Final Lecture: MCTS [1]

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2019

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

Class Logistics

- Project write up due: March 20 11:59pm (no late days)
- Poster presentations: March 22 8:30am (and submit poster pdf online for 8:30am too) (no late days)
- Any questions, reach out to us on piazza or come by office hours this week!

Class Structure

- Last time: Quiz
- **This Time: MCTS and end of course**

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

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

T and/or R model

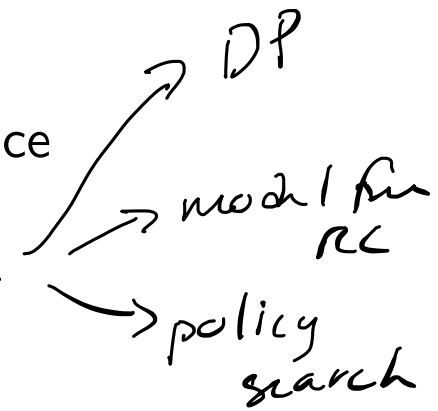
once we have a model we can use planning

Model-Based and Model-Free RL

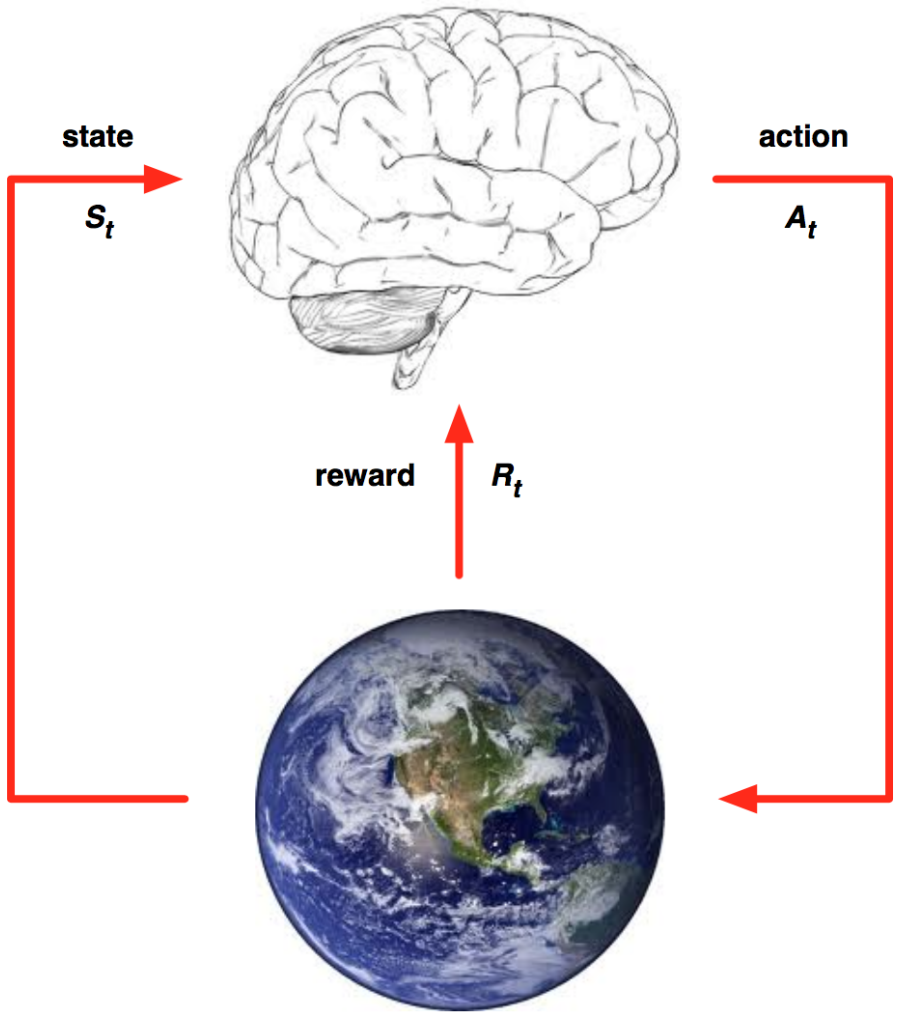
- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience

