Lecture 15: MCTS [1]

Emma Brunskill

CS234 Reinforcement Learning.

[1] With many slides from or derived from David Silver

Refresh Your Understanding: Batch RL

Select all that are true:

- Batch RL refers to when we have many agents acting in a batch
- In batch RL we generally care more about sample efficiency than computational efficiency
- Importance sampling can be used to get an unbiased estimate of policy performance
- Q-learning can be used in batch RL and will generally provide a better estimate than importance sampling in Markov environments for any function approximator used for the Q
- Not sure

Refresh Your Understanding: Batch RL Solutions

Select all that are true:

- Batch RL refers to when we have many agents acting in a batch
- In batch RL we generally care more about sample efficiency than computational efficiency
- Importance sampling can be used to get an unbiased estimate of policy performance
- Q-learning can be used in batch RL and will generally provide a better estimate than importance sampling in Markov environments for any function approximator used for the Q
- Not sure

Answer. F. T. T. F.

Class Structure

Last time: Professor Doshi-Velez guest lecture

This Time: MCTS

Video

- Review the following AlphaGo video:
  "AlphaGo Official Trailer"
  https://www.youtube.com/watch?v=8tq1C8spV_g

Monte Carlo Tree Search

Why choose to have this as well?

Responsible in part for one of the greatest achievements in AI in the last decade– becoming a better Go player than any human

Brings in ideas of model-based RL and the benefits of planning

Table of Contents

Introduction: Model-Based Reinforcement Learning

Previous lectures: For online learning, learn value function or policy directly from experience

This lecture: For online learning, 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

