Lecture 9: Integrating Learning and Planning

Joseph Modayil

Outline

1 Introduction

2 Model-Based Reinforcement Learning

3 Model-Based Reinforcement Learning

4 Integrated Architectures

5 Simulation-Based Search

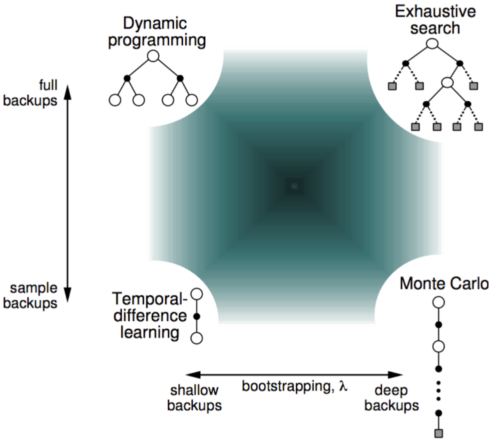
Model-Based Reinforcement Learning

- *Last lecture*: learn policy directly from experience
- *Previous lectures*: learn value function directly from experience
- *This lecture*:
  - Learn model directly from experience (or be given a model)
  - Plan with the model to construct a value function or policy
  - Integrate learning and planning into a single architecture

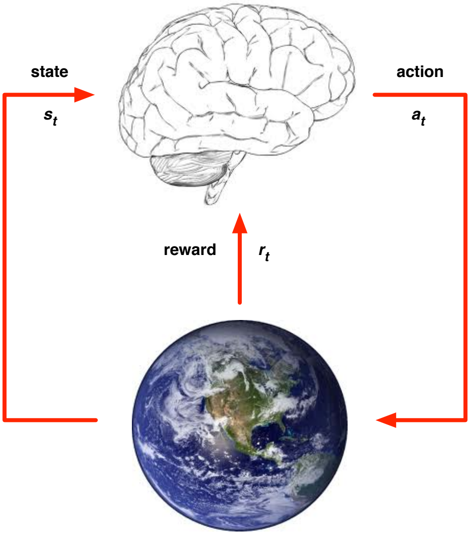
Model-Based and Model-Free RL

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

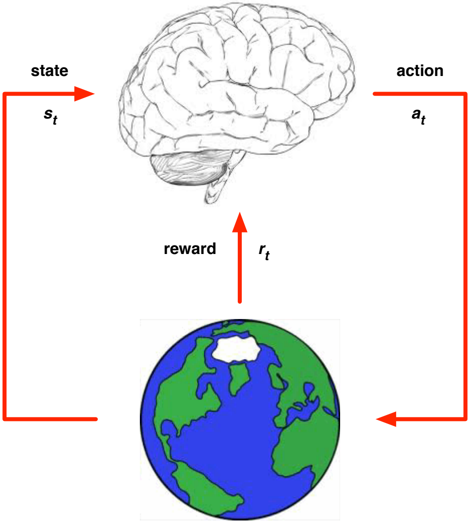
Filling in the middle of algorithm space



