

Lecture 16: MCTS ¹

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2018

¹With many slides from or derived from David Silver

Class Structure

- Last time: Batch RL
- **This Time: MCTS**
- Next time: Human in the Loop RL

Table of Contents

- 1 Introduction
- 2 Model-Based Reinforcement Learning
- 3 Simulation-Based Search
- 4 Integrated Architectures

Model-Based Reinforcement Learning

- *Previous lectures*: learn value function or policy or directly from experience
- *This lecture*: learn **model** directly from experience
- and use **planning** to construct a value function or policy
- Integrate learning and planning into a single architecture

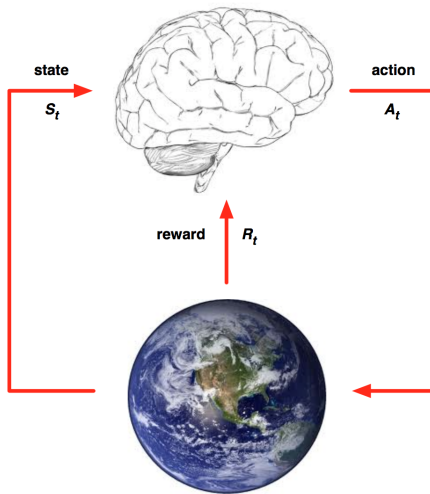
Model-Based and Model-Free RL

- Model-Free RL
 - No model
 - **Learn** value function (and/or policy) from experience

Model-Based and Model-Free RL

- Model-Free RL
 - No model
 - **Learn** value function (and/or policy) from experience
- Model-Based RL
 - Learn a model from experience
 - **Plan** value function (and/or policy) from model

Model-Free RL



Model-Based RL

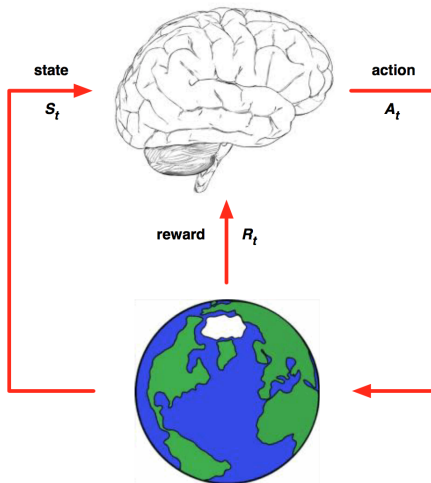
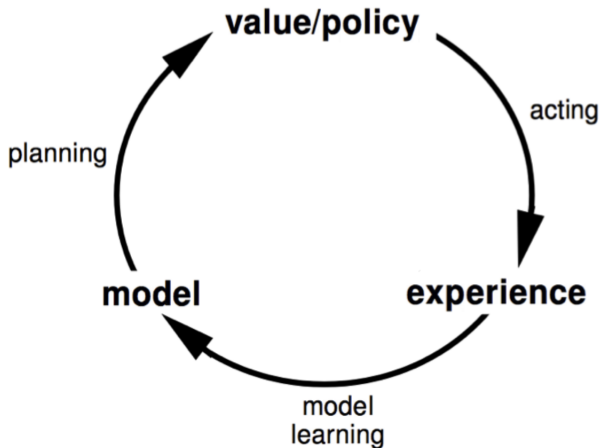


Table of Contents

- 1 Introduction
- 2 Model-Based Reinforcement Learning
- 3 Simulation-Based Search
- 4 Integrated Architectures

Model-Based RL



Advantages of Model-Based RL

- Advantages:

- Can efficiently learn model by supervised learning methods
- Can reason about model uncertainty (like in upper confidence bound methods for exploration/exploitation trade offs)

- Disadvantages

- First learn a model, then construct a value function
⇒ two sources of approximation error

MDP Model Refresher

- 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

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

Model Learning

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

$$S_1, A_1 \rightarrow R_2, S_2$$

$$S_2, A_2 \rightarrow R_3, S_3$$

$$\vdots$$

$$S_{T-1}, A_{T-1} \rightarrow R_T, S_T$$

- 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 minimize empirical loss

Examples of Models

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model
- ...