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

in a lot of cases you'll know the reward function

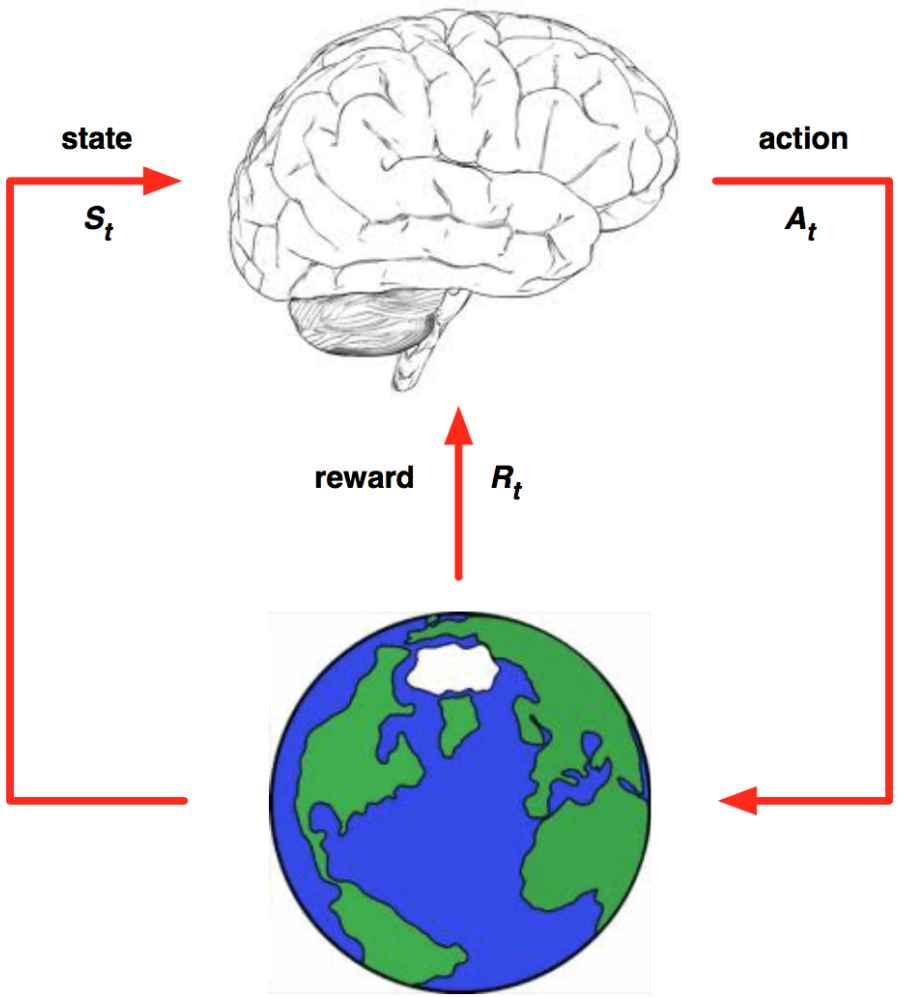
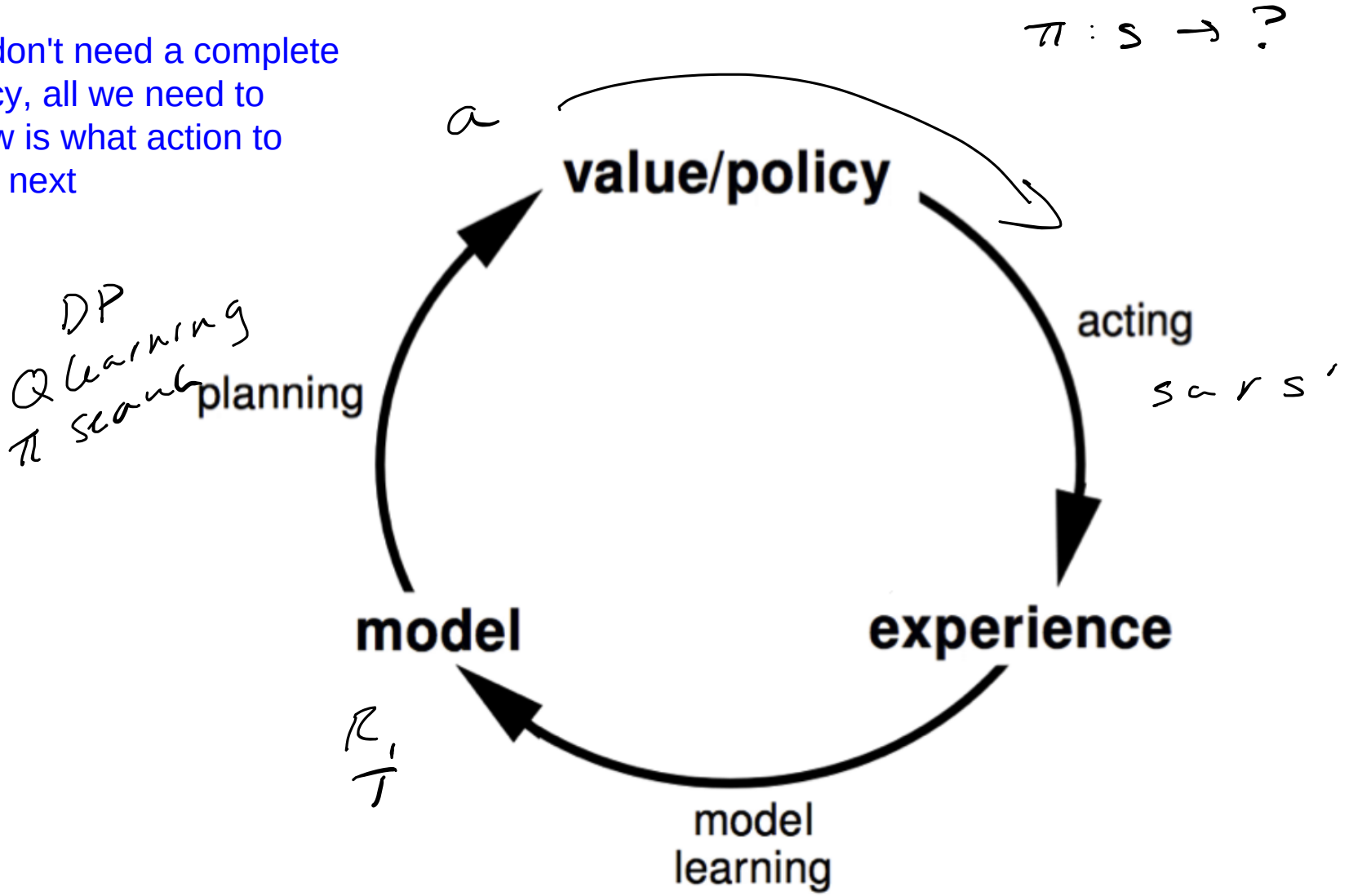


Table of Contents

Model-Based RL

we don't need a complete policy, all we need to know is what action to take next



Advantages of Model-Based RL

bandit uncertainty reward

- 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 $\mathcal{R}_\eta \approx \mathcal{R}$

Markov ✓

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

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

- 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

inputs --> outputs

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

- 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
- ... Bayesian DNN

Table Lookup Model

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

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

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

- 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

6 times we started in state B

single action MRP

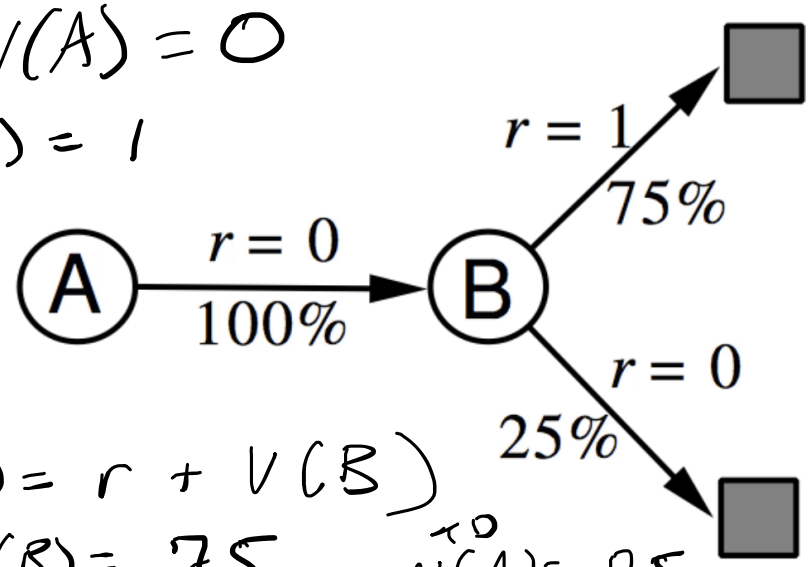
- Two states A,B; no discounting; 8 episodes of experience

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

6

$V(A) = 0$

$T(B|A) = 1$



$V(A) = r + V(B)$

$V(B) = .75$

so $V(A) = .75$

- 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?

min MSE does not assign Markov

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

can generate fake data to train with

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

$$\hat{T}(B|A) = 1$$

$$A \rightarrow B$$

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

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

why would you first build a model just to do model-free? We will see...