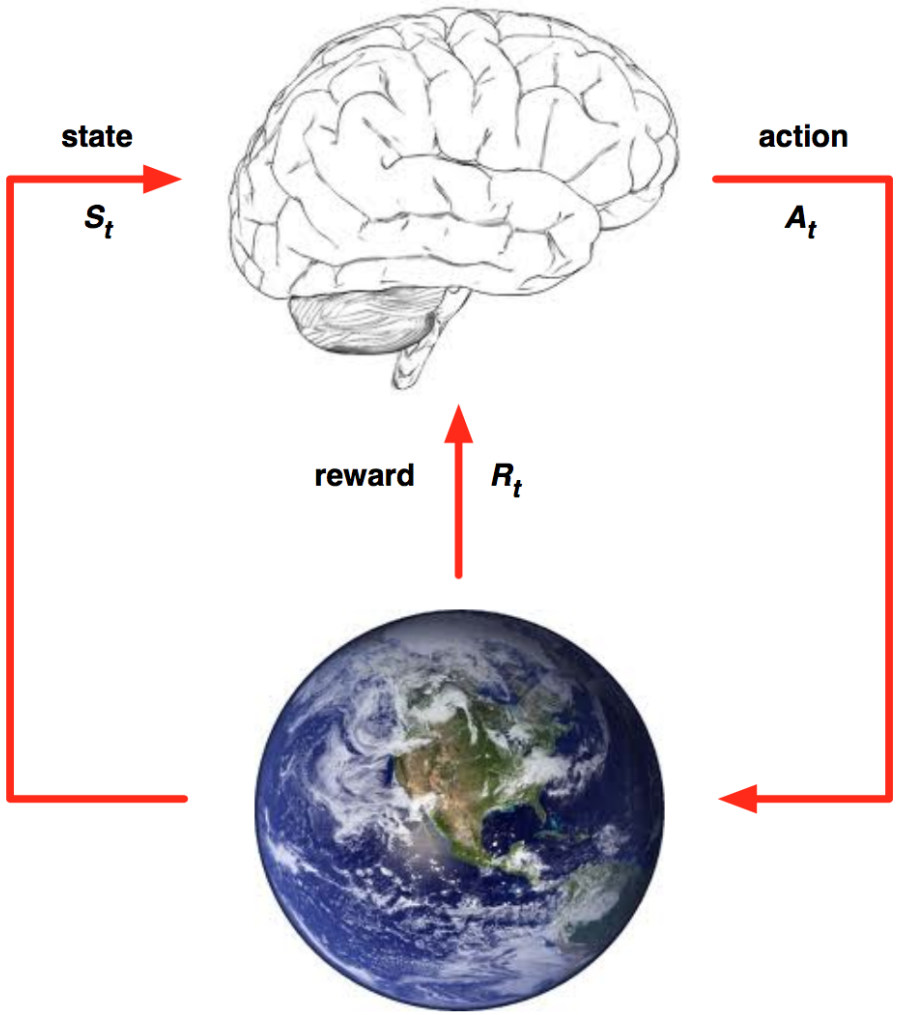
Model-Free RL

- 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

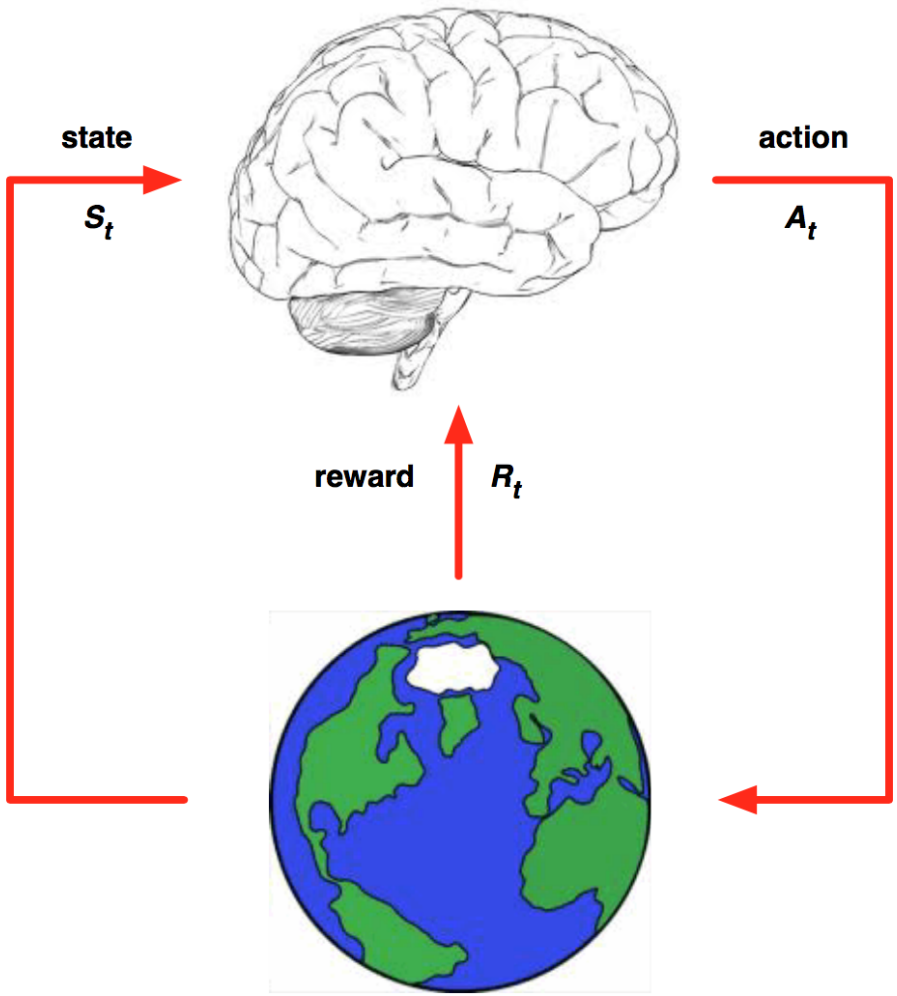
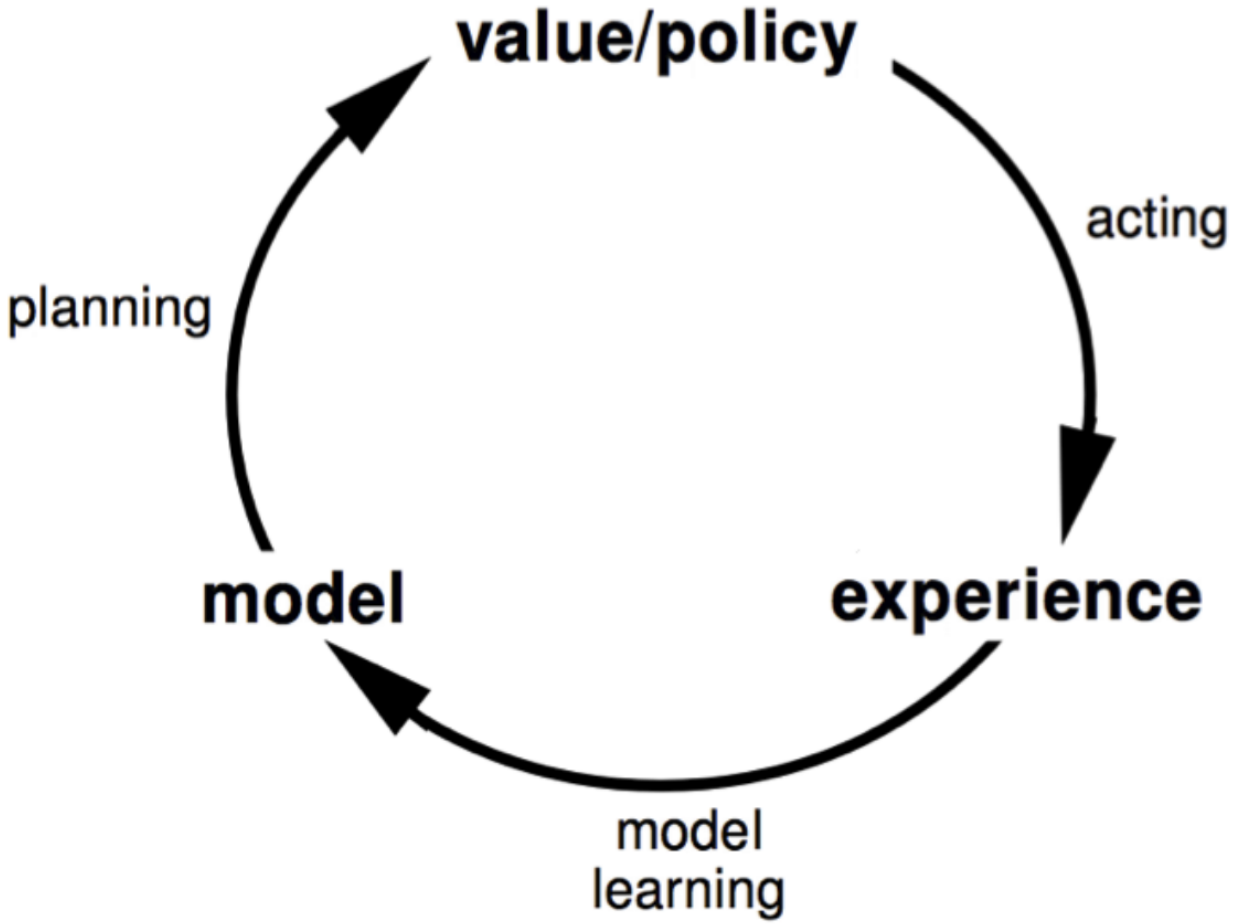


