CSE 291 – AI Agents
RL and Search Combined

Prithviraj Ammanabrolu

Thanks to David Silver’s DeepMind RL Course and Rich Sutton’s RL Book. Some slides were adapted form there.

Temporal Difference

- With *Monte Carlo*, we update the value function from a complete episode, and so we **use the actual accurate discounted return of this episode.**

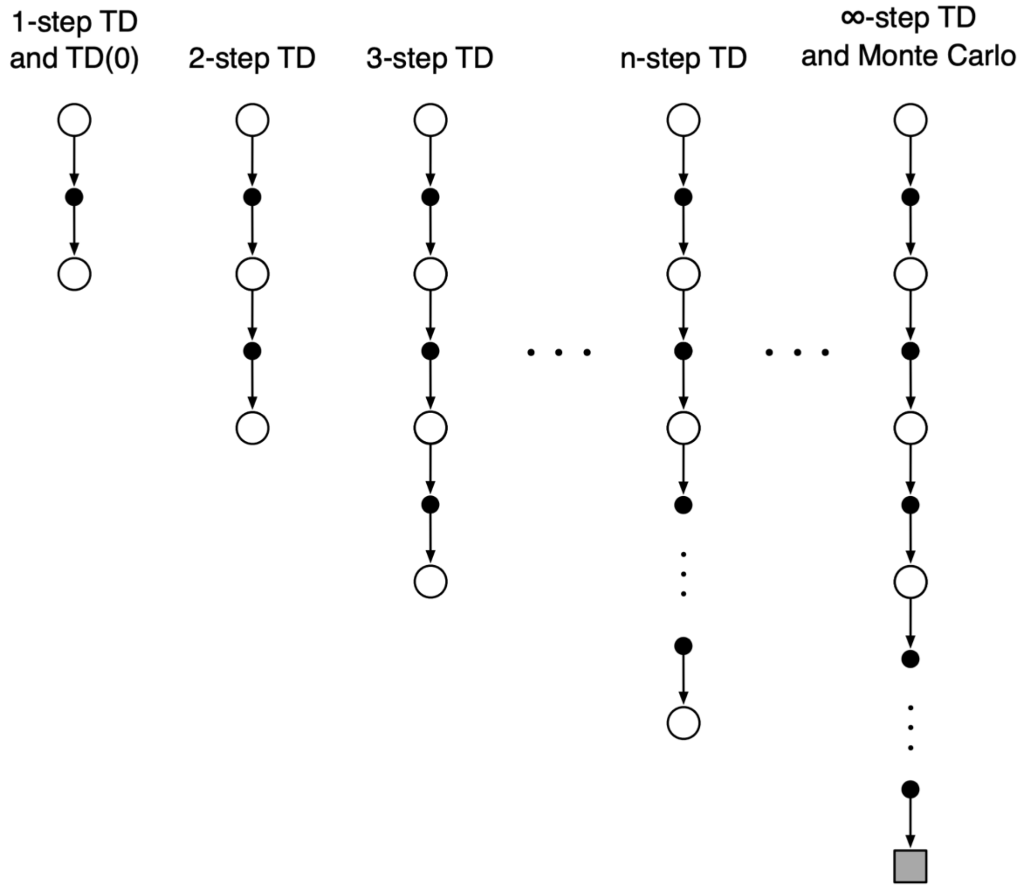
$$\text{Monte Carlo:} \quad V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- With *TD Learning*, we update the value function from a step, and we replace G_t, which we don't know, with **an estimated return called the TD target – a bootstrapping method similar to DP**

$$\text{TD Learning:} \quad V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

TD(0) → TD(∞)

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$



Off-policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While following behavior policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

Why is this important?

- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
- Learn about optimal policy while following exploratory policy
- Learn about multiple policies while following one policy

Q-Learning

- We now allow both behavior and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$
- $\pi(S_{t+1}) = \text{argmax}_{a'} Q(S_{t+1}, a')$
- The behavior policy μ is e.g. -greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$R_{t+1} + \gamma Q(S_{t+1}, A\ 0\ )$$
$$= R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_{a'} Q(S_{t+1}, a'))$$
$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

Value Function Approximation

- So far we have represented value function by a lookup table
- Every state s has an entry V(s)
- Or every state-action pair s, a has an entry Q(s, a)
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

$$\hat{v}(s, w) \approx v_{\pi}(s) \text{ or } \hat{q}(s, a, w) \approx q_{\pi}(s, a)$$

  - Generalize from seen states to unseen states
  - Update parameter w using MC or TD learning

Can you do better if you have a Model?

