Integrating Learning and Planning

Spring, 2021

Institute of Media, Information, and Network

Outline

Table of Contents

Model-Based Reinforcement Learning

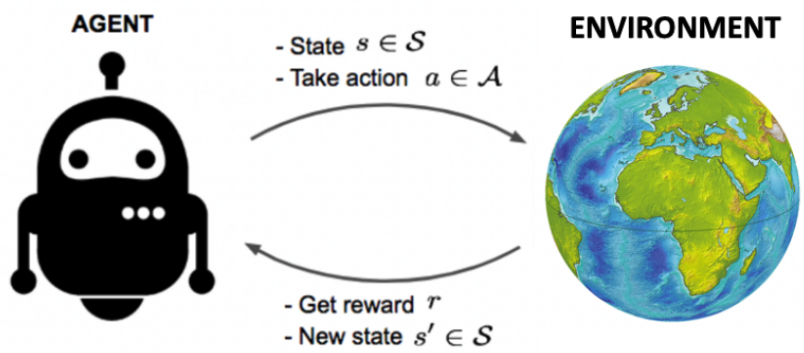
- *Previous lectures*: learn policy or value function directly from experience
- This lecture: learn model directly from experience
  - and use planning to construct a value function or policy
  - Integrate learning and planning into a single architecture

Model-Based and Model-Free RL

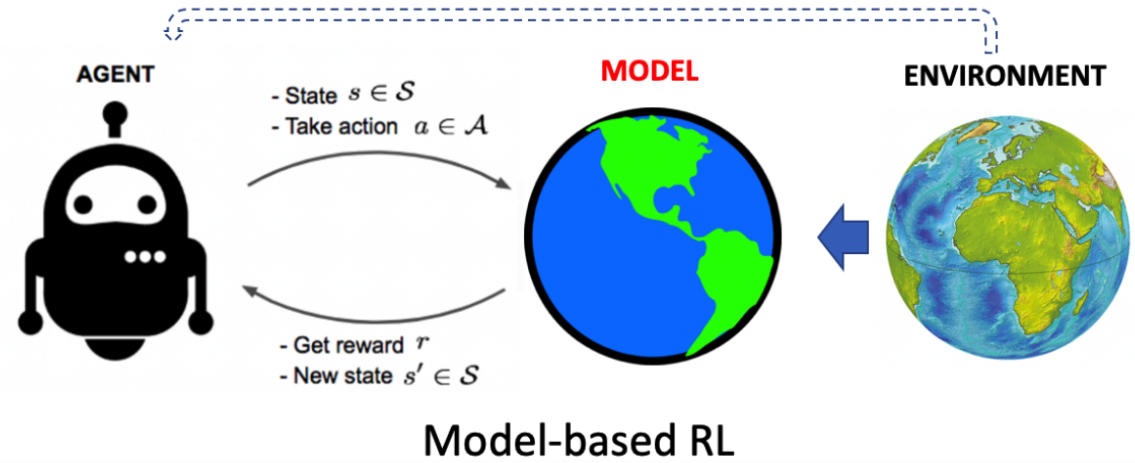
- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
  - High sample complexity
- Model-Based RL
  - Learn a model from experience
  - Plan value function (and/or policy) from model
  - Better sample efficiency

Model-Based and Model-Free RL

1. Diagram of model-free reinforcement learning



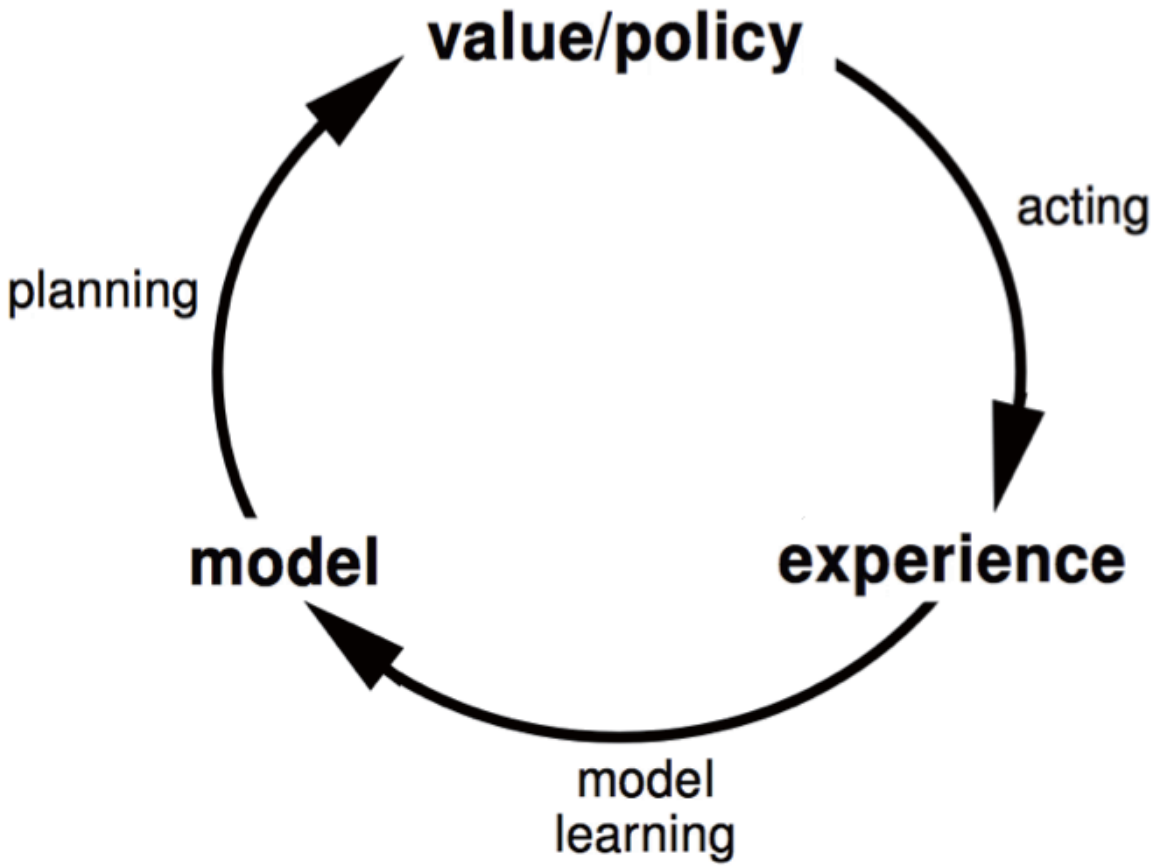
2. Diagram of model-based reinforcement learning



Iedia,
and Network

Model-Based RL

- Planning is "trying things in your head", using an internal model of the world.



Institute of Media, Information, and Network

Modeling the Environment for Planning

1. Planning is the computational process that takes a model as input and produces or improves a policy by interacting with the modeled environment

$$\text{experience} \xrightarrow{\text{learning}} \text{model} \xrightarrow{\text{planning}} \text{better policy}$$

2. State-space planning: search through the state space for an optimal policy or an optimal path to a goal
3. Model-based value optimization methods share a common structure

$$\text{model} \rightarrow \text{simulated trajectories} \xrightarrow{\textit{backups}} \text{values} \rightarrow \text{policy}$$

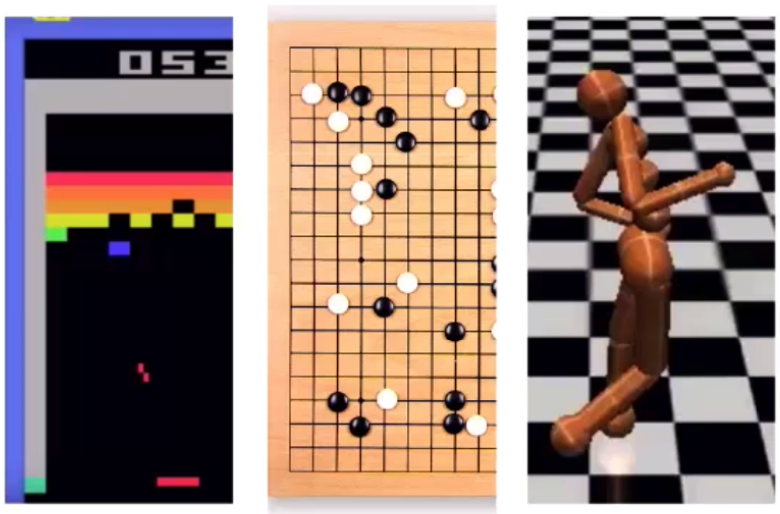
4. Model-based policy optimization methods have a simpler structure as

$$\text{model} \rightarrow \text{policy}$$

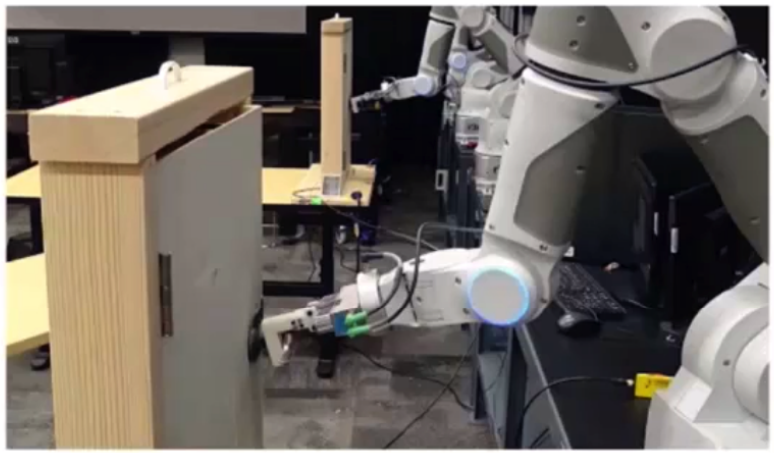
Advantages of Model-Based RL

1. Pros: Better sample efficiency

Simulation



Real-world



1. Sample-efficient learning is crucial for real-world RL applications such as robotics
   DARPA robotics failure
2. Model can be learned efficiently by supervised learning methods

work

Advantages of Model-Based RL

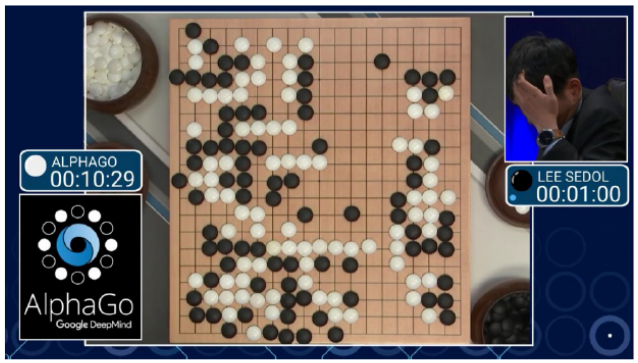
1. Better sample efficiency

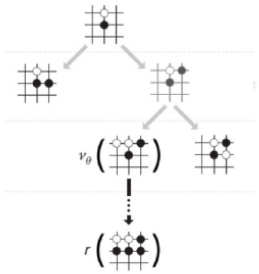
2. Cons:
   1. First learning a model then constructing a value function or policy function leads to two sources of approximation error
   2. Difficult to come up with guarantee of convergence