Table Lookup Model

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

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

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

- 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 $\langle s, a, \cdot, \cdot \rangle$

AB Example

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

A, 0, B, 0

B, 1

B, 1

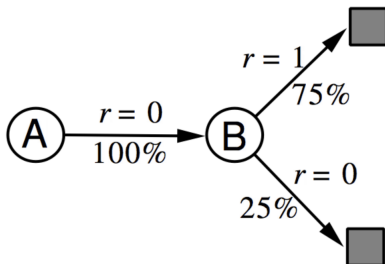
B, 1

B, 1

B, 1

B, 1

B, 0



- We have constructed a **table lookup model** from the experience
- Recall: For a particular policy, TD with a tabular representation with infinite experience replay will converge to the same value as computed if construct a MLE model and do planning
- Check Your Memory: Will MC methods converge to the same solution?

Planning with a Model

- Given a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Solve the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Using favourite planning algorithm
 - Value iteration
 - Policy iteration
 - Tree search
 - ...

Sample-Based Planning

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

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

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

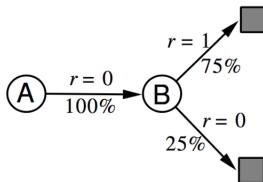
- Apply **model-free** RL to samples, e.g.:
 - Monte-Carlo control
 - Sarsa
 - Q-learning
- Sample-based planning methods are often more data efficient

Back to the 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, 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: $V(A) = 1$, $V(B) = 0.75$
- Check Your Memory: What would have MC on the original experience have converged to?

Planning with an Inaccurate Model

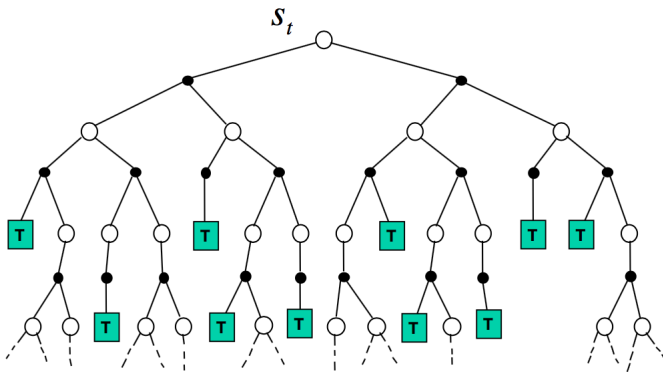
- Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$
- Performance of model-based RL is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- i.e. Model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a sub-optimal policy
- Solution 1: when model is wrong, use model-free RL
- Solution 2: reason explicitly about model uncertainty (see Lectures on Exploration / Exploitation)

Table of Contents

- 1 Introduction
- 2 Model-Based Reinforcement Learning
- 3 Simulation-Based Search**
- 4 Integrated Architectures

Forward Search

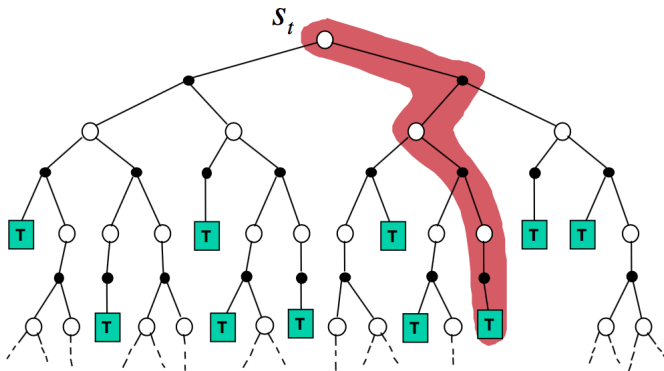
- **Forward search** algorithms select the best action by **lookahead**
- They build a **search tree** with the current state s_t at the root
- Using a **model** of the MDP to look ahead



- No need to solve whole MDP, just sub-MDP starting from now

Simulation-Based Search

- Forward search paradigm using sample-based planning
- Simulate episodes of experience from now with the model
- Apply model-free RL to simulated episodes



Simulation-Based Search (2)