- 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

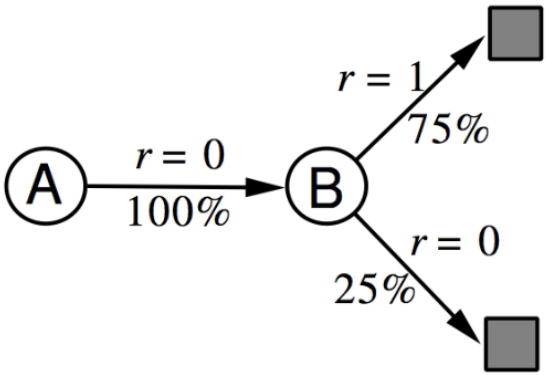
Markov

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

simulated data

Real experience

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



Sampled experience

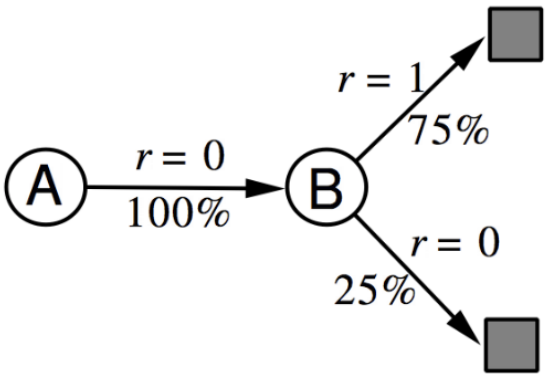
B, 1
B, 0
A, 0, B, 1

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

we calculate this from the sampled data

- 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? V(A) = 0 V(B) = .75

Planning with an Inaccurate Model

$V(s, a_1) > V(s, a_2)$

- 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 (Does this solve the issue?) depends on why

Planning with an Inaccurate Model

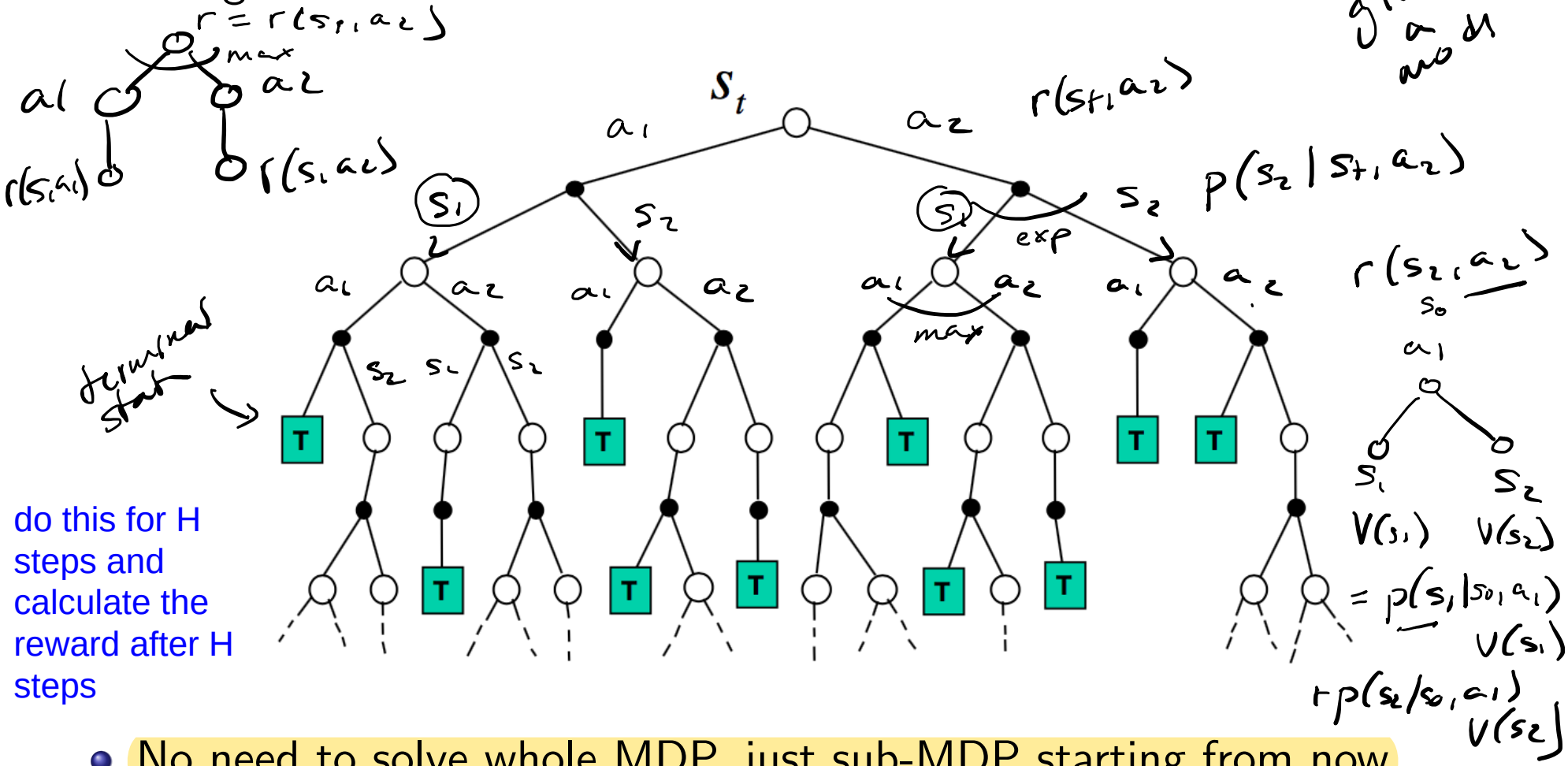
- Given an imperfect model $< \mathcal{P}_\eta, \mathcal{R}_\eta > \neq < \mathcal{P}, \mathcal{R} >$
- Performance of model-based RL is limited to optimal policy for approximate MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta >$
- i.e. Model-based RL is only as good as the estimated 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