Institute of Media, Information, and Network

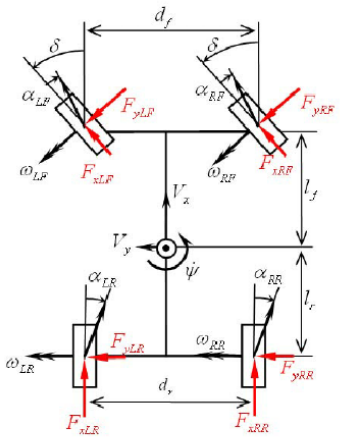
Sometimes it is easy to access the model

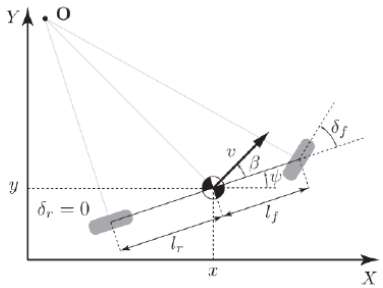
1. Known models: Game of Go: the rule of the game is the model





2. Physics models: Vehicle dynamics model and kinematics bicycle model





Network

Table of Contents

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

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

- Typically assume conditional independence between state transitions and rewards

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

Model Learning

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

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

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

Example of Models

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

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

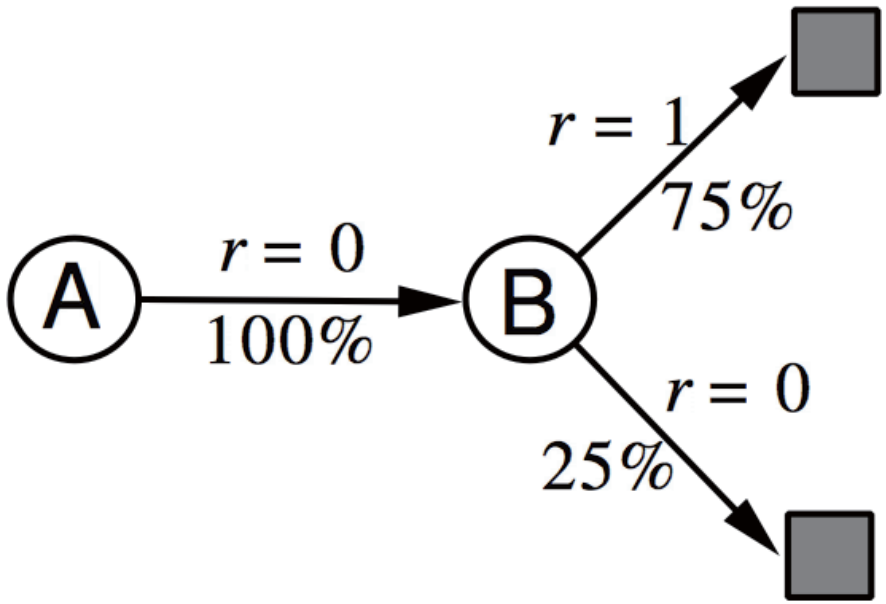
$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, = s, a) R_t$$

- 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

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

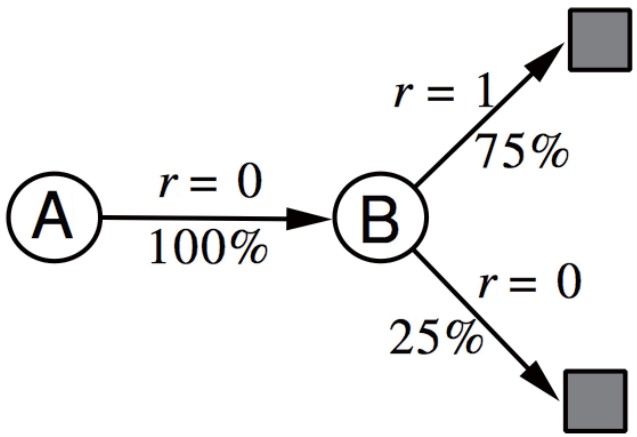
- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- Sample-based planning methods are often more efficient

Back to the AB Example

- Construct a table-lookup model from real experience
- Apply model-free RL to sample 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: $V(A) = 1, V(B) = 0.75$

Sometimes it is easy to access the model

1. Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$
2. Performance of model-based RL is limited to the optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
    1. Model-based RL is only as good as the estimated model
3. When the model is inaccurate, planning process will compute a suboptimal policy
4. Possible solutions:
    1. When the accuracy of the model is low, use model-free RL
    2. Reason explicitly about the model uncertainty (how confident we are for the estimated state): Use probabilistic model such as Bayesian and Gaussian Process

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)

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

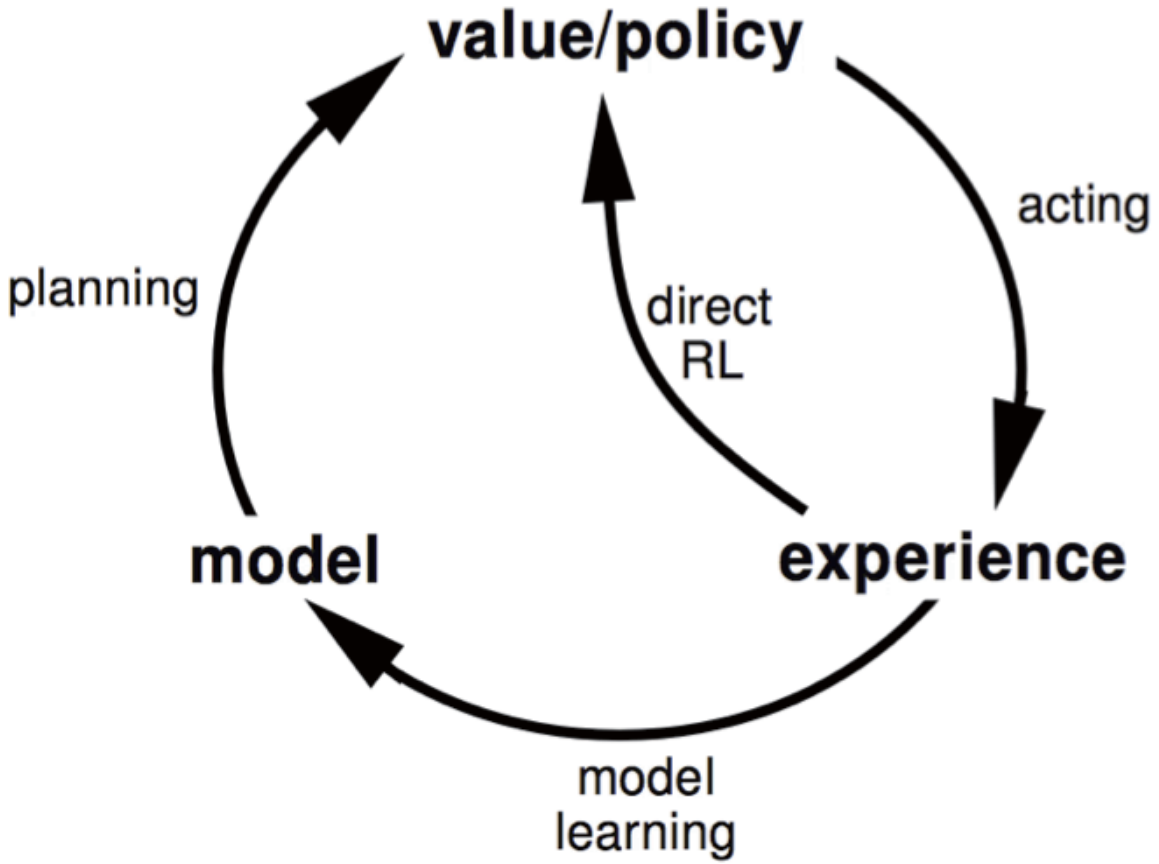
Institute of Media, Information, and Network

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

R. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, AAAI 1991.



Institute of Media, Information, and Network

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:

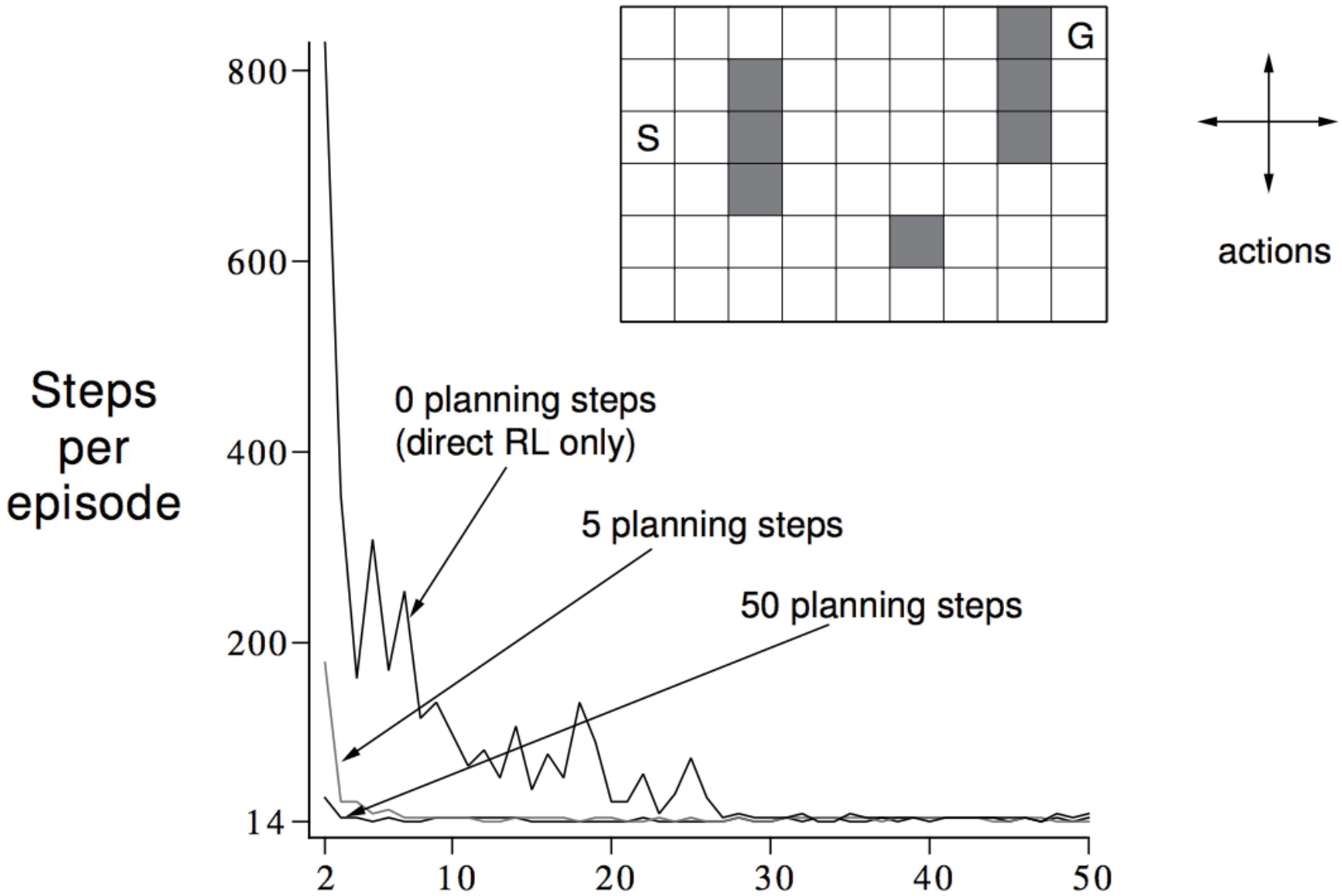
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