- **Simulate** episodes of experience from **now** with the model

$$\{S_t^k, A_t^k, R_{t+1}^k, \dots, S_T^k\}_{k=1}^K \sim \mathcal{M}_v$$

- Apply **model-free** RL to simulated episodes
 - Monte-Carlo control $\rightarrow$ Monte-Carlo search
 - Sarsa $\rightarrow$ TD search

Simple Monte-Carlo Search

- Given a model $\mathcal{M}_v$ and a **simulation policy** π
- For each action $a \in \mathcal{A}$
 - Simulate K episodes from current (real) state s_t

$$\{\mathbf{s}_t, \mathbf{a}, R_{t+1}^k, \dots, S_T^k\}_{k=1}^K \sim \mathcal{M}_v, \pi$$

- Evaluate actions by mean return (**Monte-Carlo evaluation**)

$$Q(\mathbf{s}_t, \mathbf{a}) = \frac{1}{K} \sum_{k=1}^K G_t \xrightarrow{P} q_\pi(s_t, a) \quad (1)$$

- Select current (real) action with maximum value

$$a_t = \operatorname{argmax}_{a \in A} Q(s_t, a)$$

Recall Expectimax Tree

- If have a MDP model $\mathcal{M}_v$
- Can compute optimal $q(s, a)$ values for current state by constructing an expectimax tree

- Limitations: Size of tree scales as $(|S||A|)^H$

¹With many slides from or derived from David Silver

Monte-Carlo Tree Search (MCTS)

- Given a model $\mathcal{M}_v$
- Build a search tree rooted at the current state s_t
- Samples actions and next states
- Iteratively construct and update tree by performing K simulation episodes starting from the root state
- After search is finished, select current (real) action with maximum value in search tree

$$a_t = \operatorname{argmax}_{a \in A} Q(s_t, a)$$

Monte-Carlo Tree Search

- Simulating an episode involves two phases (in-tree, out-of-tree)
 - **Tree policy**: pick actions for tree nodes to maximize $Q(S, A)$
 - **Roll out policy**: e.g. pick actions randomly, or another policy
- To evaluate the value of a tree node i at state action pair (s, a) , average over all rewards received from that node onwards across simulated episodes in which this tree node was reached

$$Q(i) = \frac{1}{N(i)} \sum_{k=1}^K \sum_{u=t}^T \mathbb{1}(i \in \text{epi}.k) G_k(i) \xrightarrow{P} q(s, a) \quad (2)$$

- Under mild conditions, converges to the optimal search tree, $Q(S, A) \rightarrow q^*(S, A)$

Upper Confidence Tree (UCT) Search

- How to select what action to take during a simulated episode?

Upper Confidence Tree (UCT) Search

- How to select what action to take during a simulated episode?
- UCT: borrow idea from bandit literature and treat each node where can select actions as a multi-armed bandit (MAB) problem
- Maintain an upper confidence bound over reward of each arm

Upper Confidence Tree (UCT) Search

- How to select what action to take during a simulated episode?
- UCT: borrow idea from bandit literature and treat each node where can select actions as a multi-armed bandit (MAB) problem
- Maintain an upper confidence bound over reward of each arm

$$Q(s, a, i) = \frac{1}{N(s, a, i)} \sum_{k=1}^K \sum_{u=t}^T \mathbb{1}(i \in \text{epi}.k) G_k(s, a, i) + c \sqrt{\frac{\ln(n(s))}{n(s, a)}} \quad (3)$$

- For simplicity can treat each state node as a separate MAB
- For simulated episode k at node i , select action/arm with highest upper bound to simulate and expand (or evaluate) in the tree

$$a_{ik} = \arg \max Q(s, a, i) \quad (4)$$

- This implies that the policy used to simulate episodes with (and expand/update the tree) can change across each episode

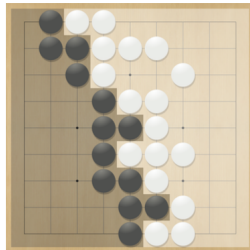
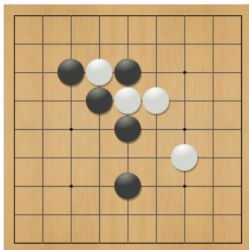
Case Study: the Game of Go

- Go is 2500 years old
- Hardest classic board game
- Grand challenge task (John McCarthy)
- Traditional game-tree search has failed in Go
- Check your understanding: does playing Go involve learning to make decisions in a world where dynamics and reward model are unknown?