Table of Contents

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
  $\Rightarrow$ two sources of approximation error

MDP Model Refresher

A model $\mathcal{M}$ is a representation of an MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} >$, parametrized by η

We will assume state space $\mathcal{S}$ and action space $\mathcal{A}$ are known

So a model $\mathcal{M} =< \mathcal{P}_\eta, \mathcal{R}_\eta >$ 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 $\{S_1, A_1, R_2, ..., S_T\}$

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

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

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

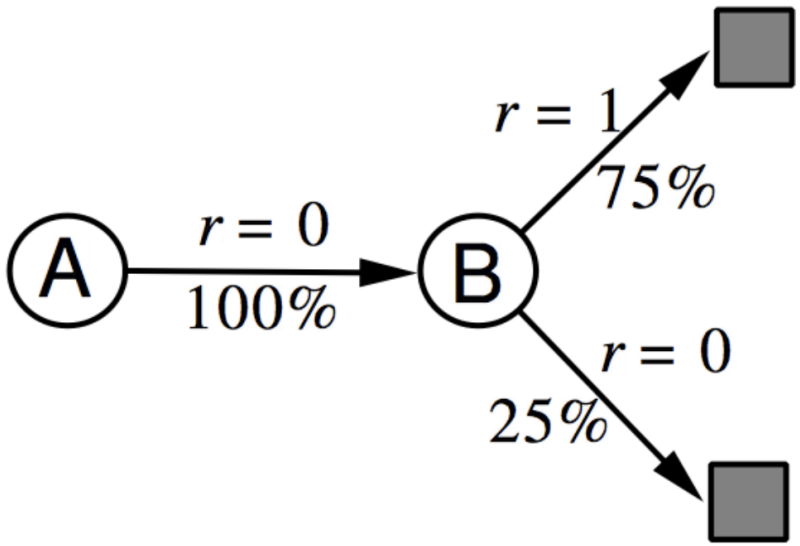
Alternatively

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

AB Example & Check Your Memory

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

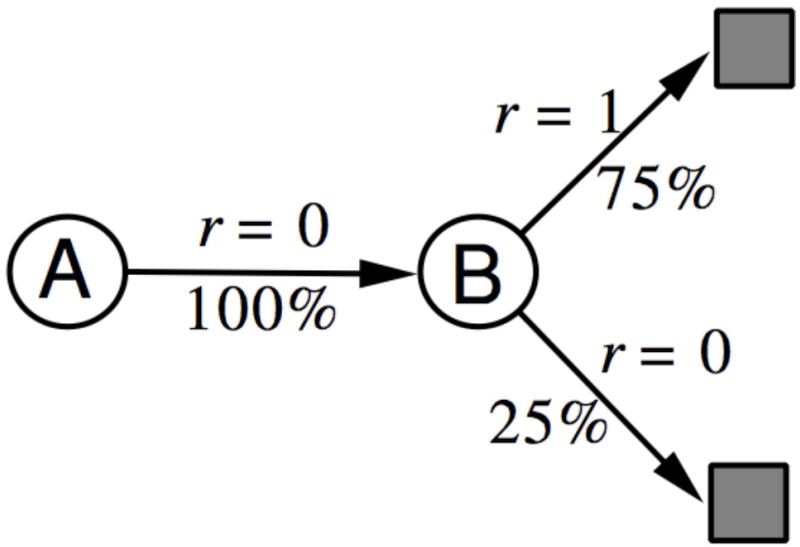
Recall: For a particular policy, TD with a tabular representation with infinite experience replay will converge to the same value as computed if construct a MLE model and do planning

Check Your Memory: Will MC methods converge to the same solution?

AB Example & Check Your Memory Solution

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?
Solution: Not necessary. MC methods will converge to solution which minimizes the MSE of the value and the observed returns

Planning with a Model

Given a model $\mathcal{M}_\eta =< \mathcal{P}_\eta, \mathcal{R}_\eta >$

Solve the MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta >$

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

Planning with an Inaccurate Model

Given an imperfect model $< \mathcal{P}_\eta, \mathcal{R}_\eta > \neq < \mathcal{P}, \mathcal{R} >$

Performance of model-based RL is limited to optimal policy for approximate MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta >$

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

Back to the AB Example

Construct a table-lookup model from real experience

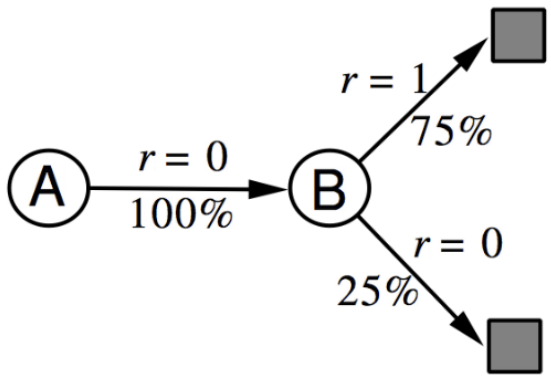
Apply model-free RL to sampled experience

Real experience

A, 0, B, 0

B, 1

B, 1



What values will TD with estimated model converge to assuming $\gamma = 1$?

Is this correct?