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

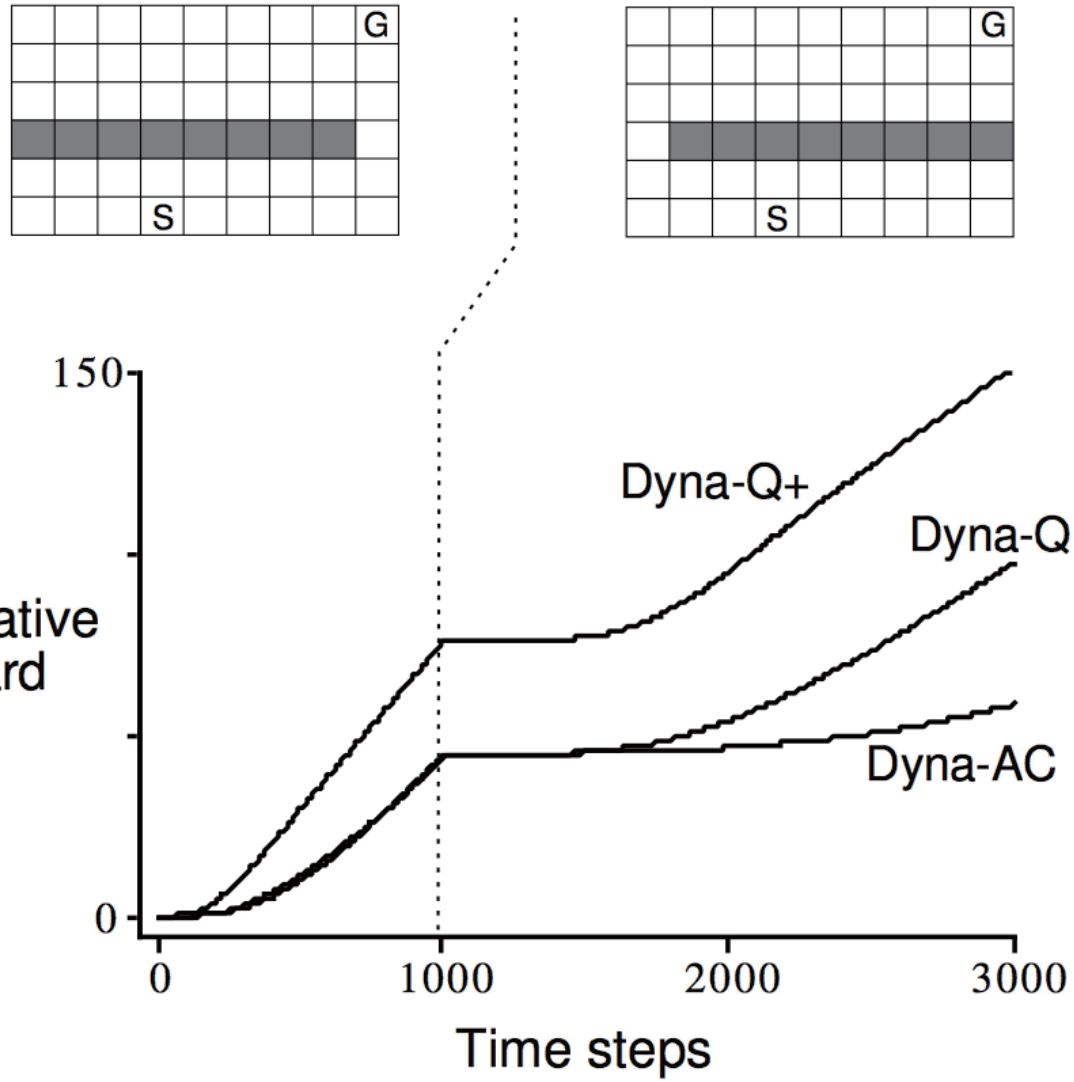
$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

Dyna-Q on a Simple Maze



Dyna-Q with an Inaccurate Model

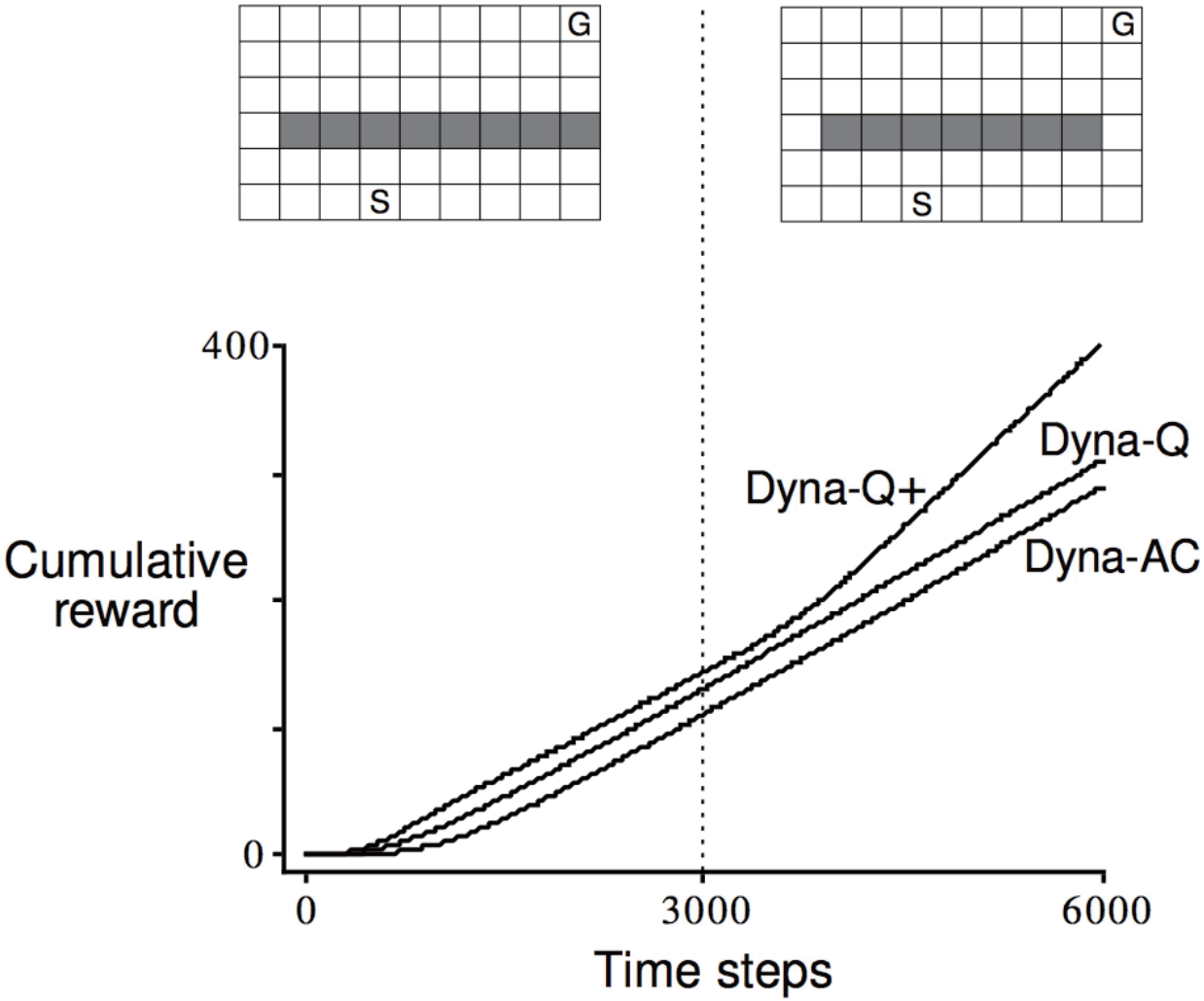
- The changed environment is harder



stitute of Media,
formation, and Network

Dyna-Q with an Inaccurate Model (2)

- The changed environment is easier



nstitute of Media,
nformation, and Network

Table of Contents

Policy Optimization with Model-based RL

1. Previous model-based value-based RL:

$$\text{model} \rightarrow \text{simulated trajectories} \xrightarrow{backups} \text{values} \rightarrow \text{policy}$$

2. Can we optimize the policy and learn the model directly, without estimating the value?

$$\text{model} \xrightarrow{improves} \text{policy}$$

Model-based Policy Optimization in RL

1. Policy gradient, as a model-free RL, only cares about the policy $\pi_\theta(a_t|s_t)$ and expected return

$$\tau = \{s_1, a_1, s_2, a_2 \ldots, s_T, a_T\} \sim \pi_\theta(a_t|s_t)$$
$$\arg\max_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t \gamma^t r(s_t, a_t)\right]$$

2. In policy gradient, no $p(s_{t+1}|s_t, a_t)$ is needed (no matter it is known or unknown)

$$p(s_1, a_1, \ldots, s_t, a_T) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