Forward Search

with our model we can use Forward Search to make decisions

- 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

Simulation-Based Search

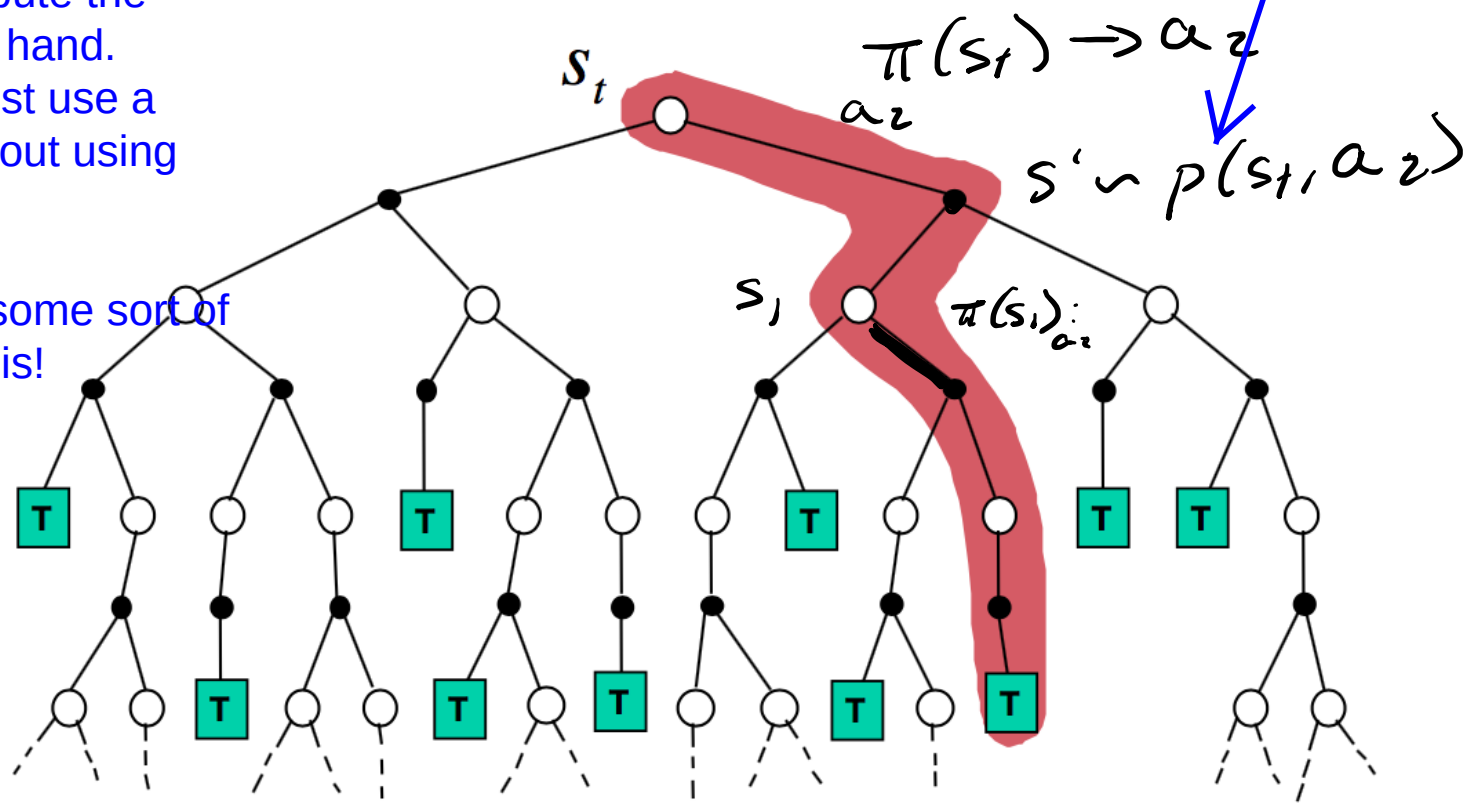
sample this from our model

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

π

we don't compute the values before hand. Instead, we just use a policy and rollout using our model.

But we need some sort of policy to do this!



Simulation-Based Search (2)

- Simulate episodes of experience from now with the model

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

- 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)

for each possible action from state s, we do multiple rollouts. Then we can calculate rewards and get Q(s, a)

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

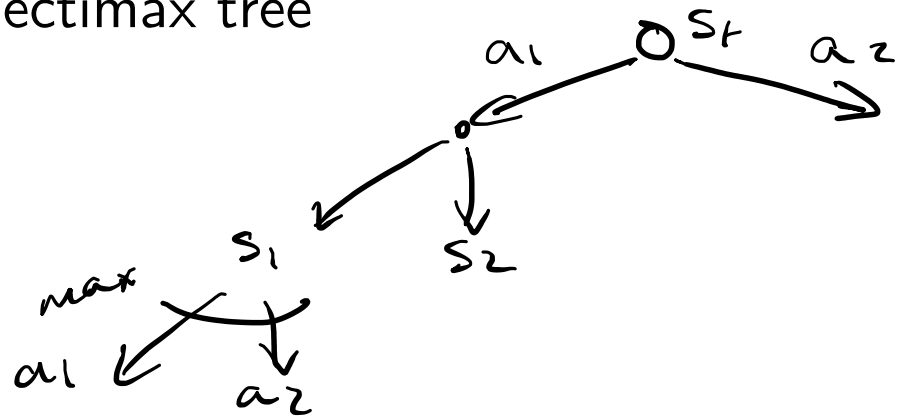
- Select current (real) action with maximum value

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

- This is essentially doing 1 step of policy improvement

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

Expectimax Tree

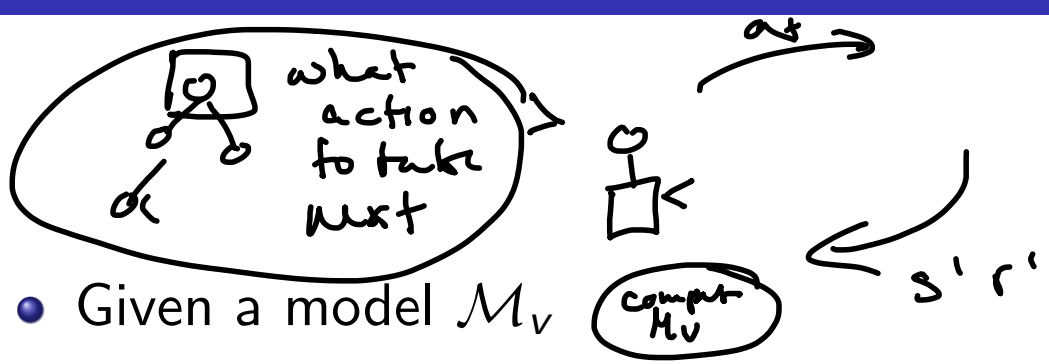
- Can we do better than 1 step of policy improvement?
- If have a MDP model $\mathcal{M}_v$
- Can compute optimal $q(s, a)$ values for current state by constructing an expectimax tree

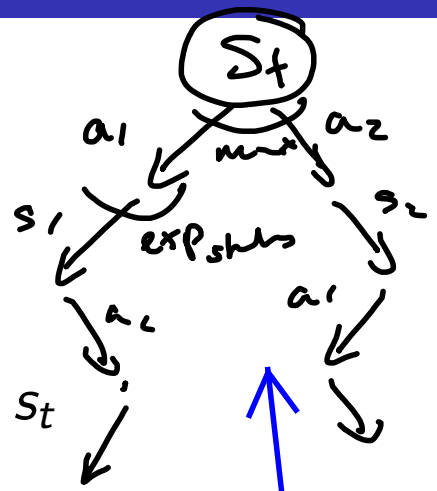
we want to be able to use simulation and also compute expectimax trees. But computing full expectimax trees is intractable for all but small contrived problems. So what can we do that's better?

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

[1] With 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

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

slowly fill in the expectimax tree using simulation. Once your done, then you can compute the expectimax

note that after your done with simulation, part of the tree will likely still be missing! So you never get a full expectimax tree.

Monte-Carlo Tree Search

What do you do if you reach a node where you've already attempted both actions. Which should you pick again? What do you do if you reach a node that hasn't been tried.