Back to the AB Example

Construct a table-lookup model from real experience

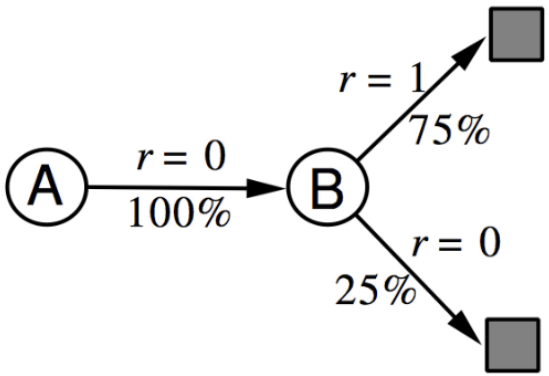
Apply model-free RL to sampled experience

Real experience

A, 0, B, 0

B, 1

B, 1



What values will TD with estimated model converge to assuming $\gamma = 1$?

V(B) $= 2/3$, V(A) $=$ V(B) $= 2/3$

Is this correct? **No**

Planning with an Inaccurate Model

Given an imperfect model $< \mathcal{P}_\eta, \mathcal{R}_\eta > \neq < \mathcal{P}, \mathcal{R} >$

Performance of model-based RL is limited to optimal policy for approximate MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta >$

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

Computing Action for Current State Only

Previously would compute a policy for whole state space

Simulation-Based Search

Simulate episodes of experience from now with the model starting from current state S_t

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

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

$$\{s_t, a, R^k_{t+1}, ..., S^k_T\}^K_{k=1} \sim \mathcal{M}_v, \pi$$

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

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

Select current (real) action with maximum value

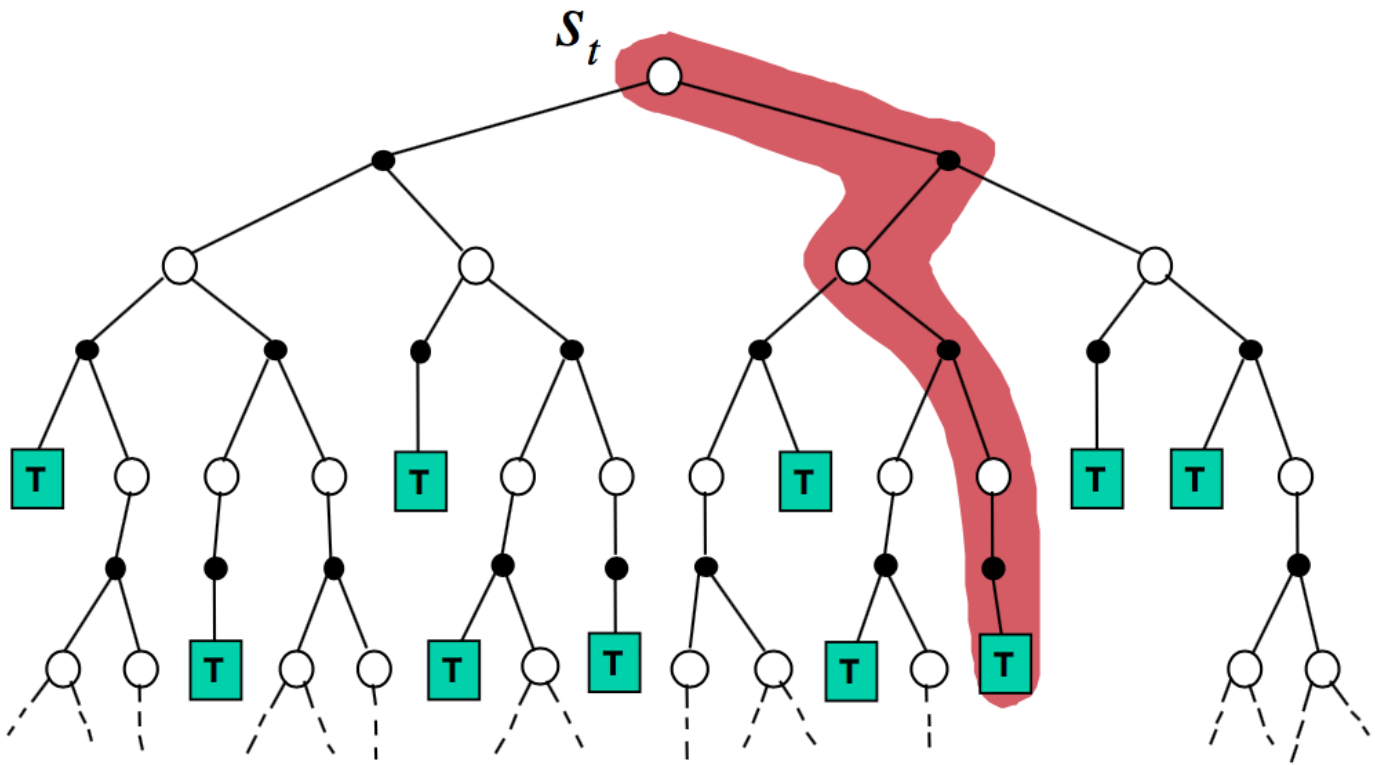
$$a_t = \underset{a \in A}{\text{argmax}}\, Q(s_t, a)$$

This is essentially doing 1 step of policy improvement

Simulation-Based Search

Simulate episodes of experience from now with the model

Apply model-free RL to simulated episodes



Expectimax Tree

Can we do better than 1 step of policy improvement?