3. But can we do better if we know the model or are able to learn the model?

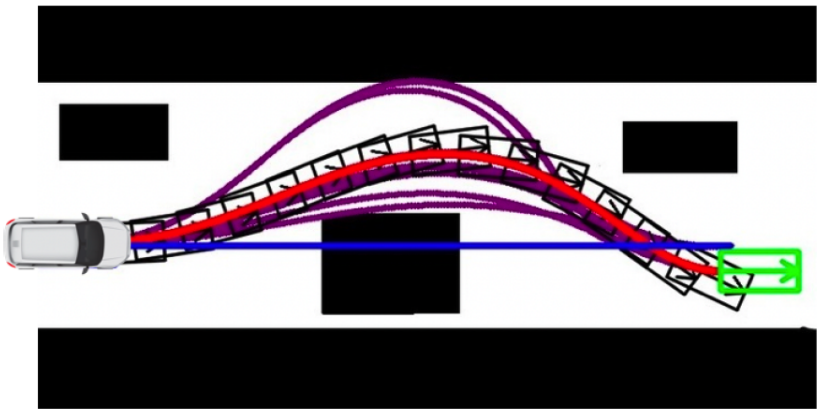
a,
Network

Model-based Policy Optimization in RL

1. Model-based policy optimization in RL has a strong influence from the control theory, that optimizes a controller
2. The controller uses the model, also termed as the system dynamics $s_t = f(s_{t-1}, a_{t-1})$, to decide the optimal controls for a trajectory to minimize the cost:

$$\arg\min_{a_1,\ldots,a_T} \sum_{t=1}^{T} c(s_t, a_t) \text{ subject to } s_t = f(s_{t-1}, a_{t-1})$$

Optimal Control for Trajectory Optimization



$$\min_{a_1,\ldots,a_T} \sum_{t=1}^{T} c(s_t, a_t) \text{ subject to } s_t = f(s_{t-1}, a_{t-1})$$

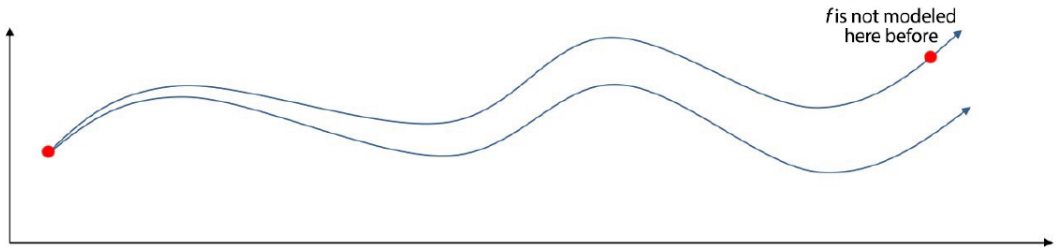
1. If the dynamics is known it becomes the optimal control problem
2. Cost function is the negative reward of the RL problem
3. The optimal solution can be solved by Linear-Quadratic Regulator (LQR) and iterative LQR (iLQR) under some simplified assumptions

Model Learning for Trajectory Optimization: Algorithm 1

1. If the dynamics model is unknown, we can combine model learning and trajectory optimization
2. **Algorithm 1**
   1. run base policy $\pi_0(a_t|s_t)$ (random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
   2. learn dynamics model $s' = f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
   3. plan through $f(s, a)$ to choose actions
3. Step 2 is supervised learning to train a model to minimize the least square error from the sampled data
4. Step 3 can be solved by Linear Quadratic Regulator (LQR), to calculate the optimal trajectory using the model and a cost function

Model Learning for Trajectory Optimization: Algorithm 2

1. The previous solution is vulnerable to drifting, a tiny error accumulates fast along the trajectory
2. We may also land in areas where the model has not been learned yet

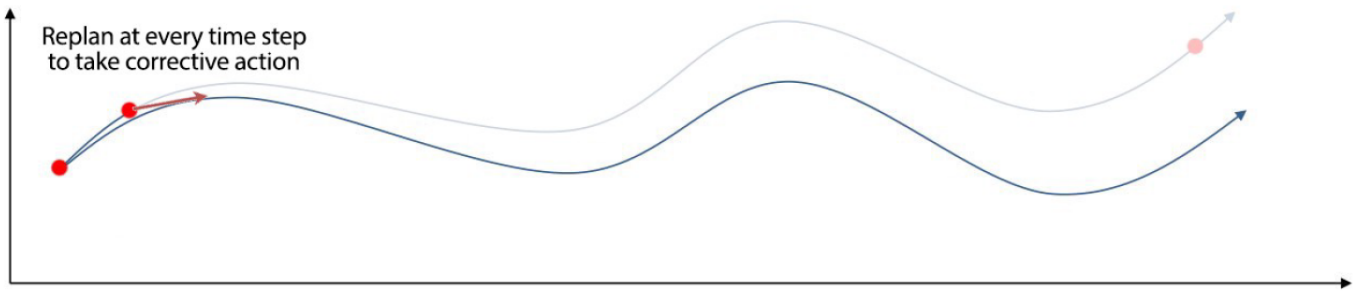


3. So we have the following improved algorithm with learning the model *iteratively*
4. **Algorithm 2**
   1. run base policy $\pi_0(a_t|s_t)$ (random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
   2. Loop
      1. learn dynamics model $s' = f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
      2. plan through $f(s, a)$ to choose actions
      3. execute those actions and add the resulting data $\{(s, a, s')_i\}$ to $\mathcal{D}$

edia,
Information, and Network

Model Learning for Trajectory Optimization: Algorithm 3

1. Nevertheless, the previous method executes all planned actions before fitting the model again. We may be off-grid too far already
2. So we can use Model Predictive Control (MPC) that we optimize the whole trajectory but we take the first action only, then we observe and replan again
3. In MPC, we optimize the whole trajectory but we take the first action only. We observe and replan again. The replan gives us a chance to take corrective action after observed the current state again



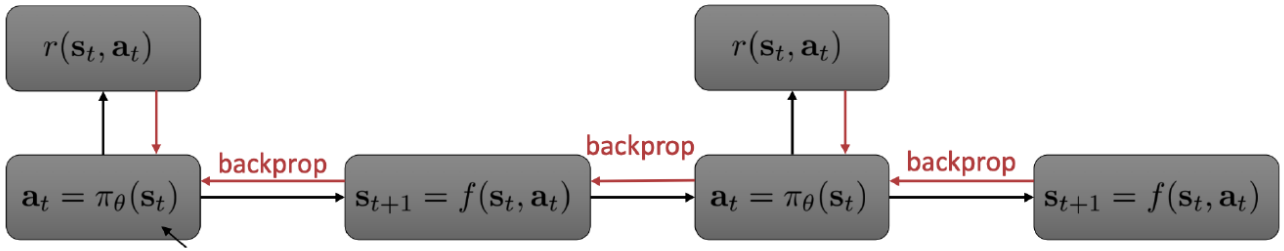
Institute of Media, Information, and Network

Model Learning for Trajectory Optimization: Algorithm 3

1. **Algorithm 3 with MPC**
   1. run base policy $\pi_0(a_t|s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$
   2. Loop every N steps
      1. learn dynamics model $s' = p(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
      2. Loop each step
         1. plan through $f(s, a)$ to choose actions
         2. execute the first planned action and observe the resulting state s' (MPC)
         3. append (s, a, s') to dataset $\mathcal{D}$

Model Learning for Trajectory Optimization: Algorithm 4

1. Finally we can plug the policy learning along with model learning and optimal control



2. **Algorithm 4: Learning Model and Policy Together**
   1. run base policy $\pi_0(a_t|s_t)$ (random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
   2. Loop
      1. learn dynamics model $f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
      2. backpropagate through $f(s, a)$ into the policy to optimize $\pi_\theta(a_t|s_t)$
      3. run $\pi_\theta(a_t|s_t)$, appending the visited (s, a, s') to $\mathcal{D}$

Institute of Media, Information, and Network

Vanilla model-based deep reinforcement learning

- It includes model learning and policy learning.
- The model and the policy are represented by neural networks.
- In model learning, samples are collected from interaction with the environment, supervised learning is used to fit a dynamics model.
- In policy learning, the learned model is used to search an improved policy.
- Dynamics modeling with neural networks does not perform well and limited to relatively simple low-dimensional tasks.
- Model-based methods are more sample efficient and their performance is worse due to model bias.
- Model-free methods achieve better performance at the expense of higher sample complexity.

Model-Ensemble Trust-Region Policy Optimization, ICLR 2018.

Model learning

- Assume that state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function $\mathcal{R}$ are known
- Transition dynamics is modeled with a neural network, to predict the next state given a state and an action as inputs, i.e.,

$$s_{t+1} = \hat{f}_\phi(s_t, a_t)$$

- The objective is to find a parameter ϕ to minimize the loss function, i.e.,

$$\min_\phi \frac{1}{|\mathcal{D}|} \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}} \left\| s_{t+1} - \hat{f}_\phi(s_t, a_t) \right\|_2^2$$

Policy learning

- During training, model-based methods maintain an approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- The policy is then updated w.r.t. the approximate MDP, to maximize the expected sum of rewards, i.e.,

$$\hat{\eta}(\theta; \phi) := \mathbb{E}_{\hat{\tau}}[\sum_{t=0}^{T} r(s_t, a_t)]$$

- The gradient can be estimated by BPTT method, i.e.,

$$\nabla_\theta \hat{\eta} = \mathbb{E}_{s_0 \sim \rho_0(s_0), \zeta_i \sim \mathcal{N}(0, I_m)}[\nabla_\theta \sum_{t=0}^{T} r(s_t, a_t)]$$

Vanilla model-based deep RL algorithm

Algorithm 1 Vanilla Model-Based Deep Reinforcement Learning

Initialize a policy π_θ and a model $\hat{f}_\phi$.
Initialize an empty dataset D.
repeat
    Collect samples from the real environment f using π_θ and add them to D.
    Train the model $\hat{f}_\phi$ using D.
    repeat
        Collect fictitious samples from $\hat{f}_\phi$ using π_θ.
        Update the policy using BPTT on the fictitious samples.
        Estimate the performance $\hat{\eta}(\theta; \phi)$.
    until the performance stop improving.
until the policy performs well in real environment f.

Institute of Media, Information, and Network

Parameterizing the Model

What function is used to parameterize the dynamics?