- Everything so far was Model Free
    - No model
    - Learn value function (and/or policy) from experience
- If you know how the world will change in response to your action before you do it, can you use that somehow to influence your actions?
- This is the problem of “given a world model” how to use it.

Model Free vs Model Based 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**

Sample Based Planning

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

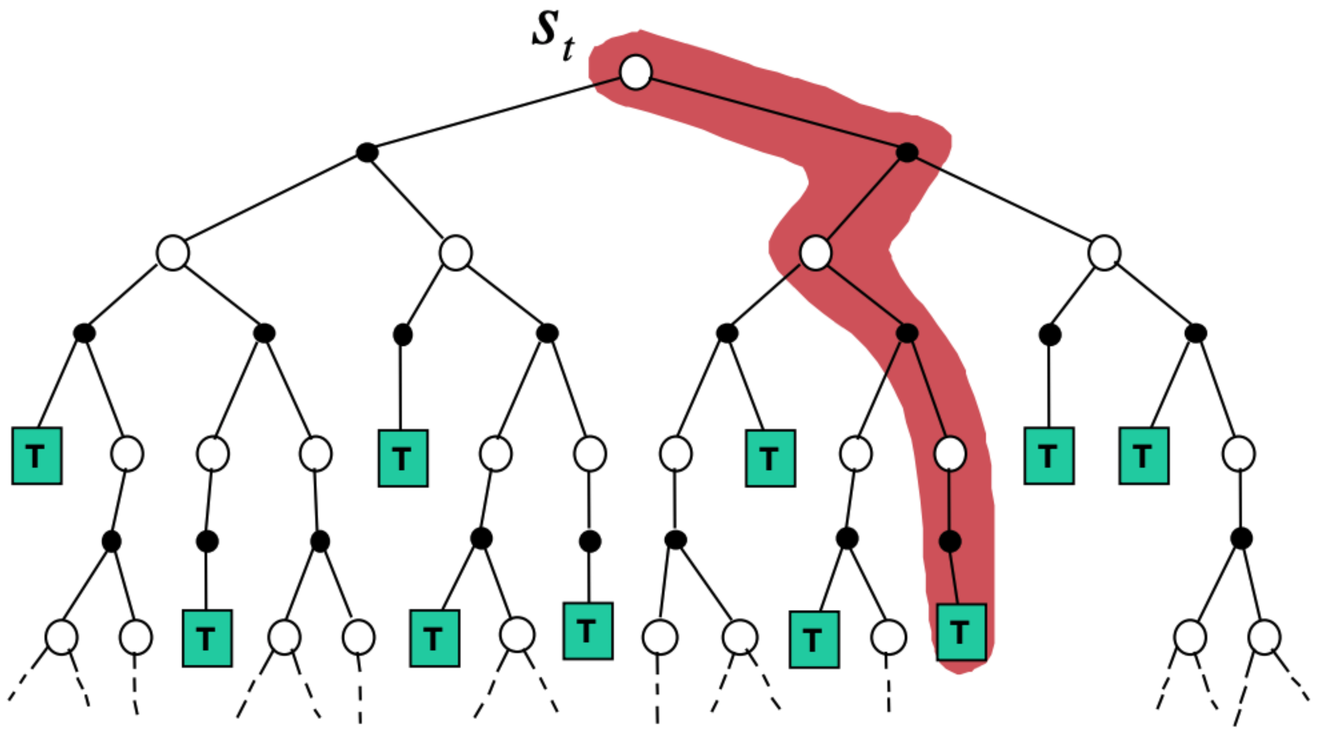
$$S_{t+1} \sim T_\eta(S_{t+1} \mid S_t , A_t)$$