If have a MDP model $\mathcal{M}_v$

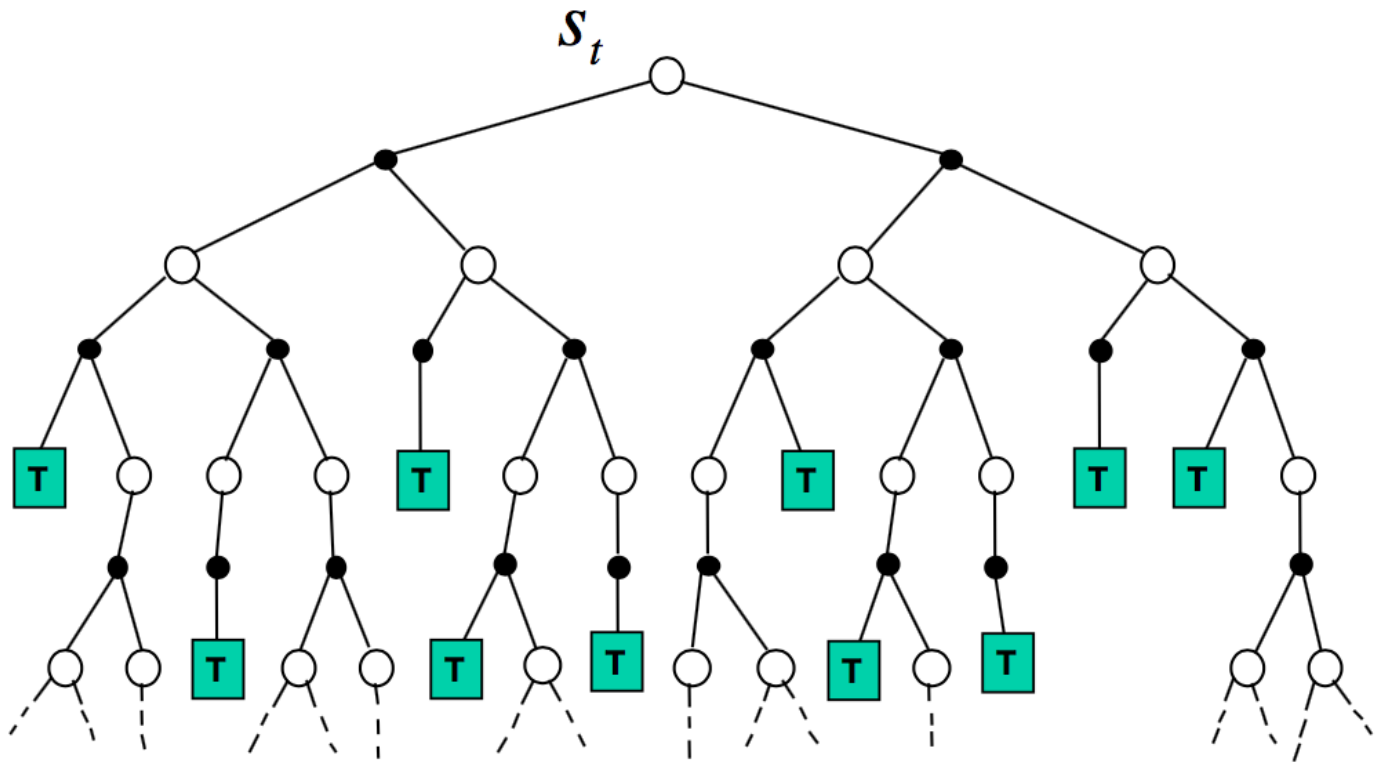
Can compute optimal $q(s, a)$ values for current state by constructing an expectimax tree

Forward Search Expectimax Tree

Forward search algorithms select the best action by lookahead

They build a search tree with the current state st at the root

Using a model of the MDP to look ahead



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

Expectimax Tree

Can we do better than 1 step of policy improvement?

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: *Try to work out yourself*

Expectimax Tree

Can we do better than 1 step of policy improvement?

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 $(|S||A|)^H$

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

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

Monte-Carlo Tree Search

Goal: Compute optimal action for current state

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

Monte-Carlo Tree Search

Goal: Compute optimal action for current state

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

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

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

Check Your Understanding: MCTS

MCTS involves deciding on an action to take by doing tree search where it picks actions to maximize $Q(S, A)$ and samples states.
Select all

- Given a MDP, MCTS may be a good choice for short horizon problems with a small number of states and actions.
- Given a MDP, MCTS may be a good choice for long horizon problems with a large action space and a small state space
- Given a MDP, MCTS may be a good choice for long horizon problems with a large state space and small action space
- Not sure

Check Your Understanding: MCTS

MCTS involves deciding on an action to take by doing tree search where it picks actions to maximize $Q(S, A)$ and samples states.
Select all

- Given a MDP, MCTS may be a good choice for short horizon problems with a small number of states and actions.
- Given a MDP, MCTS may be a good choice for long horizon problems with a large action space and a small state space
- Given a MDP, MCTS may be a good choice for long horizon problems with a large state space and small action space
- Not sure

Solutions: F F T

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

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

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

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

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

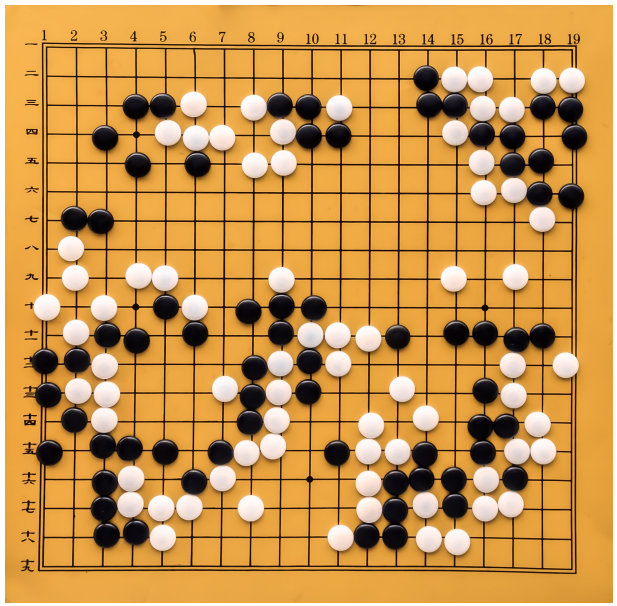
Grand challenge task (John McCarthy)