1. Global model: $s_{t+1} = f(s_t, a_t)$ is represented by a big neural network
   1. Pro: very expressive and can use lots of data to fit
   2. Con: not so great in low data regimes, and cannot express model uncertainty
2. Local model: model the transition as time-varying linear-Gaussian dynamics
   1. Pro: very data-efficient and can express model uncertainty
   2. Con: not great with non-smooth dynamics
   3. Con: very slow when dataset is big
3. Local model as time-varying linear-Gaussian dynamics

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t))$$
$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t$$

   1. All we needed are the local gradients $A_t = \frac{df}{dx_t}$ and $B_t = \frac{df}{du_t}$

Media, Information, and Network

Parameterizing the Model

1. Local model as time-varying linear-Gaussian

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t))$$
$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t$$

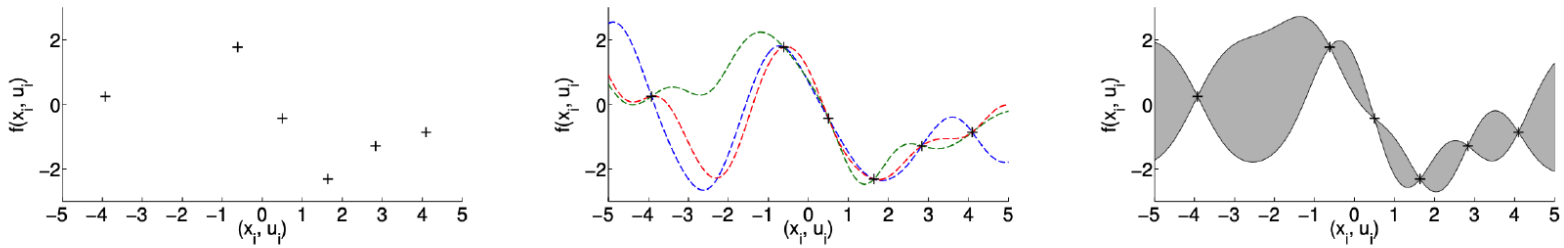
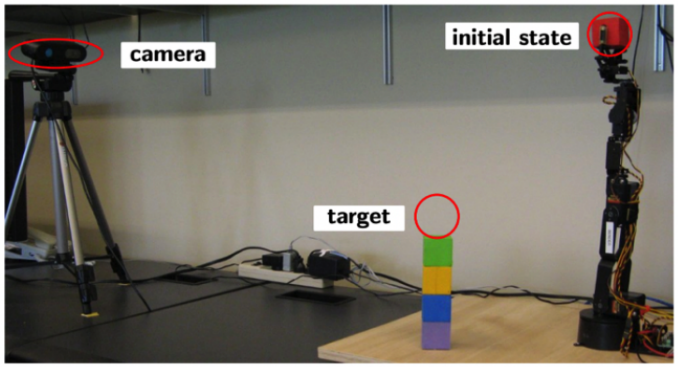


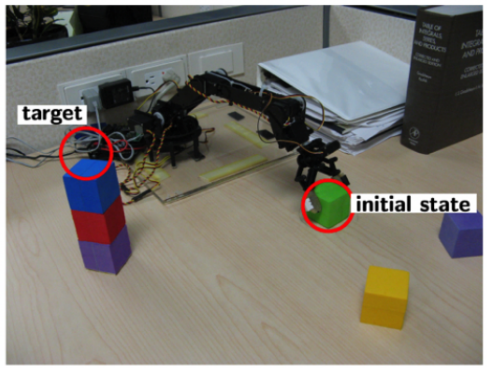
Figure 1. Small data set of observed transitions (left), multiple plausible deterministic function approximators (center), probabilistic function approximator (right). The probabilistic approximator models uncertainty about the latent function.

PILCO: A model-based and data-efficient approach to policy search

Case Study: Model-based Robotic Object Manipulation

1. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. RSS 2011





2. No pose feedback, visual feedback from a Kinetics-type depth camera
3. Total cost: \$500 = 6-degree Arm(\$370)+ Kinetics(\$130)
4. System setup:
   1. Control signal $u \in R^4$: Pulse widths for the first four motors
   2. State $x \in R^3$: 3D center of the object
   3. Policy $\pi : R^3 \rightarrow R^4$
   4. Expected return $J^\pi = \sum_{t=0}^{T} \mathbb{E}_{x_t}[c(x_t)]$ where $c = -\exp(-d^2/\sigma_c^2)$

dia,
Information, and Network

Case Study: Model-based Robotic Object Manipulation

1. Model the system dynamics as probabilistic non-prarametric Gaussian process GP

Algorithm 1 PILCO

init: Set controller parameters ψ to random.
Apply random control signals and record data.
repeat
  Learn probabilistic GP dynamics model using all data
  repeat ▷ Model-based policy search
    Approx. inference for policy evaluation: get $J^\pi(\psi)$
    Gradients $\mathrm{d}J^\pi(\psi)/\mathrm{d}\psi$ for policy improvement
    Update parameters ψ (e.g., CG or L-BFGS).
  until convergence; **return** ψ^*
  Set $\pi^* \leftarrow \pi(\psi^*)$.
  Apply π^* to robot (single trial/episode); record data.
until task learned

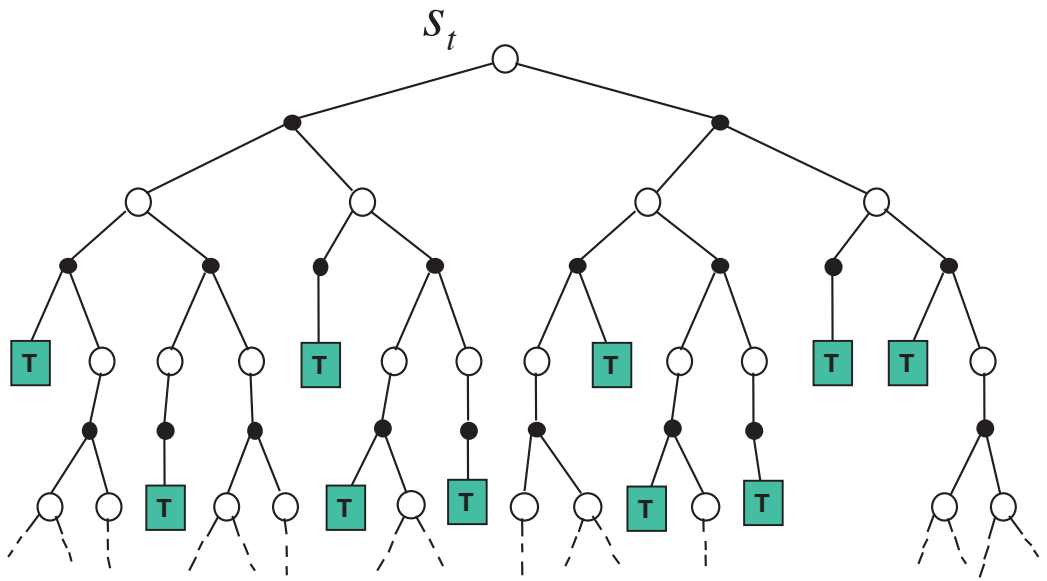
2. PILCO: A model-based and data-efficient approach to policy search. Deisenroth and Rasmussen. ICML 2011
3. Demo link: `http://mlg.eng.cam.ac.uk/pilco/`

Institute of Media, Information, and Network

Table of Contents

Forward Search

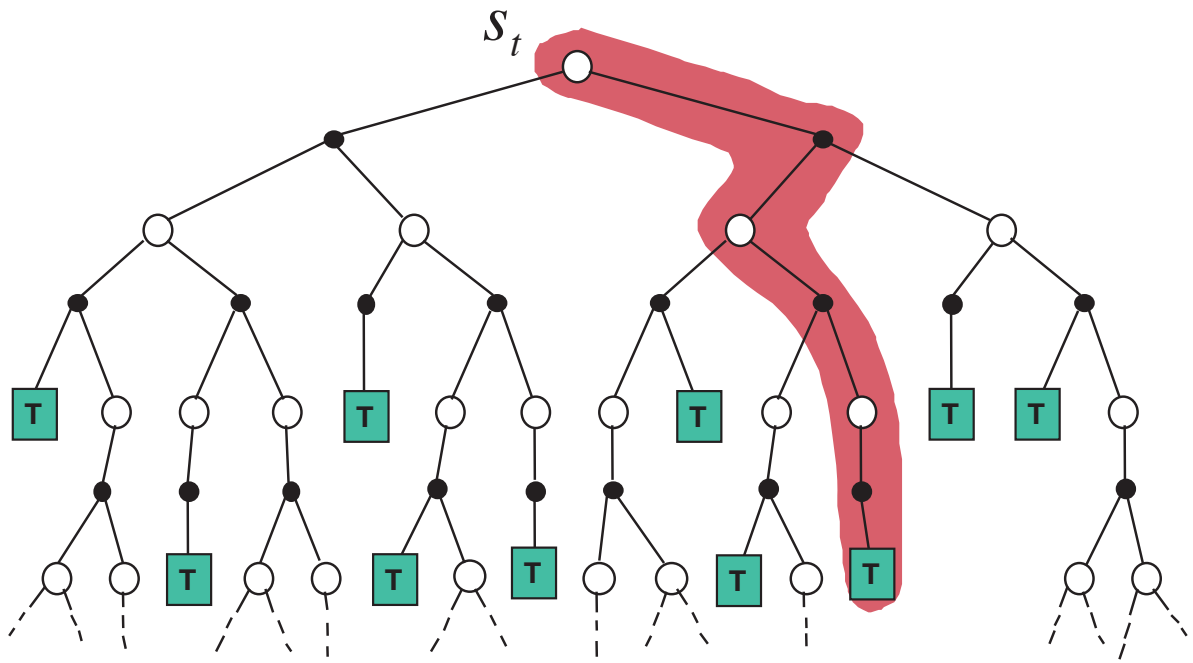
- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead



- 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



Institute of Media, Information, and Network

Simulation-Based Search (2)

- Simulate episodes of experience from now with the model $\{s_t^k, A_t^k, R_{t+1}^k, \cdots, S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu$
- Apply model-free RL to simulated episodes
  - Monte-Carlo control $\rightarrow$ Monte-Carlo search
  - Sarsa $\rightarrow$ TD search

Simple Monte-Carlo Search

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

$$\{s_t, a, R^k_{t+1}, S^k_{t+1}, \cdots, S^k_T\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

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

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

- Select current (real) action with maximum value

$$a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$$

Institute of Media, Information, and Network

Monte-Carlo Tree Search (Evaluation)

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

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

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

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

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

$$a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$$

Institute of Media, Information, and Network

Monte-Carlo Tree Search (Simulation)