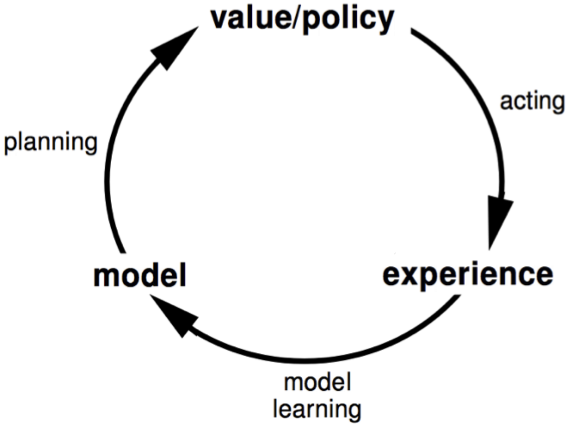
Model-Free RL



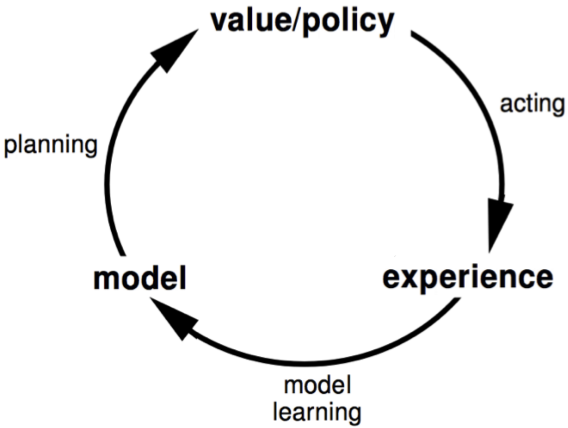
Model-Based RL



Model-Based RL



Model-Based RL



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
- A model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ approximates the state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$. e.g.

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

  This particular model imposes conditional independence between state transitions and rewards

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

- Conventionally a method is called *model-based* when the transition and reward dynamics are explicitly represented (to support planning), and as *model-free* otherwise. Some new methods lie in-between these extremes.

Model Learning

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

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

- Learn a function $s, a \rightarrow r$ and also learn a function $s, a \rightarrow s'$
- Pick loss function (e.g. mean-squared error), and find parameters η that minimise 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

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

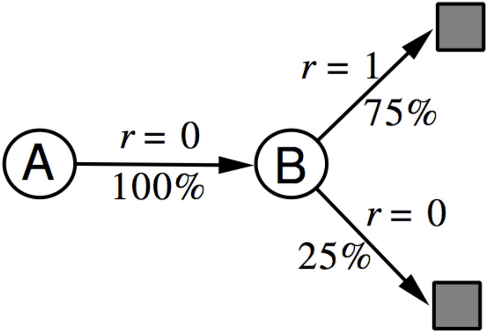
$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t = s, a) R_t$$

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

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

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

$$
\begin{aligned}
s_{t+1} &\sim \mathcal{P}_\eta(s_{t+1} \mid s_t, a_t) \\
r_{t+1} &= \mathcal{R}_\eta(r_{t+1} \mid s_t, a_t)
\end{aligned}
$$

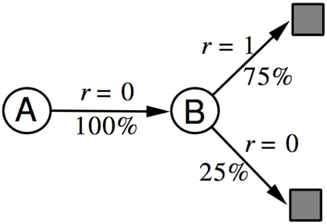
- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- Sample-based planning methods are often more efficient

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, 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: $V(A) = 1, V(B) = 0.75$

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 suboptimal policy
- Approach 1: when model is wrong, use model-free RL
- Approach 2: reason explicitly about model uncertainty over η (e.g. Bayesian methods)
- Approach 3: Combine model-based and model-free methods in a safe way.

Real and Simulated Experience

We consider two sources of experience

Real experience Sampled from environment (true MDP)

$$s' \sim \mathcal{P}_{s,s'}^{a}$$
$$r = \mathcal{R}_{s}^{a}$$

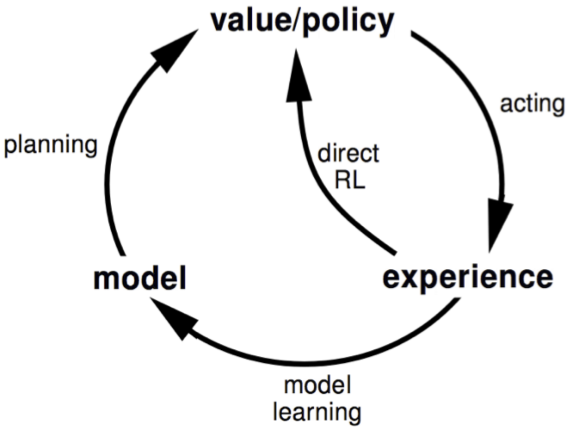
Simulated experience Sampled from model (approximate MDP)

$$s' \sim \mathcal{P}_{\eta}(s' \mid s, a)$$
$$r = \mathcal{R}_{\eta}(r \mid s, a)$$

Integrating Learning and Planning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- 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
  - Treat real and simulated experience equivalently. Conceptually, the updates from learning or planning are not distinguished.