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

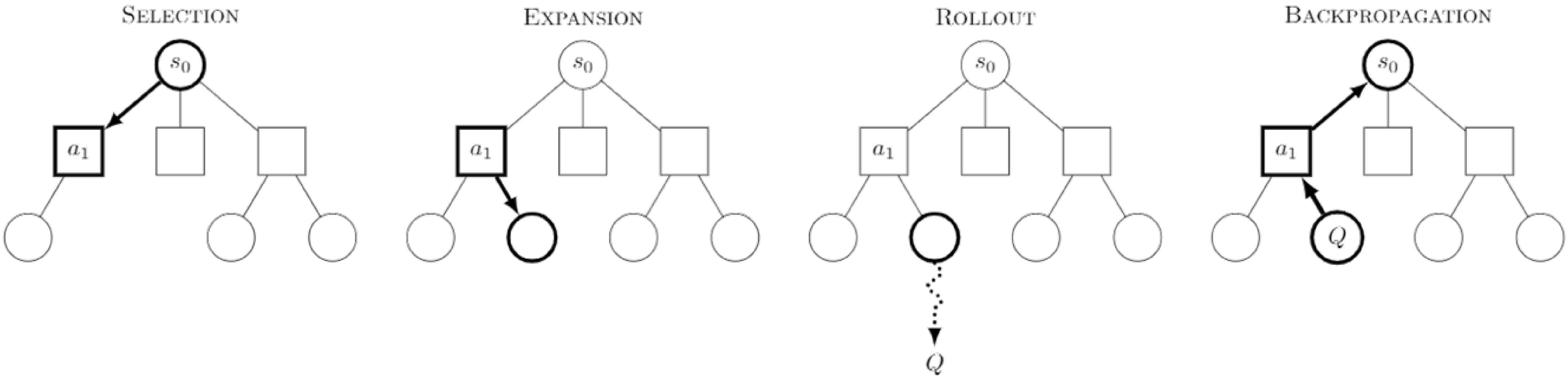
- Apply model-free RL to samples, e.g.: Monte-Carlo control Sarsa Q-learning
- Sample-based planning methods are often more efficient

Simulation Search

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



Revisit MCTS



MCTS (contd)

- Given a model M_v and a simulation policy π
- For each action $a \in A$
  - Simulate K episodes from current (real) state

    $s_t \{s_t , a, R^k_{t+1}, S^k_{t+1}, A^k_{t+1}, \ldots, S^k_T \}^K_{k=1} \sim M_{v, \pi}$
  - Evaluate actions by mean return (Monte-Carlo evaluation)

    $Q(s_t , a) = 1/K \sum_{k=1}^{K} G_t \rightarrow q_\pi(s_t , a)$
- Select current (real) action with maximum value

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

MCTS Evaluation

- Given a model M_v
- Simulate K episodes from current state st using current simulation policy π $\{s^t, A^k_t, R^k_{t+1}, S^k_{t+1}, A^k_{t+1}, \ldots, S^k_T\}^K_{k=1} \sim M_{v,\pi}$
- Build a search tree containing visited states and actions
- Evaluate states Q(s, a) by mean return of episodes from s, a

$$Q(s, a) = 1 / N(s, a) \sum_{k=1}^{K} \sum_{u=t}^{T} \mathbf{1}(Su, Au = s, a) Gu \rightarrow q_\pi(s, a)$$

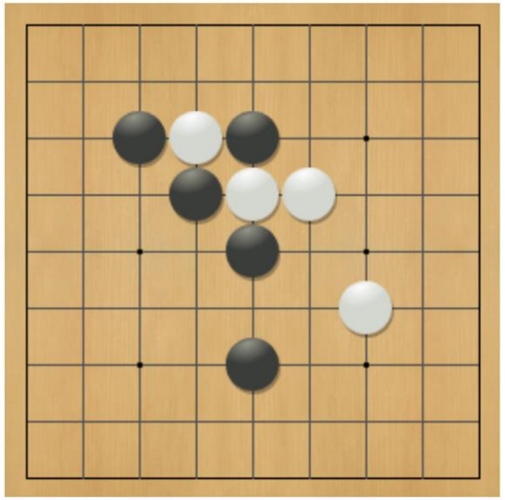
- After search is finished, select current (real) action with maximum value in search tree $a_t = \text{argmax}_{a \in A} Q(s_t, a)$

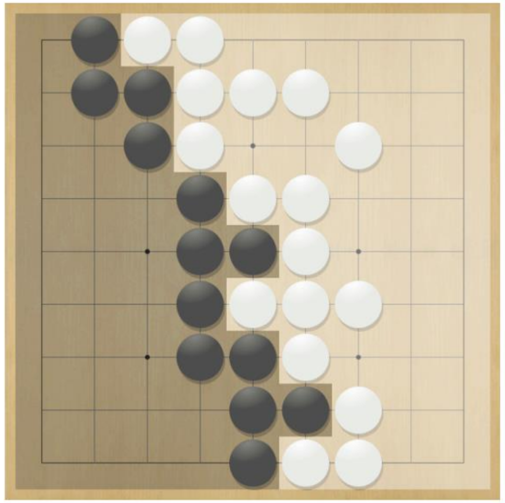
MCTS Simulation

- 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 − greedy(Q)
- Monte-Carlo control applied to simulated experience
- Converges on the optimal search tree, Q(S, A) → q*(S, A)