- 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

if you're never been to that state before, do a roll out policy (i.e. pick randomly)

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

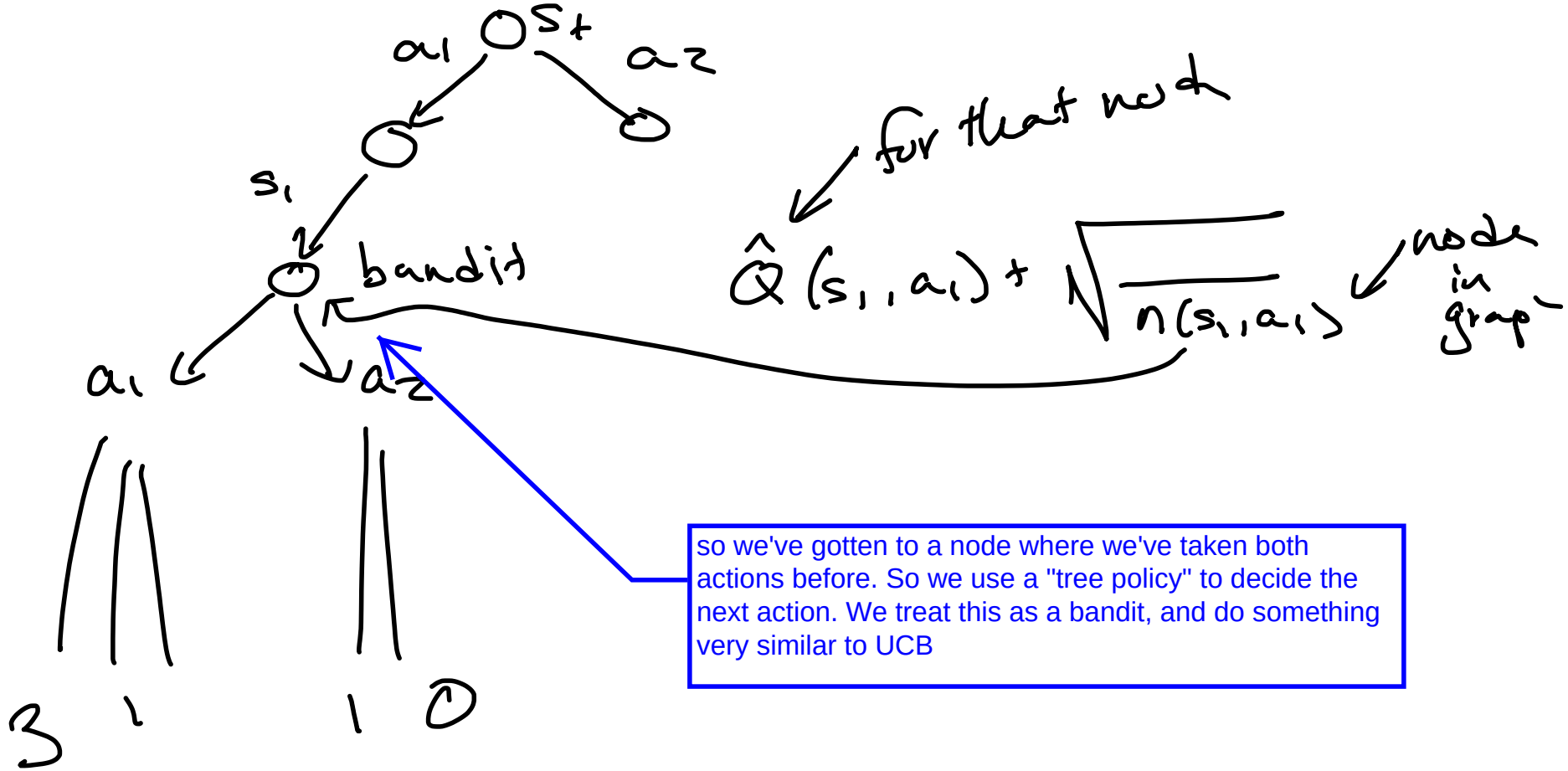
- Under mild conditions, converges to the optimal search tree, $Q(S, A) \rightarrow q^*(S, A)$

note that we average, instead of take the max

Upper Confidence Tree (UCT) Search

tree policy

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



so we've gotten to a node where we've taken both actions before. So we use a "tree policy" to decide the next action. We treat this as a bandit, and do something very similar to UCB

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

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

bandit

- 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

i think we care about node i because in tree methods, we can get to the same state 2+ times, but we treat them separately

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

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

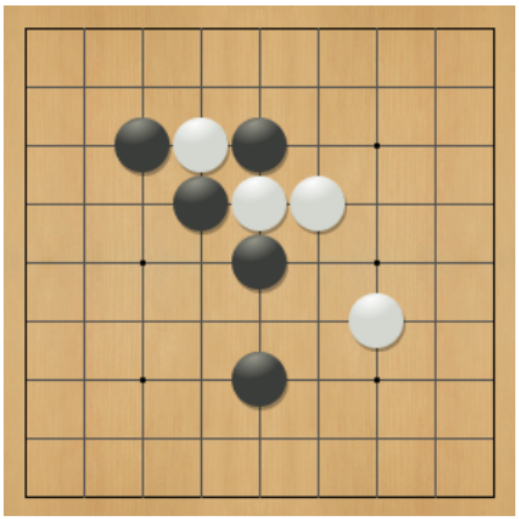
Case Study: the Game of Go

- 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

finite horizon

- 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):