Dyna Architecture



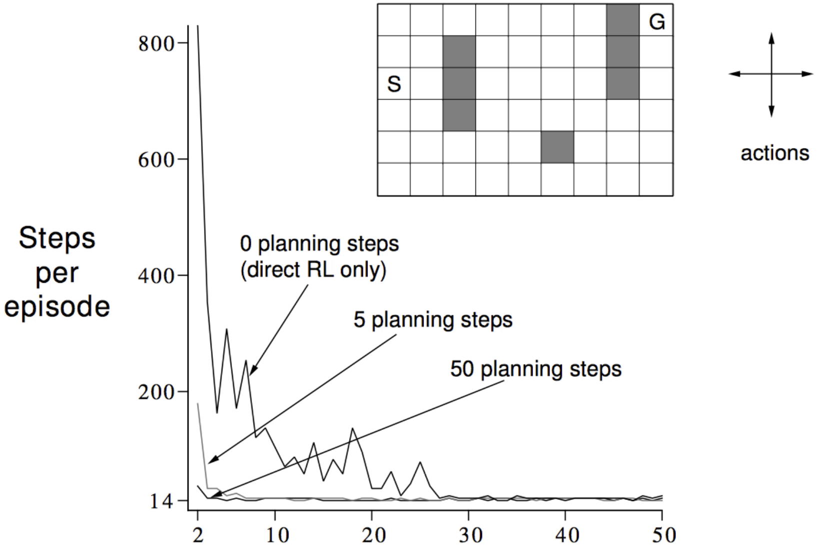
Dyna-Q Algorithm

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 state, s', and reward, r
(d) $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
(f) Repeat N times:

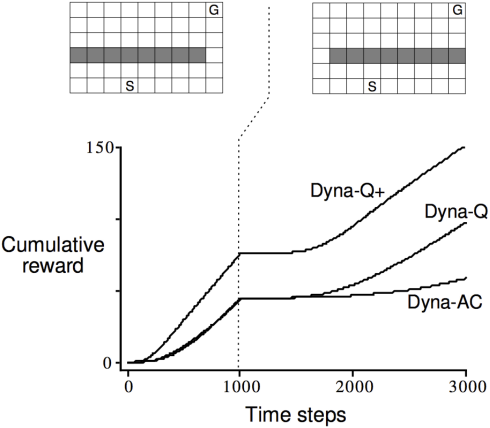
$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in s
$s', r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Dyna-Q on a Simple Maze



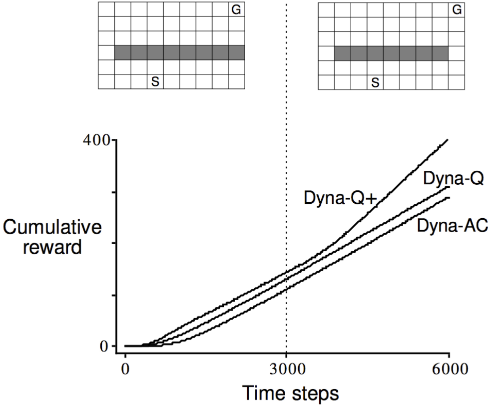
Dyna-Q with an Inaccurate Model

- The changed environment is harder



Dyna-Q with an Inaccurate Model (2)

- The changed environment is easier



Linear Dyna

- What can we do when the actual states are not known?
- Suppose we have features vectors for states $\phi(s)$.
- Consider the *expectation* models of the next reward and of the next feature vector.

$$\hat{\mathcal{R}}_s^a = \mathbb{E}_\pi\left[R_{t+1} | S_t = s, A_t = a\right]$$

$$\hat{\mathcal{F}}_s^a = \mathbb{E}_\pi\left[\phi(S_{t+1}) | S_t = s, A_t = a\right]$$

- We can make linear approximations to the expectation models.
- Equivalent fixed points with linear function approximation when learning model-free or planning with the linear expectation model.