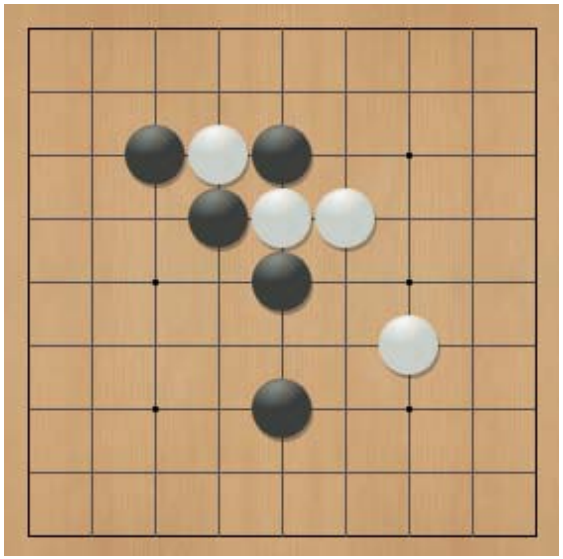
- In MCTS, the simulation policy π improves
- Each simulation consists of two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions to maximize $Q(S, A)$
  - Default policy (fixed): pick actions randomly
- Repeat (each simulation)
  - Evaluate states $Q(S, A)$ by Monte-Carlo evaluation
  - Improve tree policy, e.g. by ϵ-greedy(Q)
- Monte-Carlo control applied to simulated experience
- Converges on the optimal search tree, $Q(S, A) \rightarrow q_*(S, A)$

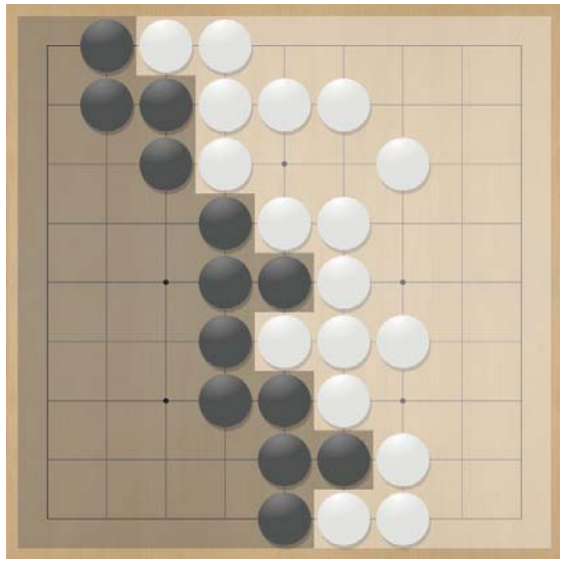
Case Study: the Game of Go

- 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





Media,
Information, and Network

Position Evaluation in Go

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

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

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

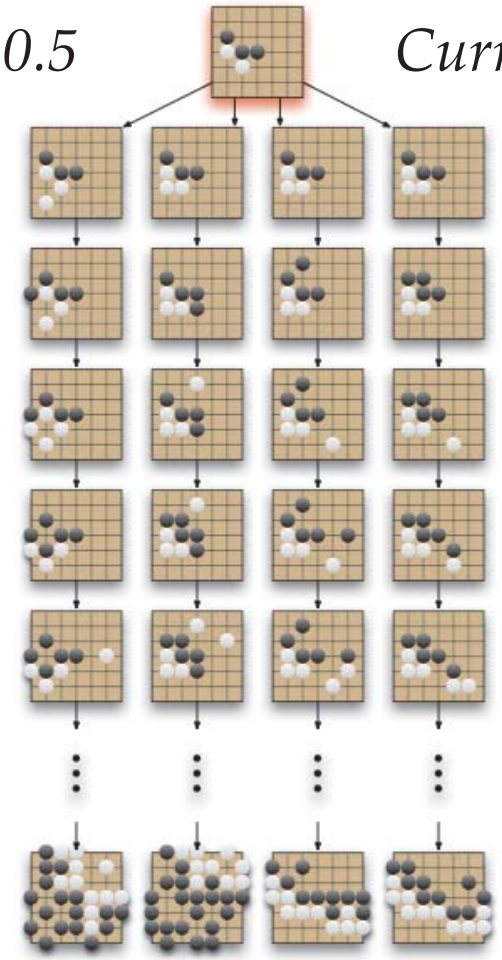
- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