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

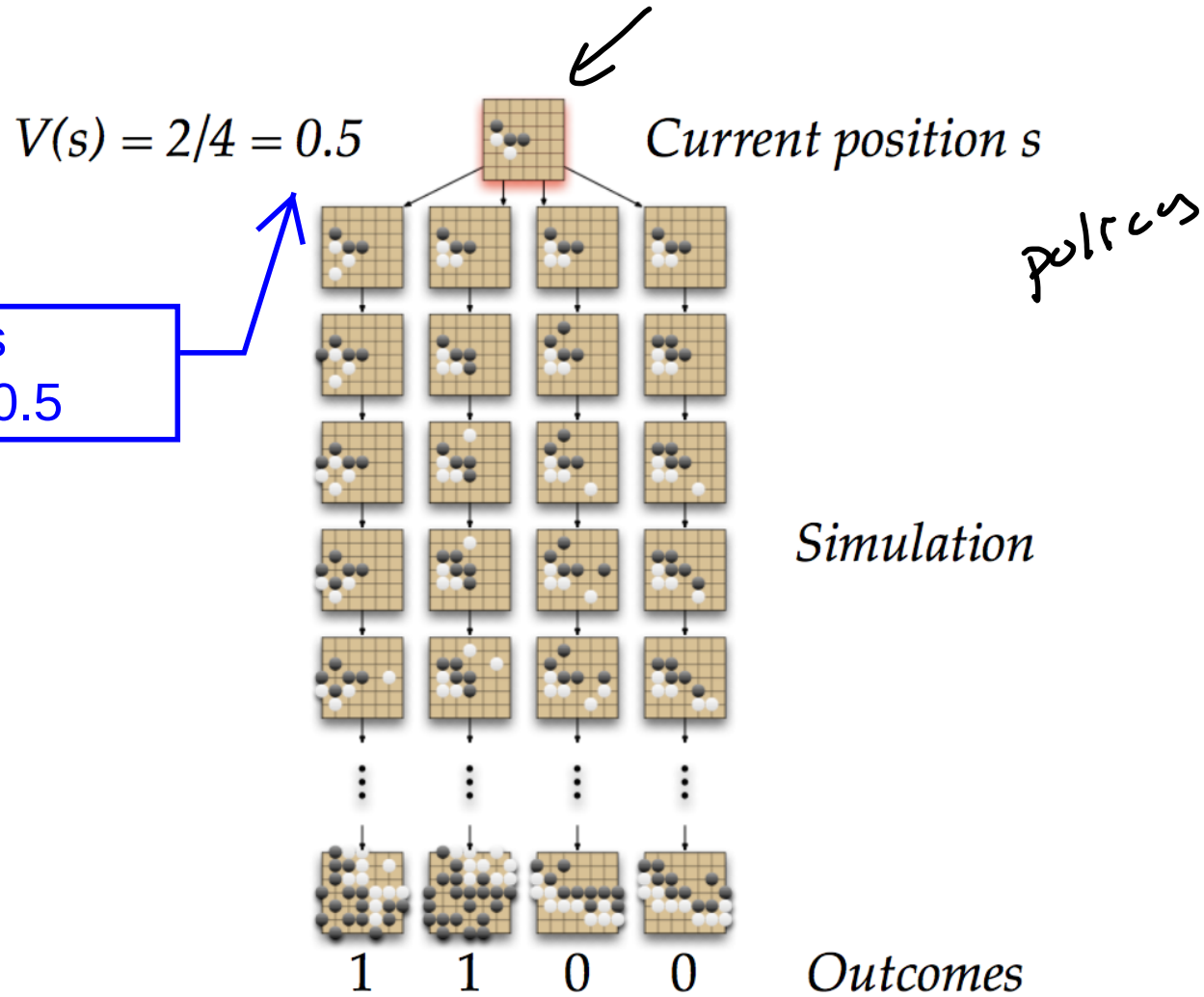
very sparse rewards

$$R_T = \begin{cases} 1, & \text{if Black wins.} \\ 0, & \text{if White wins.} \end{cases} \tag{5}$$

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

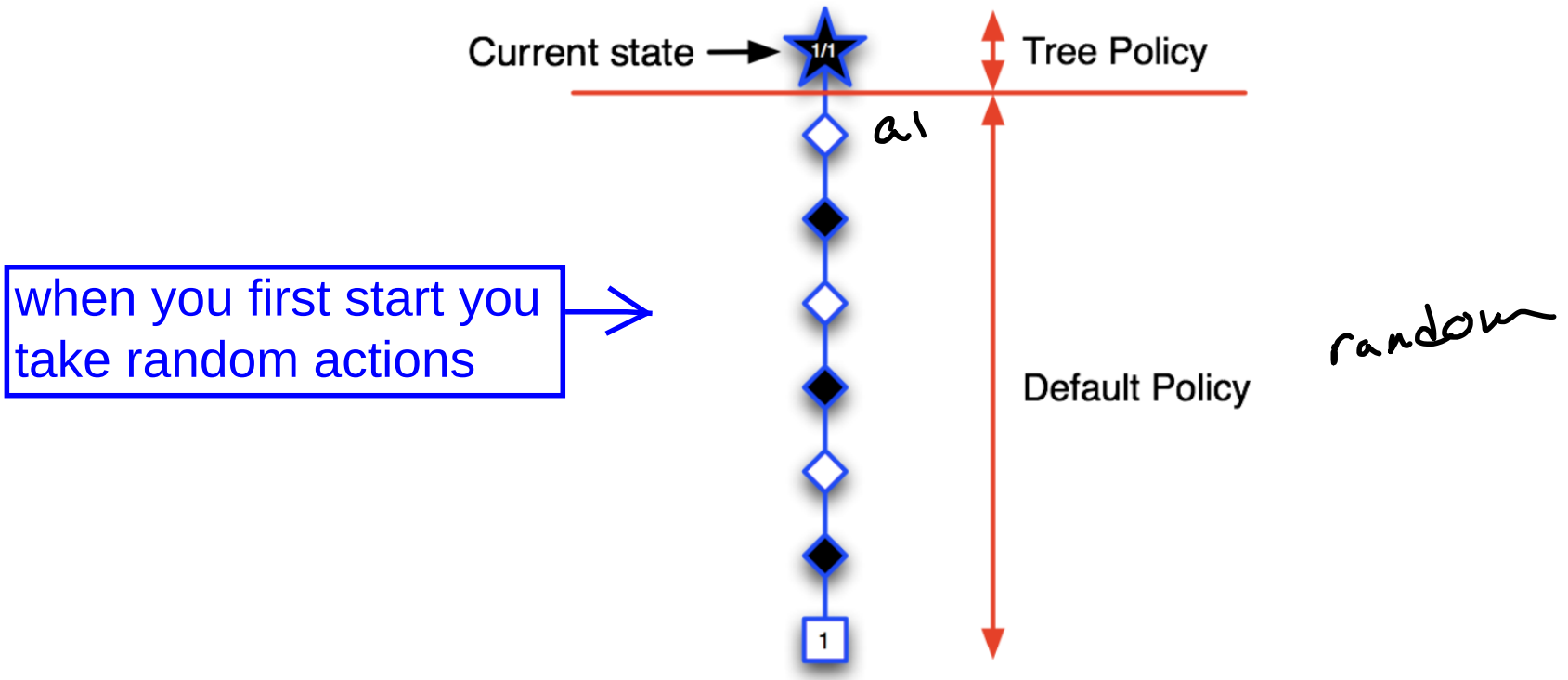
$$v_\pi(s) = \mathbb{E}_\pi[R_T \mid S = s] = \mathbb{P}[\text{Black wins} \mid S = s]$$
$$v^*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

Monte-Carlo Evaluation in Go



the value for this starting state is 0.5

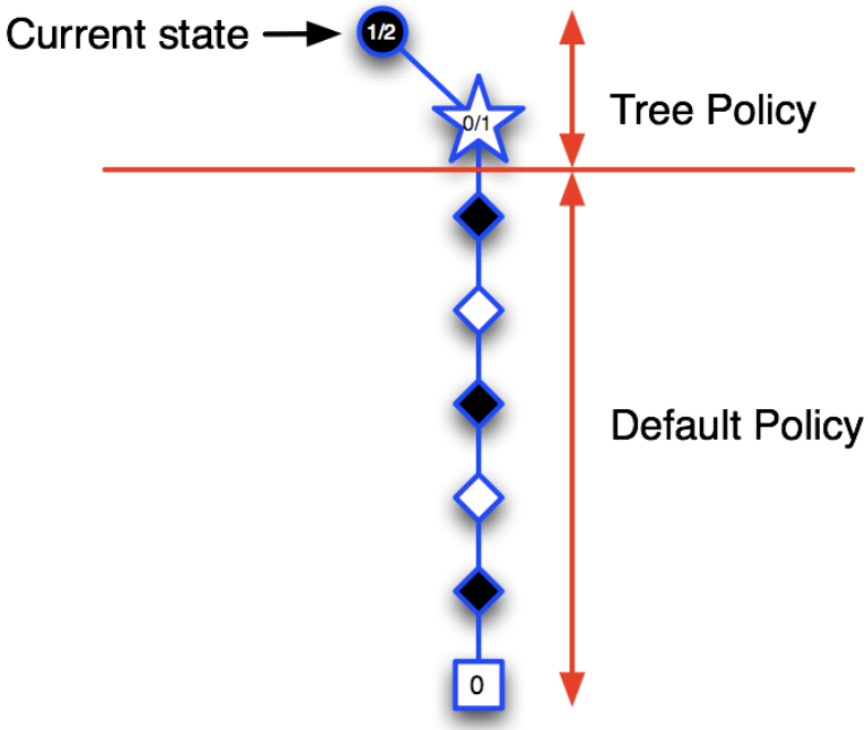
Applying Monte-Carlo Tree Search (1)