Linear Dyna Model Updates

- Consider a single transition $\langle \phi(s), a, r, \phi(s') \rangle$
- Learn vector b_a for a reward model. $b_a^\top \phi(s) \approx \hat{\mathcal{R}}_s^a$

$$b_a \leftarrow b_a + \alpha(r - b_a^\top \phi(s))\phi(s)$$

- Learn matrix F_a for a transition model. $F_a^\top \phi(s) \approx \hat{\mathcal{F}}_s^a$

$$F_a \leftarrow F_a + \alpha(\phi(s') - F_a\phi(s))\phi(s)^\top$$

Linear Dyna Value Function Updates

- We consider a linear approximation for the action value function with $\theta_a^\top \phi(s) \approx q(s, a)$
- *Learning* uses an observed transition $\langle \phi(s), a, r, \phi(s') \rangle$,

$$\delta \leftarrow \max_{a'} r + \gamma \theta_{a'}^\top \phi(s') - \theta_a^\top \phi(s)$$
$$\theta_a \leftarrow \theta_a + \alpha \delta \phi(s)$$

- *Planning* uses the model for an expected transition from an arbitrary feature vector ψ, $\langle \psi, a, b_a^\top \psi, F_a \psi \rangle$

$$\delta \leftarrow \max_{a'} b_a^\top \psi + \gamma \theta_{a'}^\top F_a \psi - \theta_a^\top \psi$$
$$\theta_a \leftarrow \theta_a + \alpha \delta \psi$$

Linear Dyna Algorithm

- We consider a linear approximation for the action value function with $\theta_a^\top \phi(s) \approx q(s, a)$
- *Learning* uses an observed transition $\langle \phi(s), a, r, \phi(s') \rangle$,

$$\delta \leftarrow \max_{a'} r + \gamma \theta_{a'}^\top \phi(s') - \theta_a^\top \phi(s)$$
$$\theta_a \leftarrow \theta_a + \alpha \delta \phi(s)$$

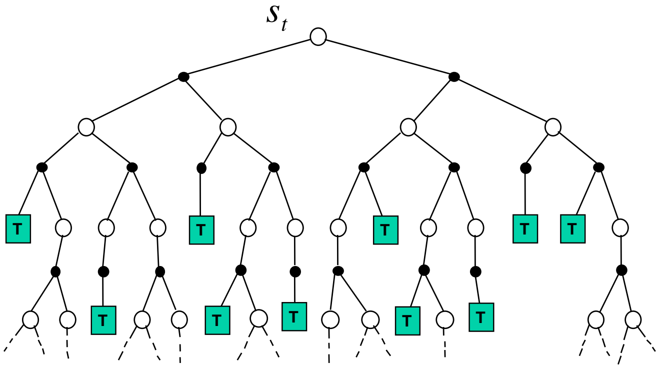
- *Planning* uses the model for an expected transition from an arbitrary feature vector ψ, $\langle \psi, a, b_a^\top \psi, F_a \psi \rangle$

$$\delta \leftarrow \max_{a'} b_a^\top \psi + \gamma \theta_{a'}^\top F_a \psi - \theta_a^\top \psi$$
$$\theta_a \leftarrow \theta_a + \alpha \delta \psi$$

- We have been learning a model and planning with it.
- Now consider that setting where the model is given (fixed), and we want to use it.

Forward Search

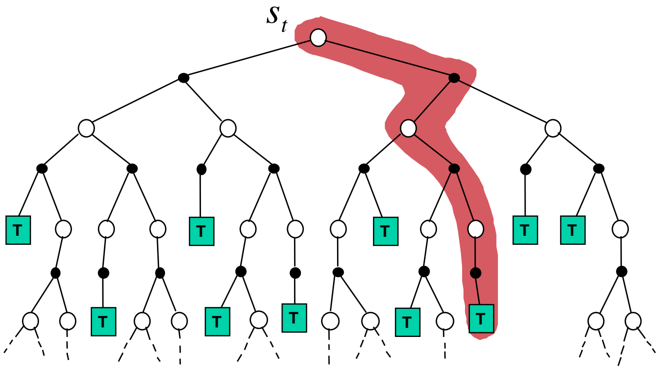
- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead



- 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

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

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

Monte-Carlo Simulation

- Given a parameterized model $\mathcal{M}_\nu$ and a simulation policy π
- Simulate K episodes from current state S_t

$$\{S_t^k = S_t, A_t^k, R_{t+1}^k, S_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

- Evaluate state by mean return (Monte-Carlo evaluation)

$$V(S_t) = \frac{1}{K}\sum_{k=1}^K G_t^k \overset{P}{\rightsquigarrow} V^\pi(S_t)$$

Monte-Carlo Tree Search (Evaluation)

- Given a model $\mathcal{M}_\nu$
- Simulate K episodes from current state S_t using current simulation policy π

$$\{S_t^k = S_t, A_t^k, R_{t+1}^k, S_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

- Build a search tree containing visited states and actions
- Evaluate states $Q(s, a)$ by mean return of episodes from s, a

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{u=t}^{T} \mathbf{1}(S_u^k, A_u^k = s, a) G_u \overset{P}{\rightsquigarrow} Q^\pi(s,a)$$

- After search is finished, select current (real) action with maximum value in search tree

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

Monte-Carlo Tree Search (Simulation)

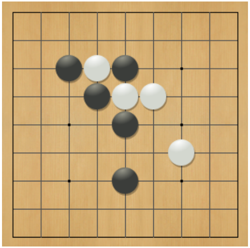
- In MCTS, the simulation policy π improves
- The simulation policy π has two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions from $Q(s, a)$ (e.g. $\epsilon - \text{greedy}(Q(s, a))$)
  - Default policy (fixed): pick actions randomly
- Repeat (for each simulated episode)
  - Select actions in tree according to tree policy.
  - Expand search tree by one node
  - Rollout to termination with default policy
  - Update action-values Q(s,a) in the tree
- Output best action when simulation time runs out.
- With some assumptions, converges to the optimal values, $Q(s, a) \rightarrow Q^*(s, a)$

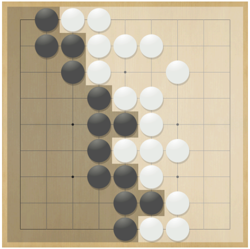
Case Study: the Game of Go

- The ancient oriental game of Go is 2500 years old
- Considered to be the hardest classic board game
- Considered a grand challenge task for AI *(John McCarthy)*
- Traditional game-tree search has failed in Go

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

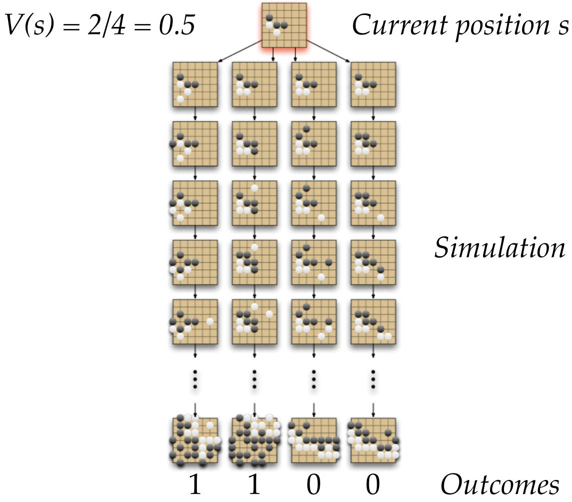
- How good is a position s?
- Reward function (undiscounted):