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

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

Monte-Carlo Evaluation in Go

$V(s) = 2/4 = 0.5$



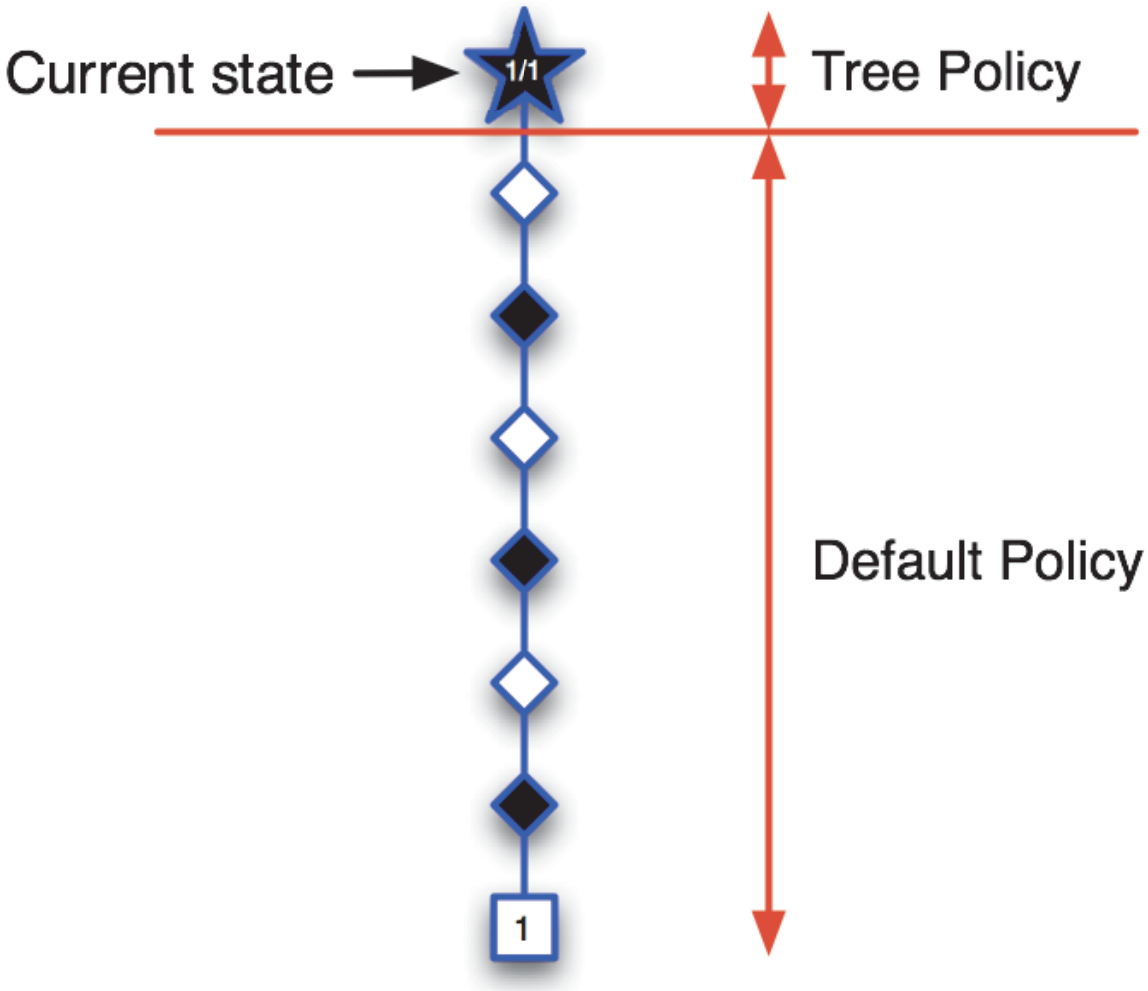
Current position s

Simulation

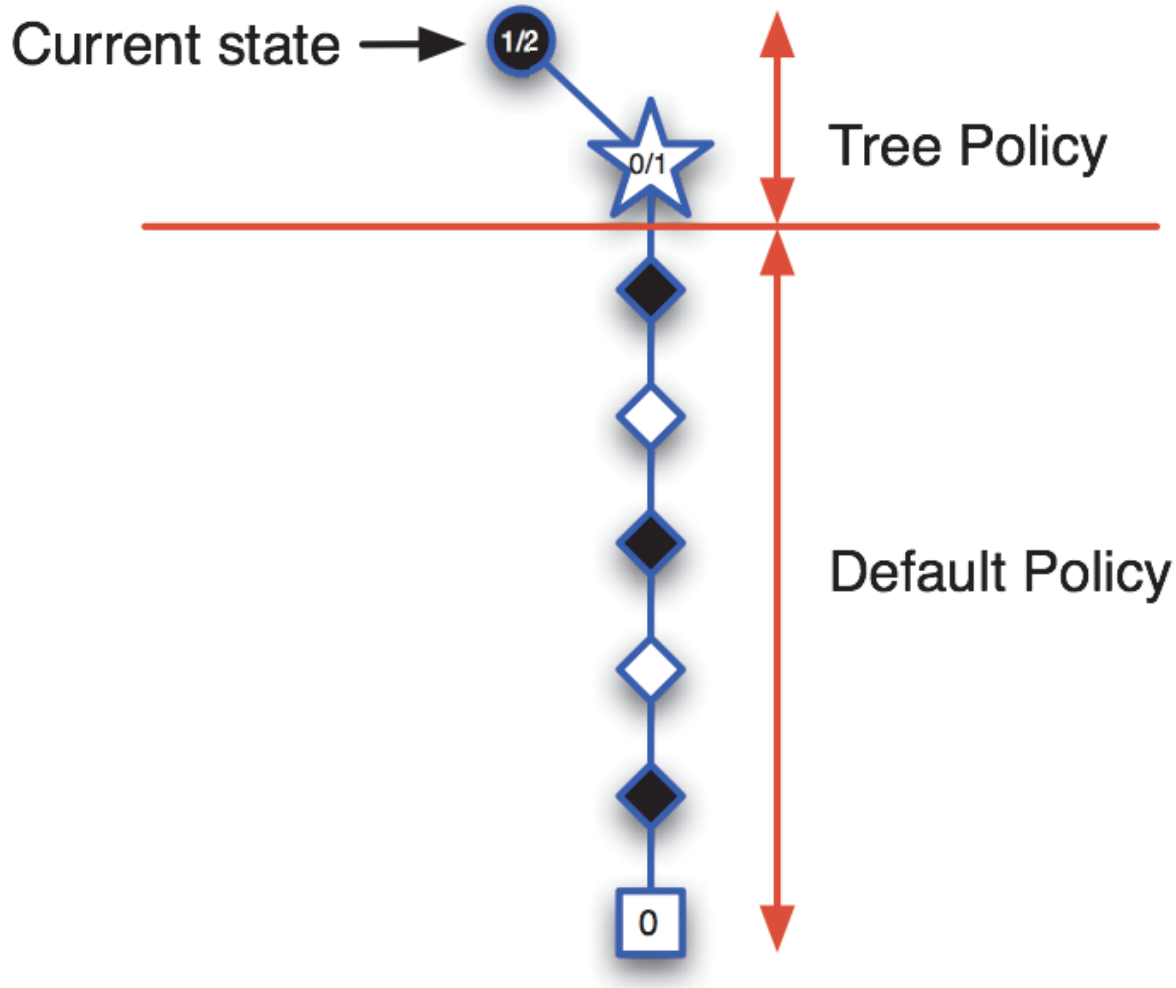
1 1 0 0

Outcomes

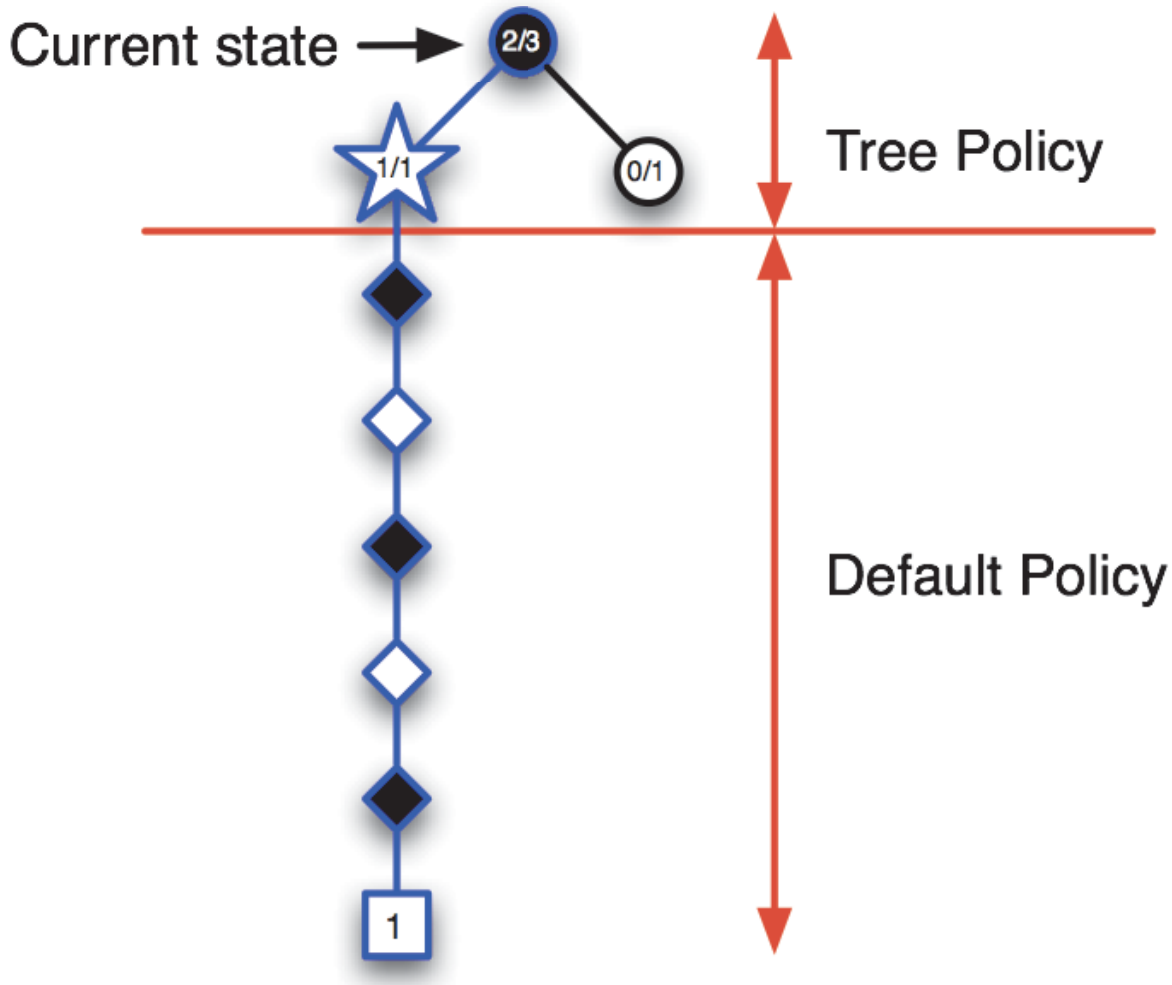
Applying Monte-Carlo Tree Search (1)



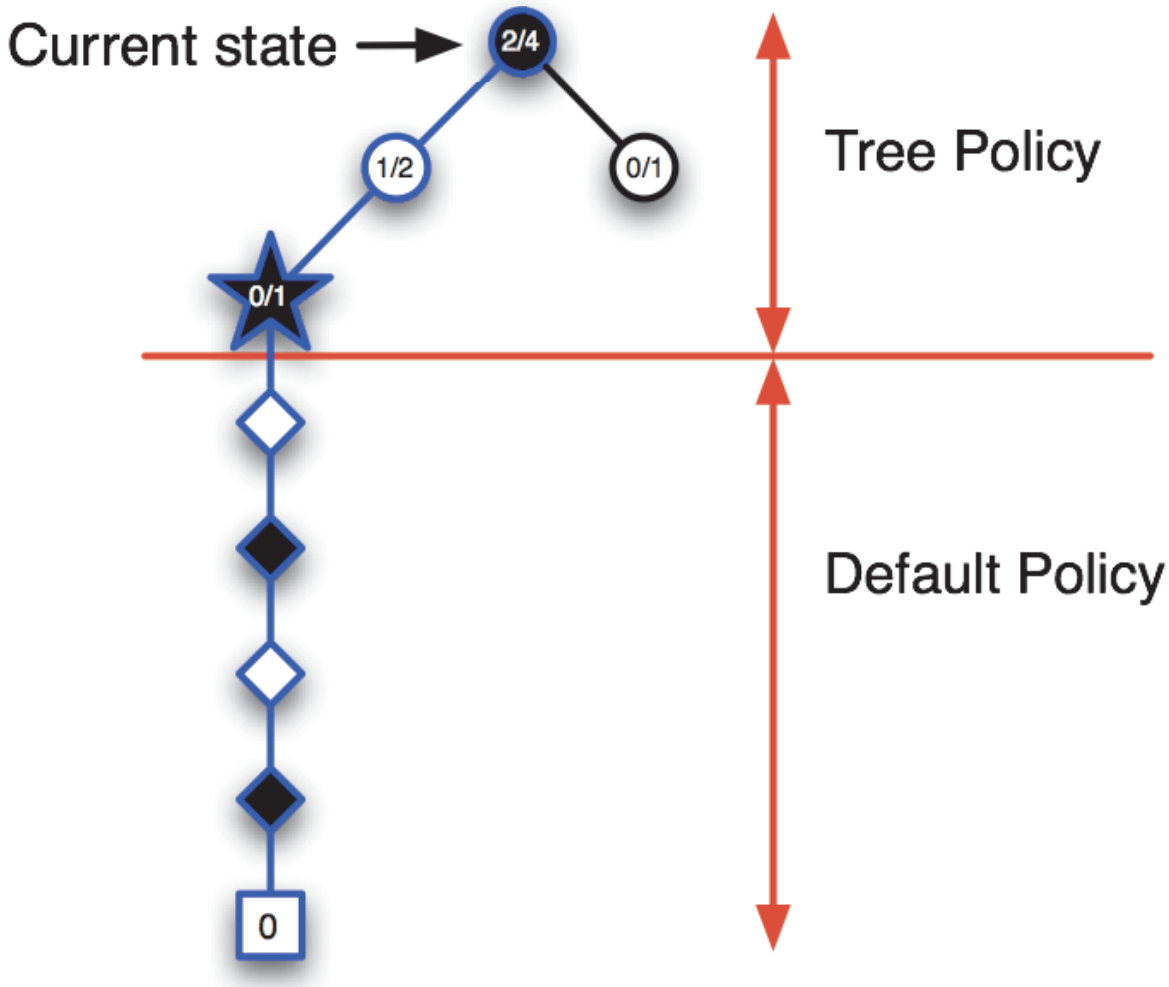
Applying Monte-Carlo Tree Search (2)



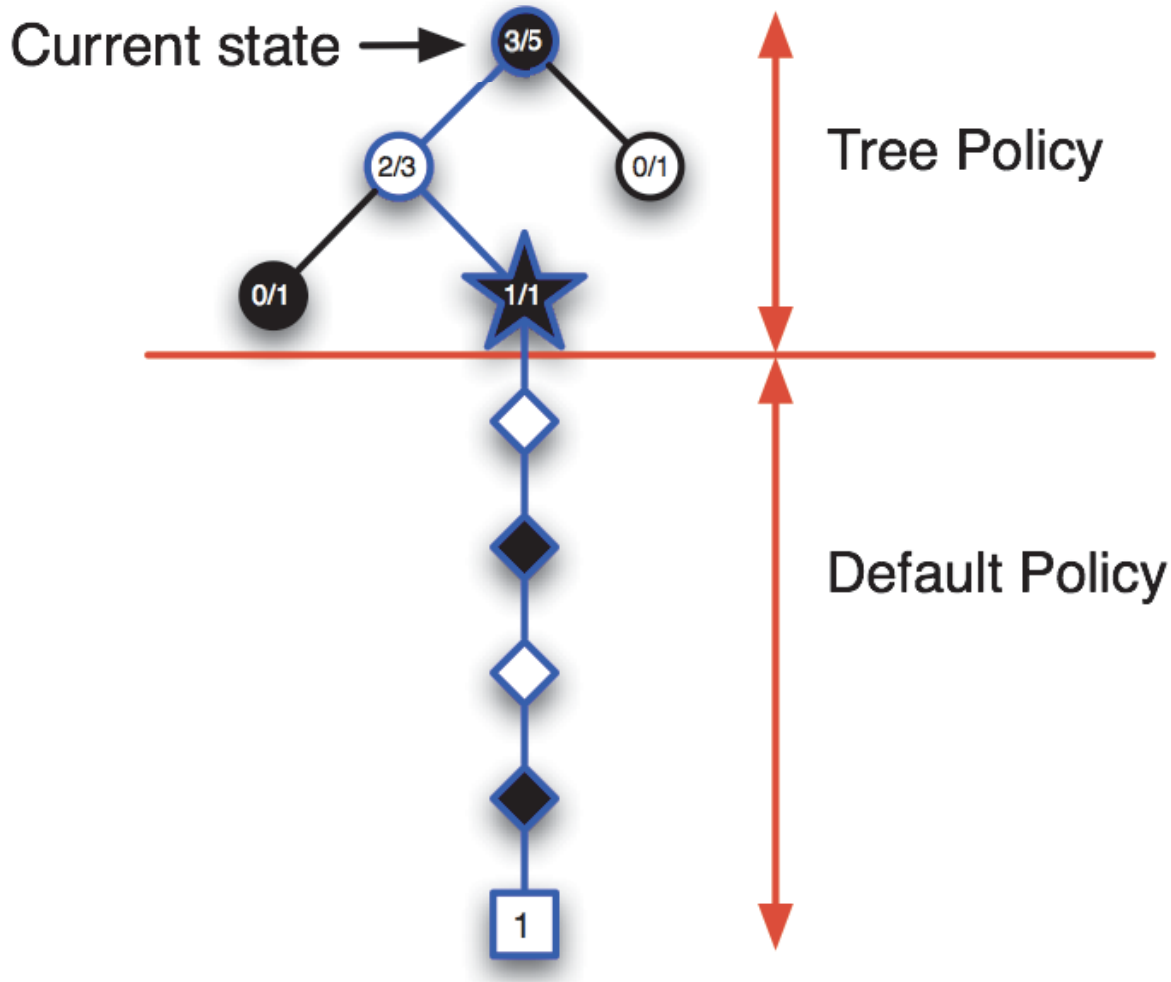
Applying Monte-Carlo Tree Search (3)