- 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

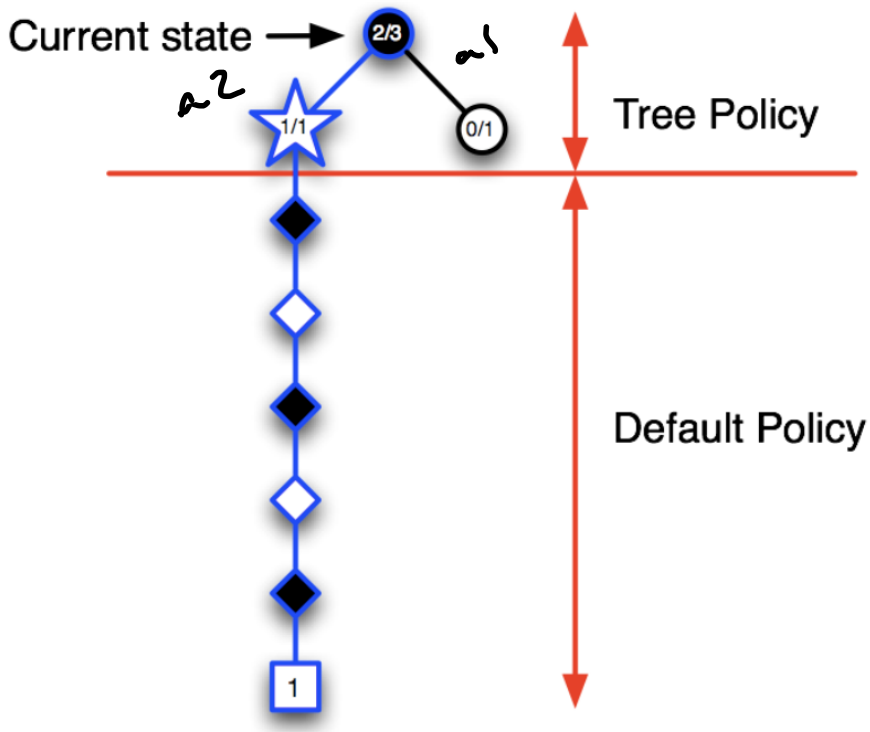
Applying Monte-Carlo Tree Search (2)

2nd rollout



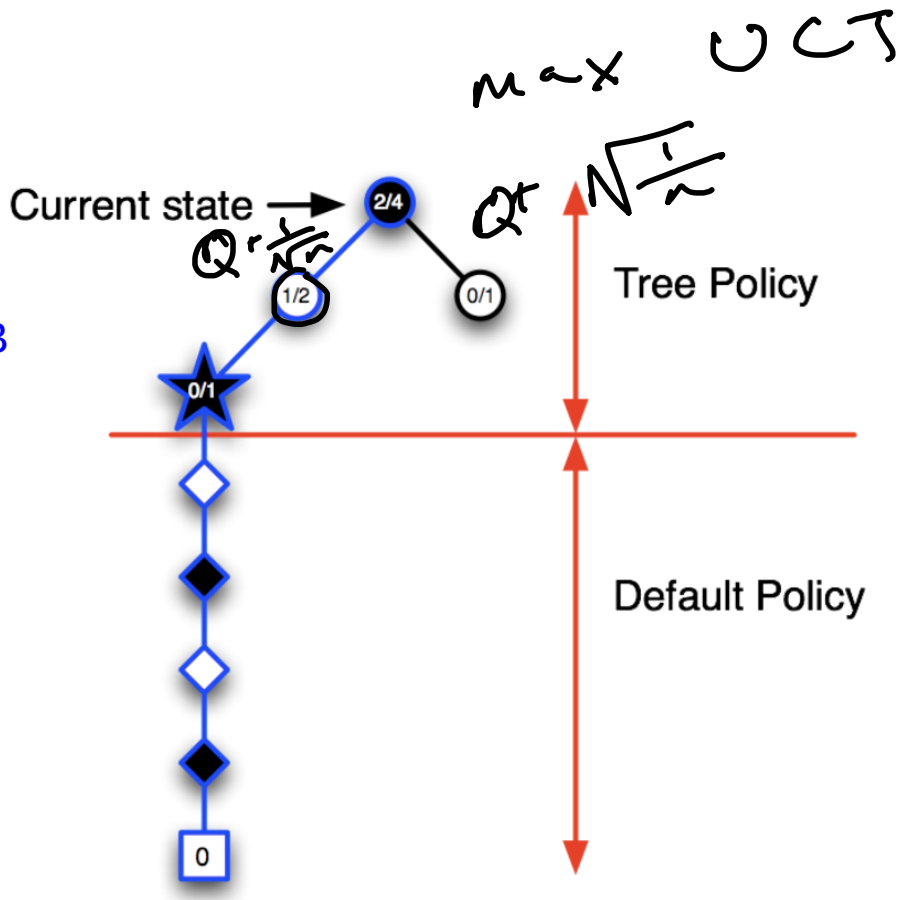
Applying Monte-Carlo Tree Search (3)

generally you want to try all the actions from a particular node

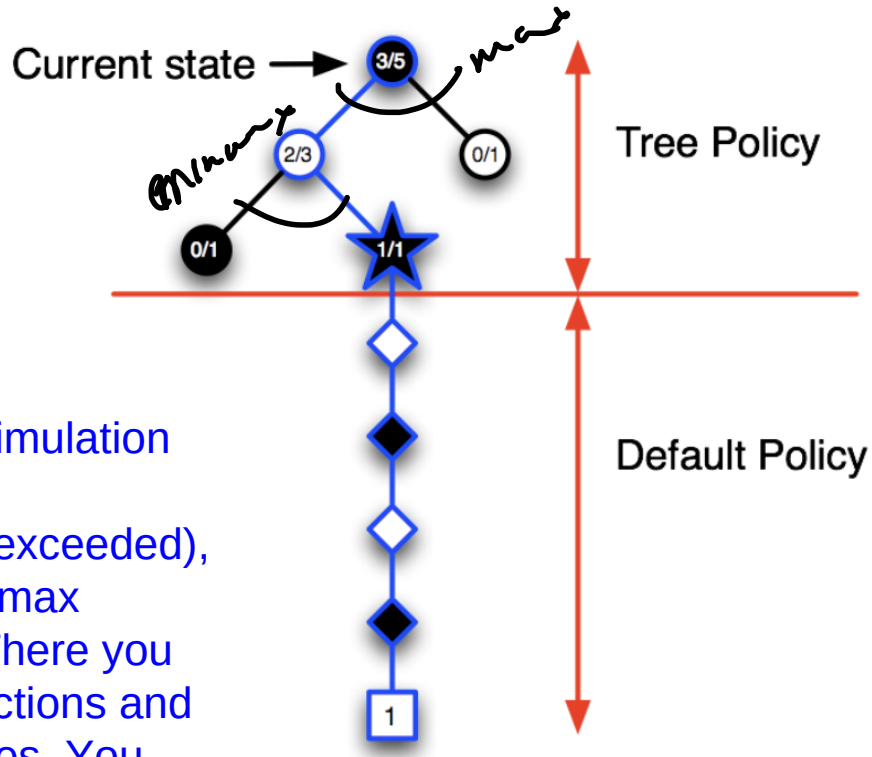


Applying Monte-Carlo Tree Search (4)

now that we've tried all the actions, we treat it as a MAB and select which action to take based on their UCB