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):

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

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

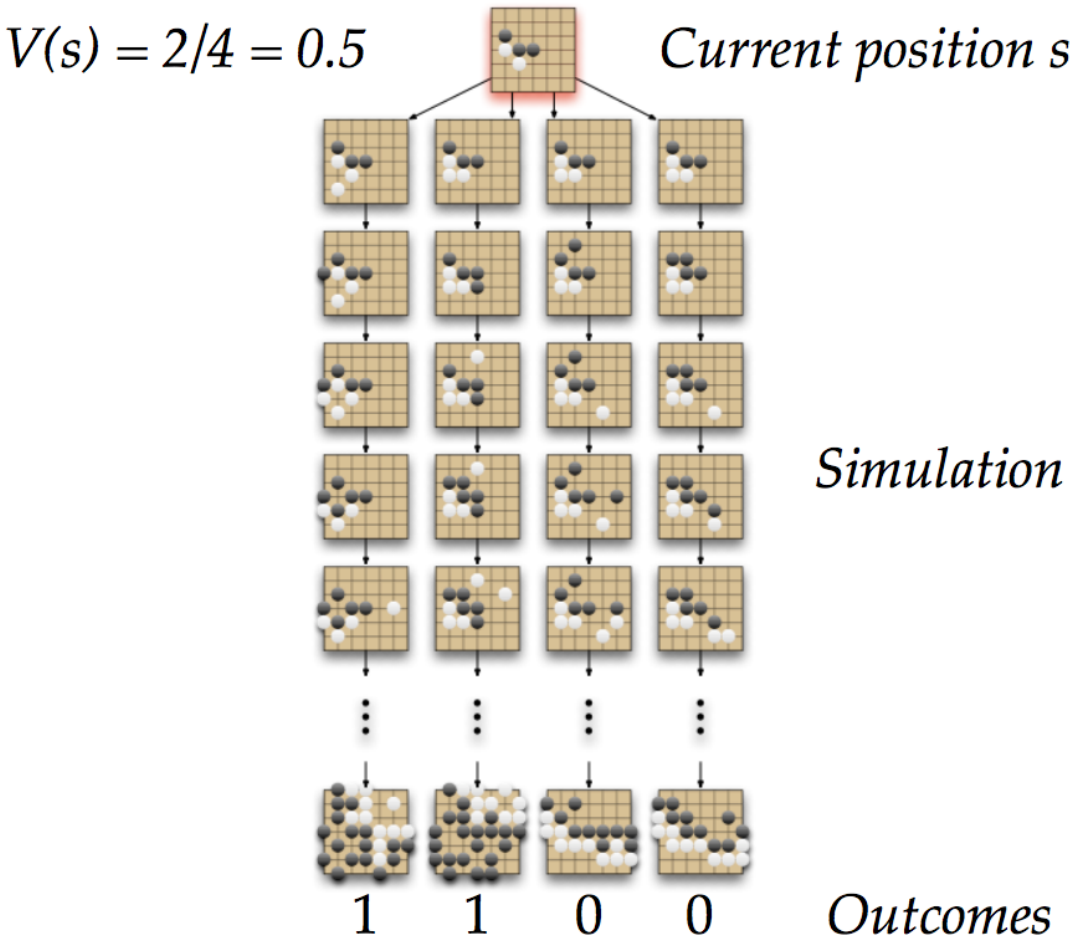
Policy $\pi =< \pi_B, \pi_W >$ selects moves for both players

Value function (how good is position s):

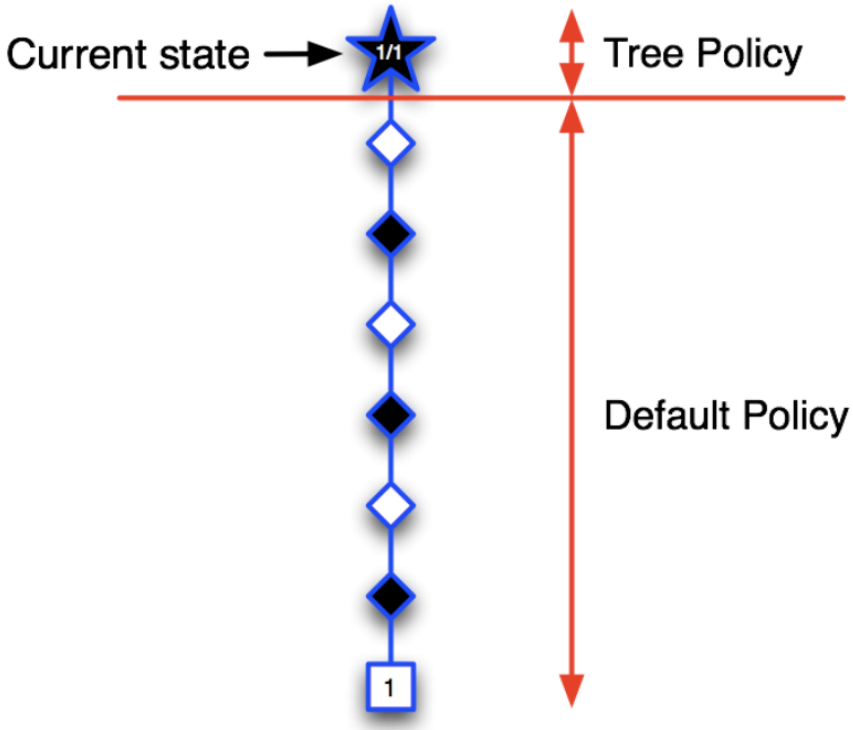
$$v_\pi(s) = \mathbb{E}_\pi[R_T \mid S = s] = \mathbb{P}[\text{Black wins} \mid S = s]$$