Applying Monte-Carlo Tree Search (4)



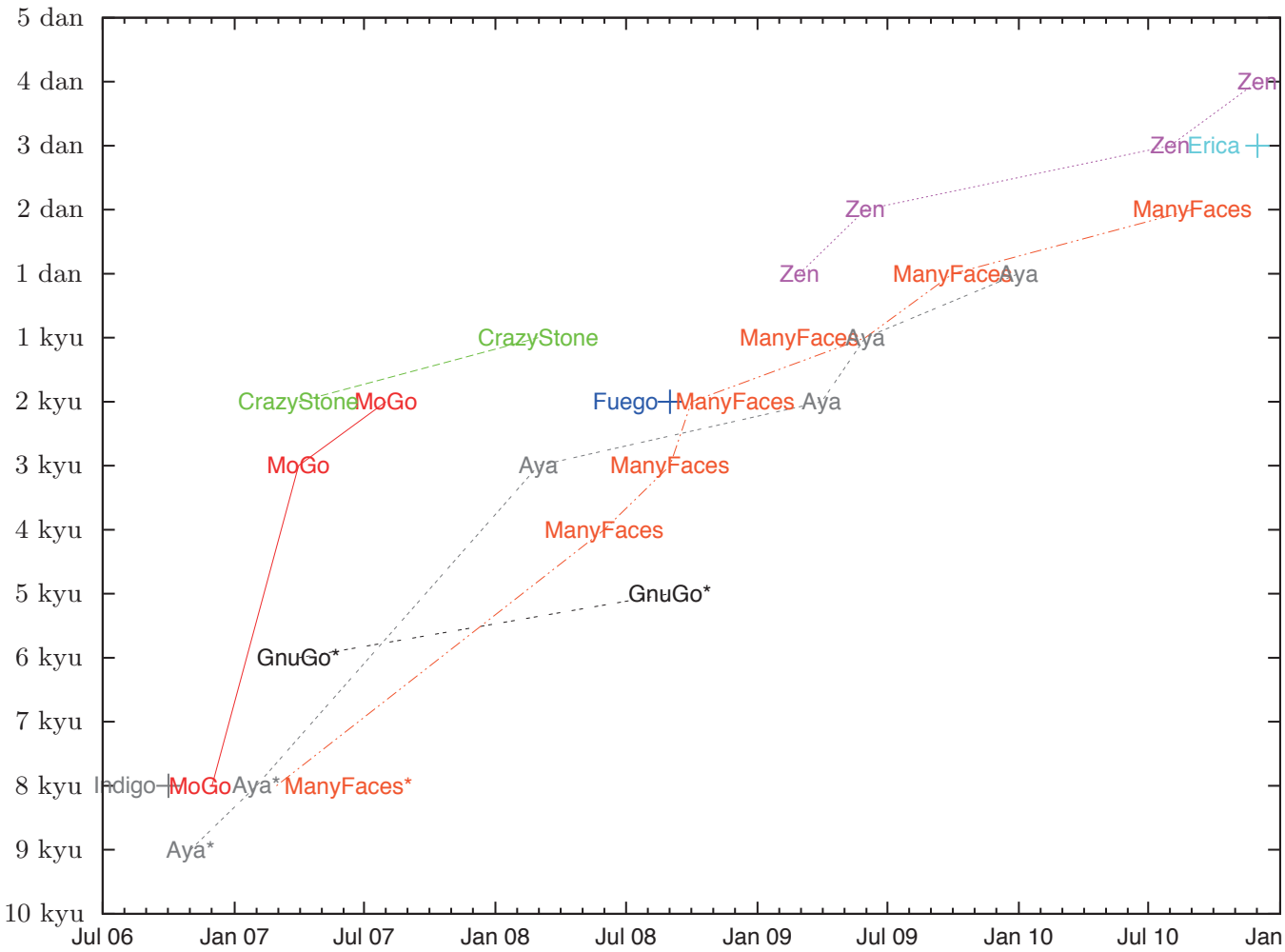
Applying Monte-Carlo Tree Search (5)



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



Institute of Media, Information, and Network

Temporal-Difference Search

- Simulation-based search
- Using TD instead of MC (bootstrapping)
- MC tree search applies MC control to sub-MDP from now
- TD search applies Sarsa to sub-MDP from now

MC vs. TD search

- For model-free reinforcement learning, bootstrapping is helpful
  - TD learning reduces variance but increases bias
  - TD learning is usually more efficient than MC
  - TD(λ) can be much more efficient than MC
- For simulation-based search, bootstrapping is also helpful
  - TD search reduces variance but increases bias
  - TD search is usually more efficient than MC search
  - TD(λ) search can be much more efficient than MC search

TD Search

- Simulate episodes from the current (real) state s_t
- Estimate action-value function $Q(s, a)$
- For each step of simulation, update action-values by Sarsa

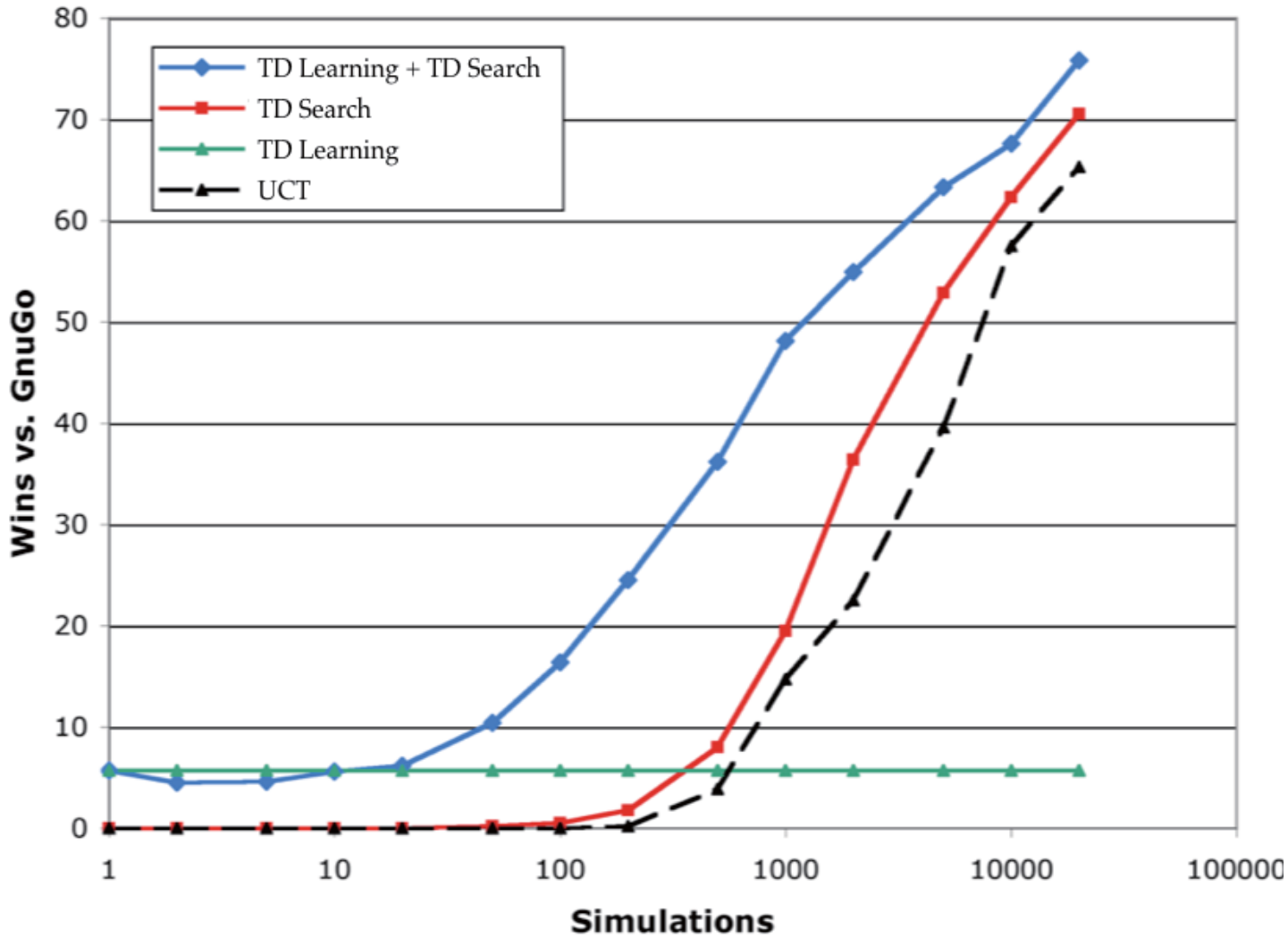
$$\Delta Q(S, A) = \alpha(R + \gamma Q(S', A') - Q(S, A))$$

- Select actions based on action-values
  - e.g. ϵ-greedy
- May also use function approximation for Q

Dyna-2

- In Dyna-2, the agent stores two sets of feature weights
  - Long-term memory
  - Short-term (working) memory
- Long-term memory is updated from real experience using TD learning
  - General domain knowledge that applies to any episode
- Short-term memory is updated from simulated experience using TD search
  - Specific local knowledge about the current situation
- Over value function is sum of long and short-term memories

Results of TD search in Go



tute of Media,
nformation, and Network