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

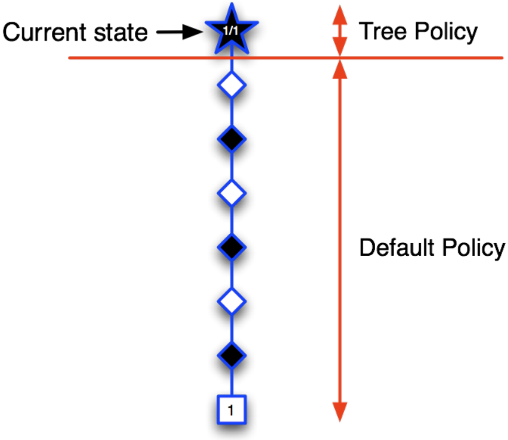
- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

$$V^\pi(s) = \mathbb{E}_\pi \left[R_T \mid s \right] = \mathbb{P}\left[\text{Black wins} \mid s \right]$$
$$V^*(s) = \max_{\pi_B} \min_{\pi_W} V^\pi(s)$$

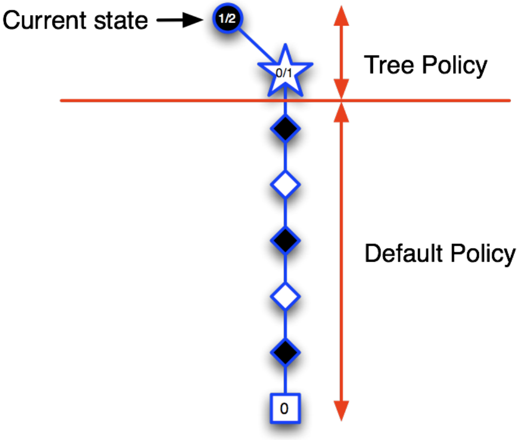
Monte-Carlo Evaluation in Go



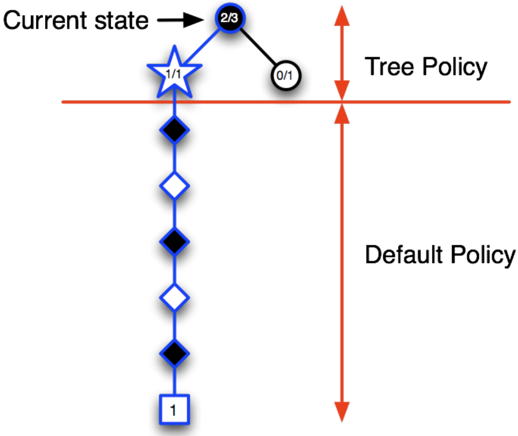
Applying Monte-Carlo Tree Search (1)



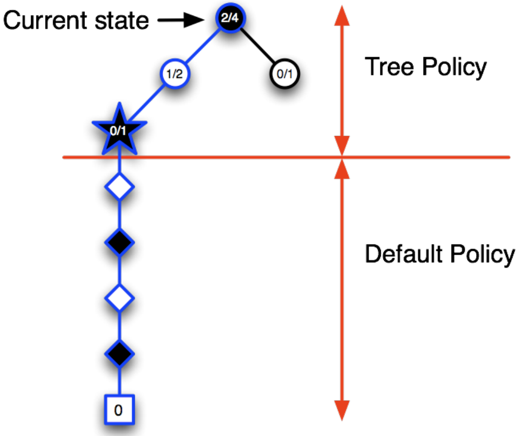
Applying Monte-Carlo Tree Search (2)



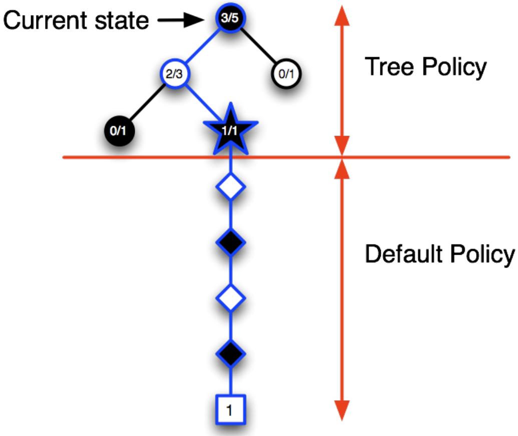
Applying Monte-Carlo Tree Search (3)



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

Example: MC Tree Search in Computer Go