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

Monte-Carlo Evaluation in Go



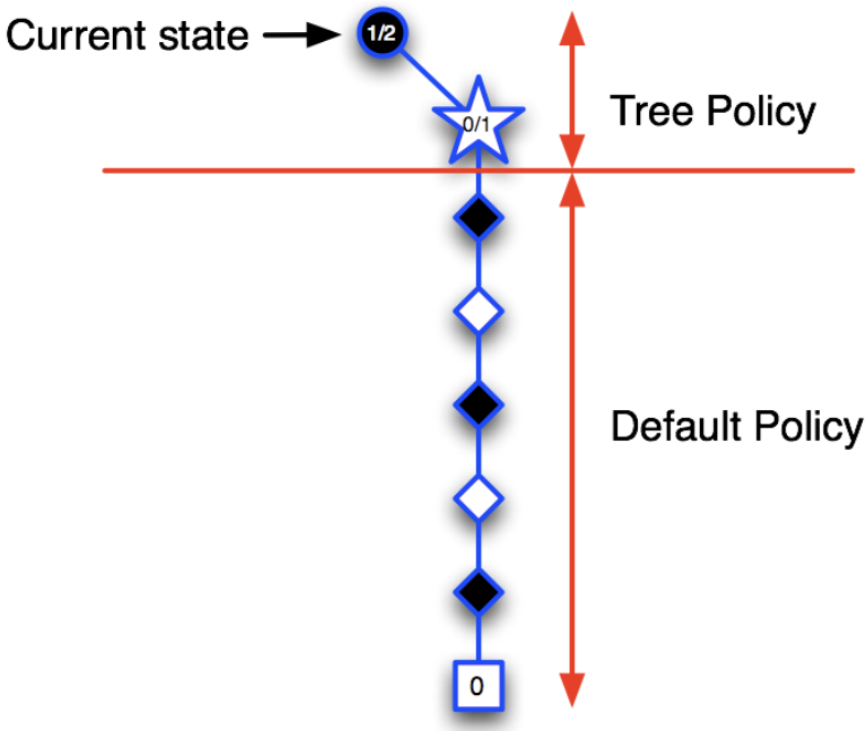
Applying Monte-Carlo Tree Search (1)



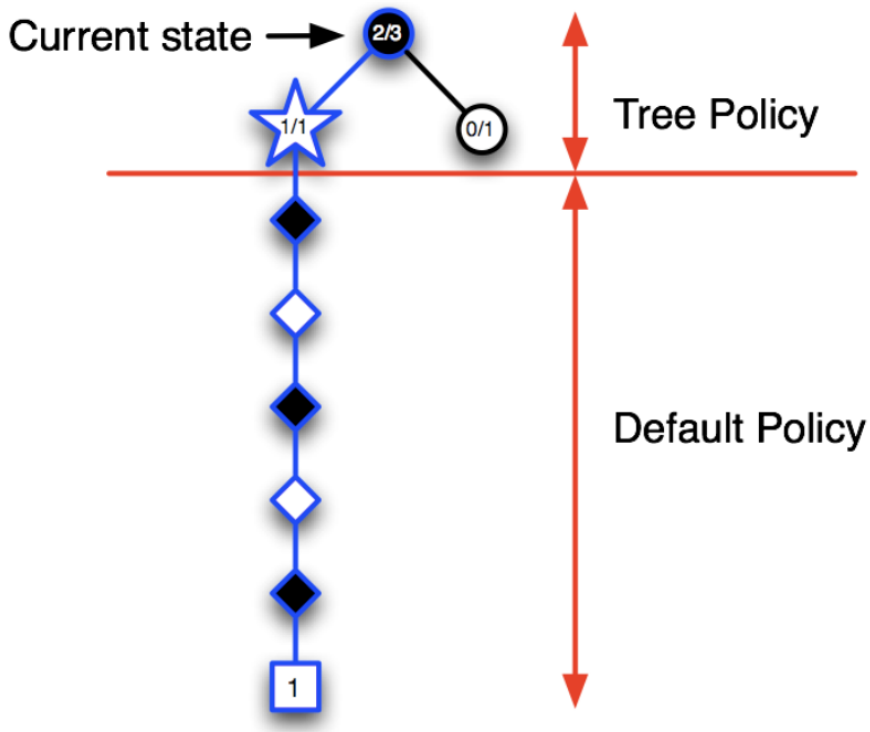
Go is a 2 player game so tree is a minimax tree instead of expectimax

White minimizes future reward and Black maximizes future reward when computing action to simulate

