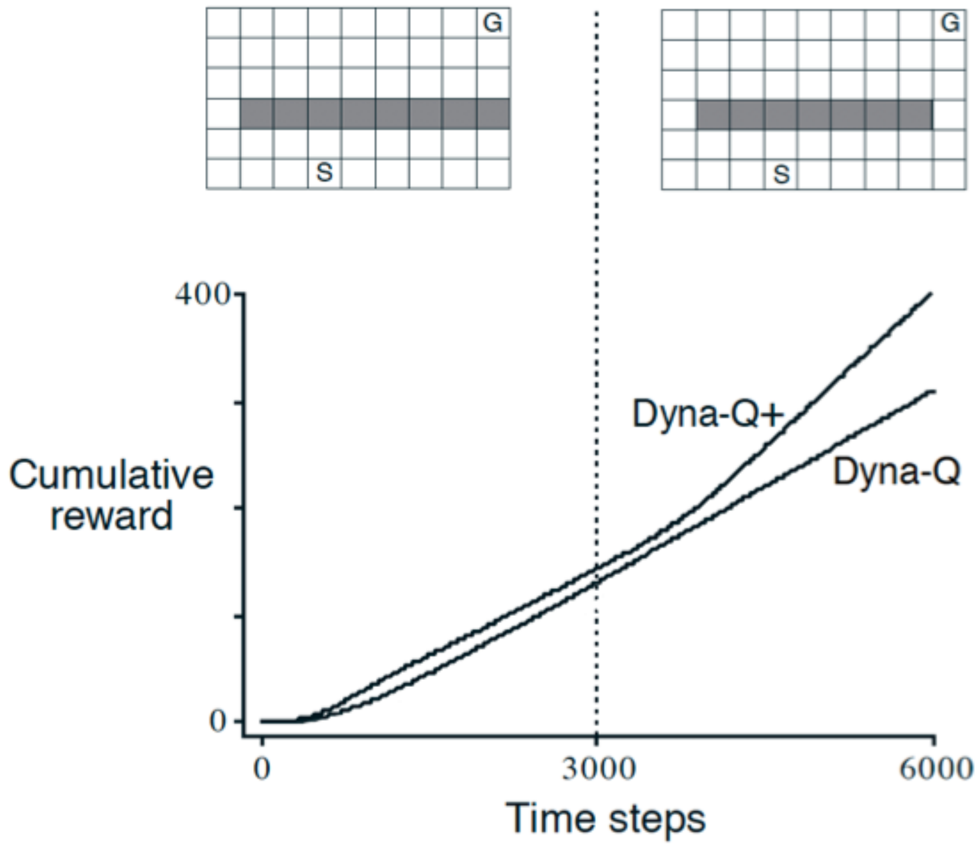


Dyna-Q on Blocking Maze



The left environment was used for the first 1000 steps, the right environment for the rest.
Dyna-Q+ is Dyna-Q with an exploration bonus that encourages exploration

Dyna-Q on Shortcut Maze



The left environment was used for the first 3000 steps, the right environment for the rest