Applying Monte-Carlo Tree Search (5)



okay, after you run this simulation "enough" times (or your computational budget is exceeded), you calculate the expectimax starting at the terminal. There you calculate the max over actions and the expectation over states. You work your way from bottom to top.

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)

Class Structure

- Last time: Quiz
- **This Time: MCTS and end of course**

End of Class Goals

- Define the key features of reinforcement learning that distinguish it from AI and non-interactive machine learning

End of Class Goals

- Define the key features of reinforcement learning that distinguish it from AI and non-interactive machine learning
- Given an application problem (e.g. from computer vision, robotics, etc) decide if it should be formulated as a RL problem, if yes be able to define it formally (in terms of the state space, action space, dynamics and reward model), state what algorithm (from class) is best suited to addressing it, and justify your answer.

End of Class Goals

- Define the key features of reinforcement learning that distinguish it from AI and non-interactive machine learning
- Given an application problem (e.g. from computer vision, robotics, etc) decide if it should be formulated as a RL problem, if yes be able to define it formally (in terms of the state space, action space, dynamics and reward model), state what algorithm (from class) is best suited to addressing it, and justify your answer.
- Implement (in code) common RL algorithms including a deep RL algorithm
- Describe (list and define) multiple criteria for analyzing RL algorithms and evaluate algorithms on these metrics: e.g. regret, sample complexity, computational complexity, empirical performance, convergence, etc.
- Describe the exploration vs exploitation challenge and compare and contrast at least two approaches for addressing this challenge (in terms of performance, scalability, complexity of implementation, and theoretical guarantees

Learning more about RL

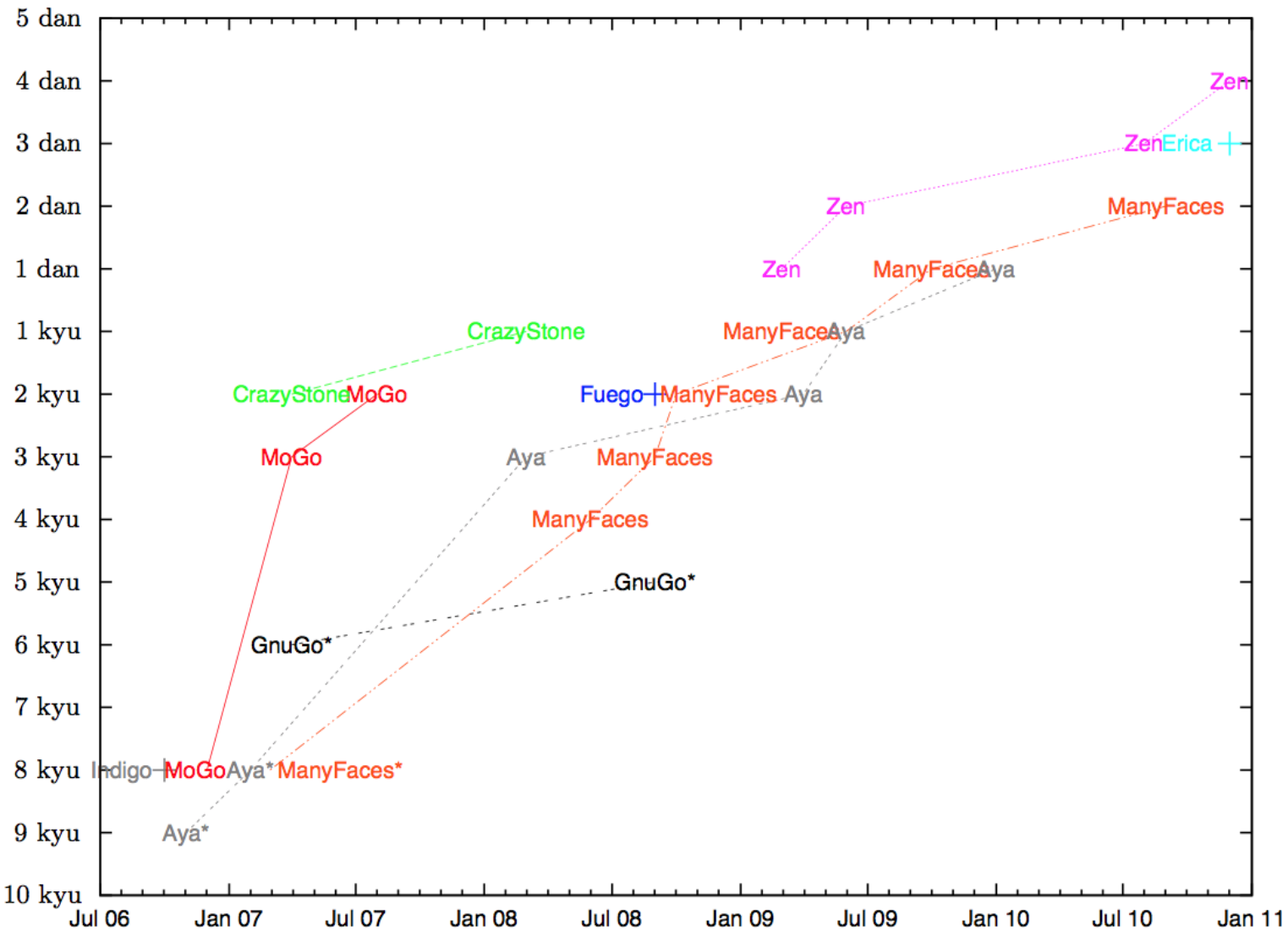
- Sequential decision making under uncertainty
- CS238: Decision Making under Uncertainty
- CS239: Advanced Topics in Sequential Decision Making
- MSE351 Dynamic Programming and Stochastic Control
- MSE338 Reinforcement Learning (advanced version)
- CS332: Advanced Survey of Reinforcement Learning (current topics, project class)

Reinforcement learning

- Already seeing incredible results in games and some terrific successes in robotics
- Healthcare, education, consumer marketing...
- Machines learning to help us, in safe, fair and accountable ways

Getting Your Feedback

MCTS and Early Go Results



MCTS Variants

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

MCTS Variants

- 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