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

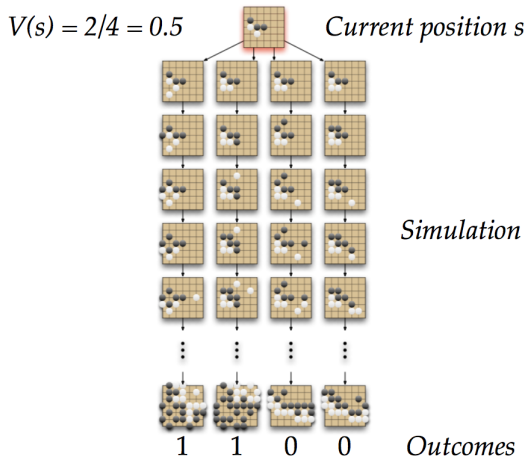
- How good is a position s
- Reward function (undiscounted):

$$\begin{aligned} R_t &= 0 \text{ for all non-terminal steps } t < T \\ R_T &= \begin{cases} 1, & \text{if Black wins.} \\ 0, & \text{if White wins.} \end{cases} \end{aligned} \tag{5}$$

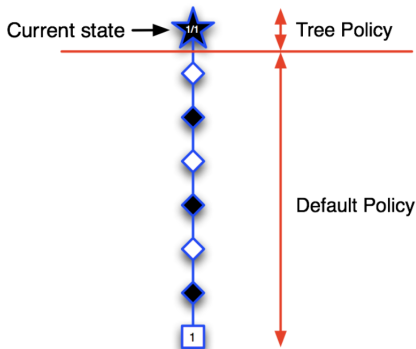
- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_T \mid S = s] = \mathbb{P}[\text{Black wins} \mid S = s] \\ v^*(s) &= \max_{\pi_B} \min_{\pi_W} v_\pi(s) \end{aligned}$$

Monte-Carlo Evaluation in Go

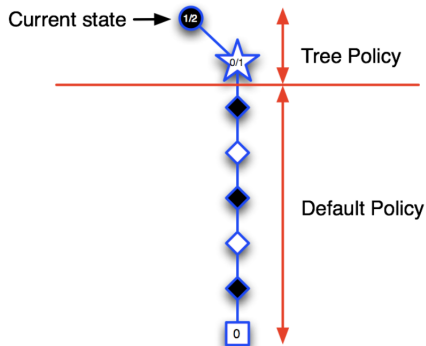


Applying Monte-Carlo Tree Search (1)

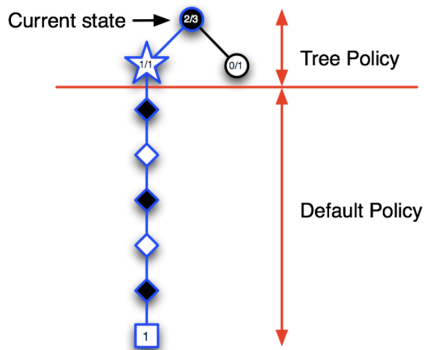


- Go is a 2 player game so tree is a minimax tree instead of expectimax
- White minimizes future reward and Black maximizes future reward when computing action to simulate

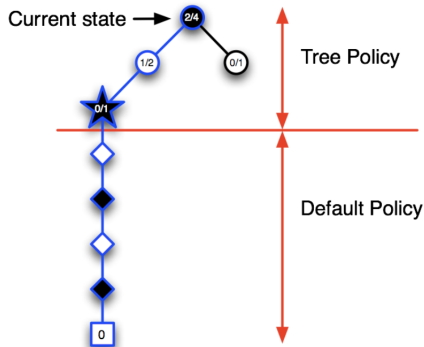
Applying Monte-Carlo Tree Search (2)