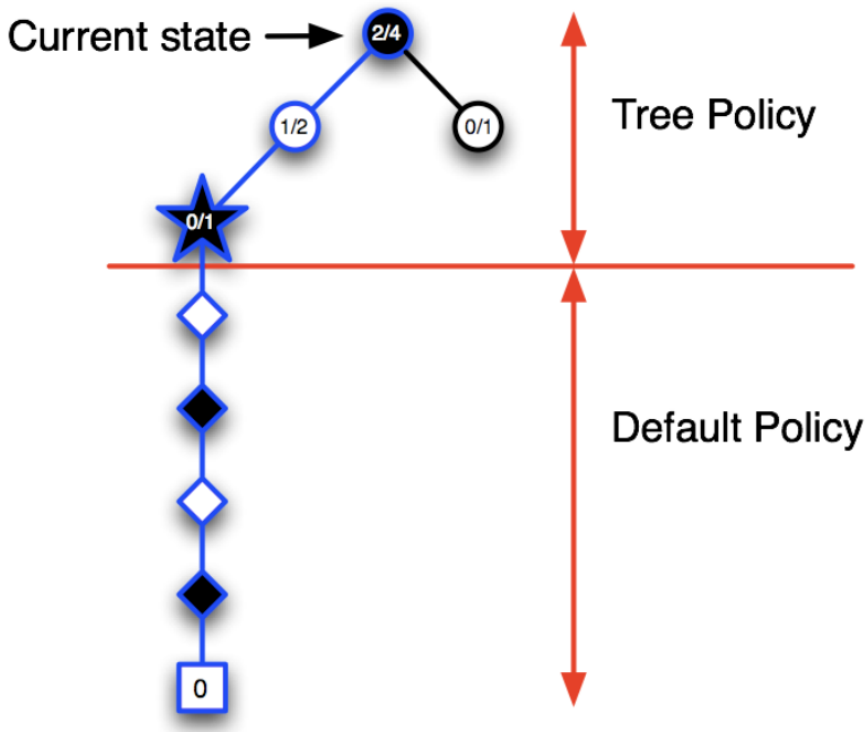
Applying Monte-Carlo Tree Search (2)



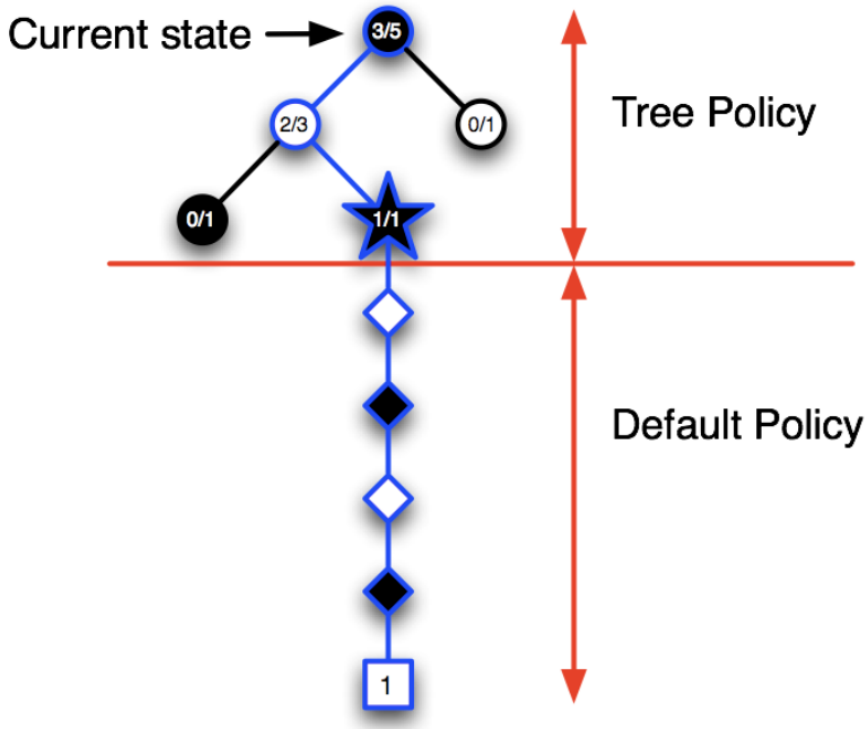
Applying Monte-Carlo Tree Search (3)



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]

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

Check Your Understanding: UCT Search

In Upper Confidence Tree (UCT) search we treat each tree node as a multi-armed bandit (MAB) problem, and use an upper confidence bound over the future value of each action to help select actions for later rollouts. Select all that are true

- This may be useful since it will prioritize actions that lead to later good rewards
- UCB minimizes regret. UCT is minimizing regret within rollouts of the tree. (If this is true, think about if this a good idea?)
- Not sure

Check Your Understanding: UCT Search Solutions

In Upper Confidence Tree (UCT) search we treat each tree node as a multi-armed bandit (MAB) problem, and use an upper confidence bound over the future value of each action to help select actions for later rollouts. Select all that are true

- This may be useful since it will prioritize actions that lead to later good rewards
- UCB minimizes regret. UCT is minimizing regret within rollouts of the tree. (If this is true, think about if this a good idea?)
- Not sure

T. T (but not a good idea)

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

Hint: why were upper confidence bounds a good idea for exploration/ exploitation? Is there an exploration/ exploitation problem during simulated episodes?[2]

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

AlphaGo

▸ AlphaGo trailer link

MCTS is part of what lead to success in Go

A number of key additional choices

There are an enormous number of possible moves. Prioritizing which actions to take when branching matters. In AlphaGo this was initially done by training a supervised learning NN to mimic moves made by humans.

Instead of rolling all the way out to the leaves (game won or lost), one can bootstrap and substitute an estimate of the future value. They did this in alphaGo and this also made a big difference.

For training this value estimate they used **self-play**. This was extremely useful. Self-play in games means that an agent will frequently win and lose because the other agent it is playing is of a similar difficulty level. This is very helpful for providing enough reward signal.

Beyond AlphaGo

There have been several more impressive insights that don't rely on human knowledge:

- Mastering the game of Go without human knowledge. Silver et al. Nature 2017.
- A general RL algorithm that masters chess, shogi, and Go through self-play. Silver et al. Science 2018.

What You Should Know

MCTS computes a decision only for the current state

It is an alternative to value iteration or policy iteration and can be used in extremely large state spaces

UCT uses optimism under uncertainty to choose to further expand action nodes that seem promising

MCTS has been used with RL to create an AI Go player that exceeds the best human performance

Class Structure

Last time: Quiz

This Time: MCTS