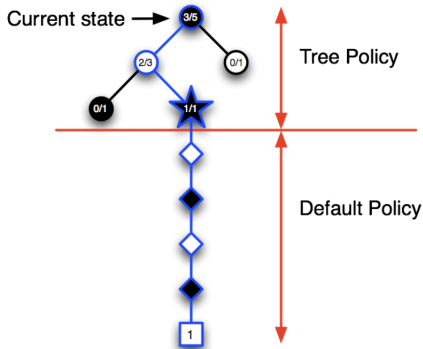
Applying Monte-Carlo Tree Search (3)



Applying Monte-Carlo Tree Search (4)



Applying Monte-Carlo Tree Search (5)



Advantages of MC Tree Search

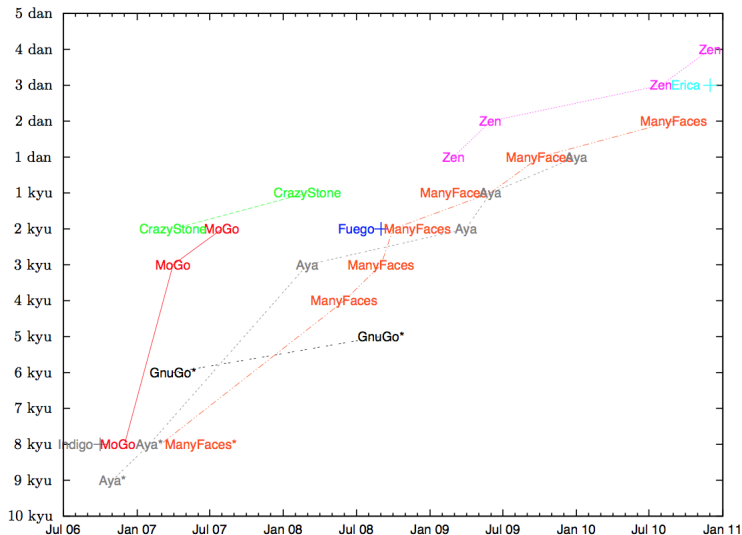
- Highly selective best-first search
- Evaluates states dynamically (unlike e.g. DP)
- Uses sampling to break curse of dimensionality
- Works for “black-box” models (only requires samples)
- Computationally efficient, anytime, parallelisable

In more depth: Upper Confidence Tree (UCT) Search

- UCT: borrow idea from bandit literature and treat each tree node where can select actions as a multi-armed bandit (MAB) problem
- Maintain an upper confidence bound over reward of each arm and select the best arm
- Check your understanding: Why is this slightly strange? Hint: why were upper confidence bounds a good idea for exploration/ exploitation? Is there an exploration/ exploitation problem during simulated episodes?¹

¹Relates to metalevel reasoning (for an example related to Go see "Selecting Computations: Theory and Applications", Hay, Russell, Tolpin and Shimony 2012)

MCTS and Early Go Results



MCTS Variants

- UCT and vanilla MCTS are just the beginning
- Potential extensions / alterations?

- UCT and vanilla MCTS are just the beginning
- Potential extensions / alterations?
 - Use a better rollout policy (have a policy network? Learned from expert data or from data gathered in the real world)
 - Learn a value function (can be used in combination with simulated trajectories to get a state-action estimate, can be used to bias initial actions considered, can be used to avoid having to rollout to the full episode length, ...)
 - Many other possibilities

MCTS and AlphaGo / AlphaZero ...

- MCTS was a critical advance for defeating Go
- Several new versions including AlphaGo Zero and AlphaZero which have even more impressive performance
- AlphaZero has also been applied to other games now including Chess

Table of Contents

- 1 Introduction
- 2 Model-Based Reinforcement Learning
- 3 Simulation-Based Search
- 4 Integrated Architectures

Real and Simulated Experience

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

$$S' \sim \mathcal{P}_{s,s'}^a$$

$$R = \mathcal{R}_s^a$$

- 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

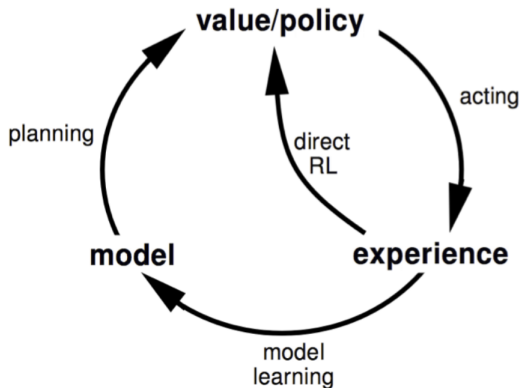
Integrating Learning and Planning

- 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

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

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

(f) Repeat n times:

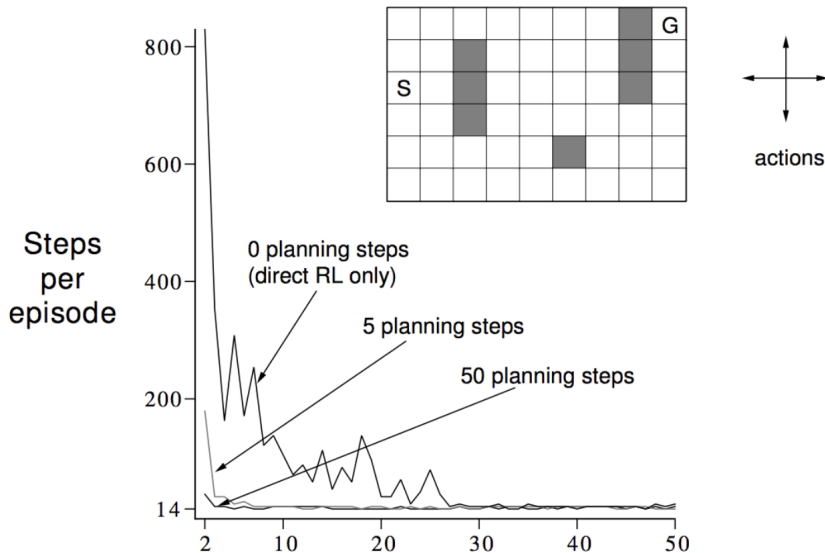
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

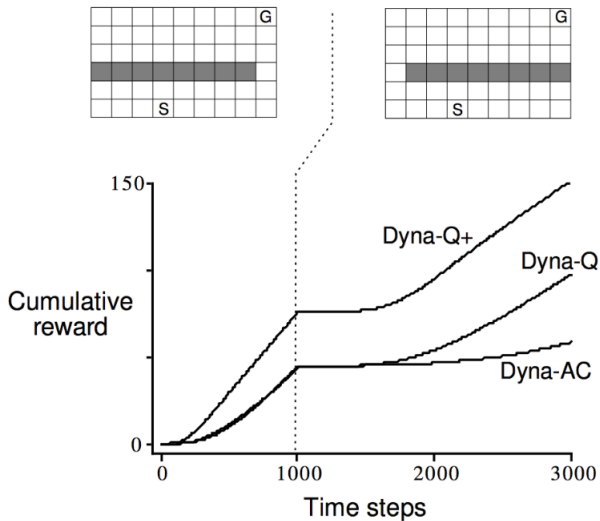
$R, S' \leftarrow Model(S, A)$

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

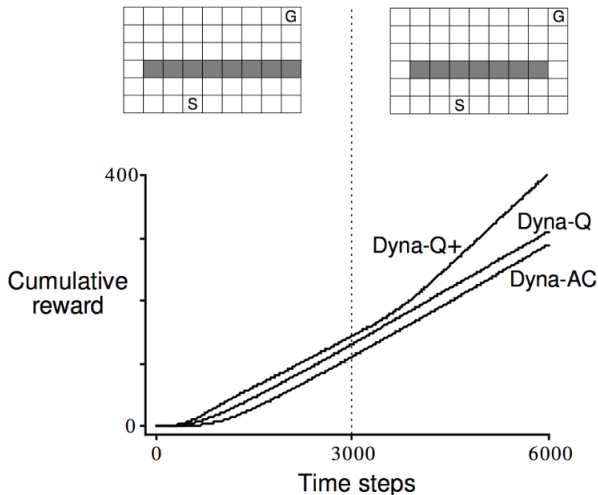
Dyna-Q on a Simple Maze



Dyna-Q with an Inaccurate Model



Dyna-Q with an Inaccurate Model (2)



Class Structure

- Last time: Batch RL
- **This Time: MCTS**
- Next time: Human in the Loop RL