Go Case Study

- 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



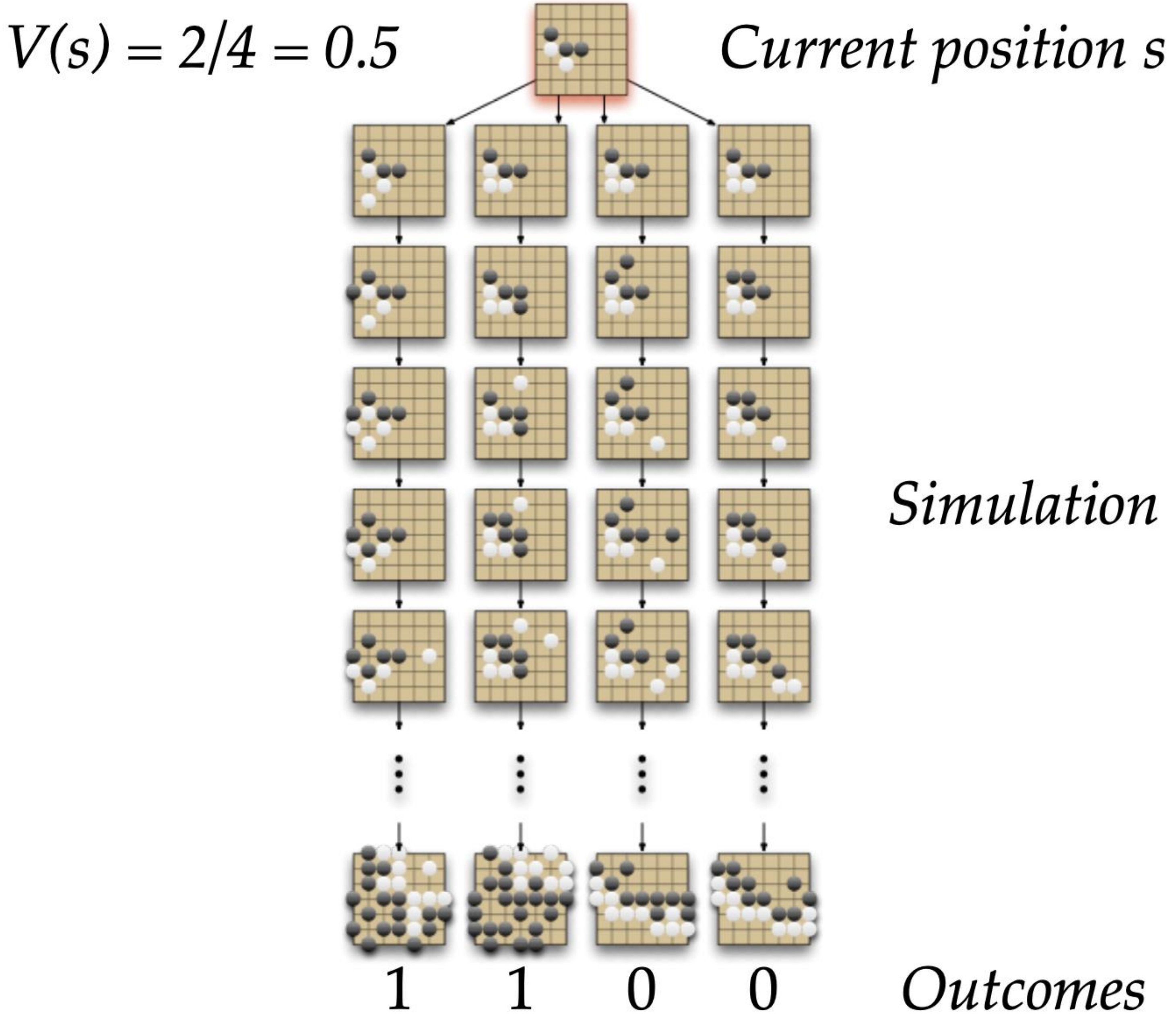


Go Case Study

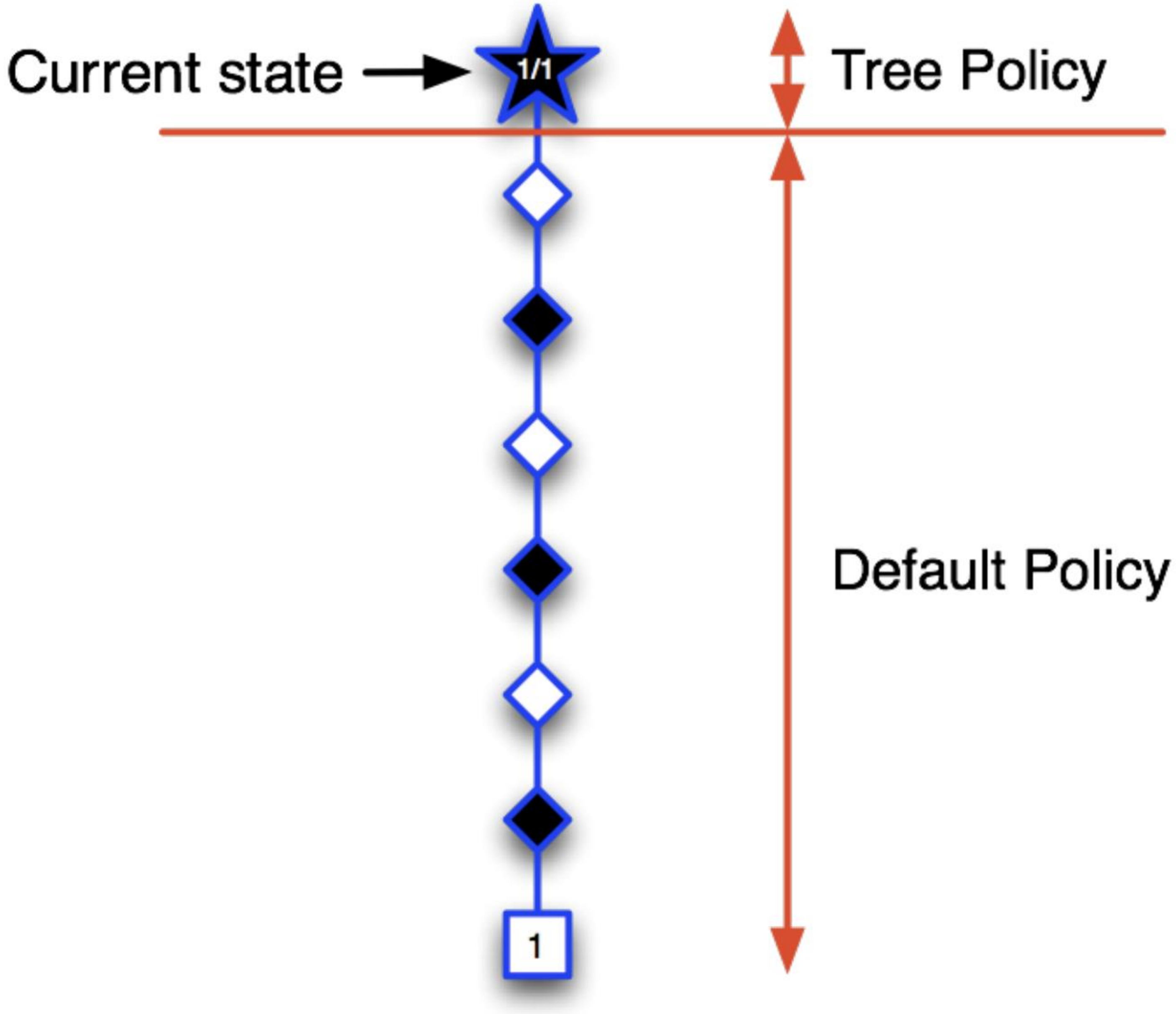
- How good is a position s?
- Reward function (undiscounted):
  - $R_t = 0$ for all non-terminal steps $t < T$
  - $R_T = 1$ if Black wins
  - $R_T = 0$ if White wins
- Policy $\pi = \langle \pi_B , \pi_W \rangle$ selects moves for both players, Self Play
- Value function (how good is position s):

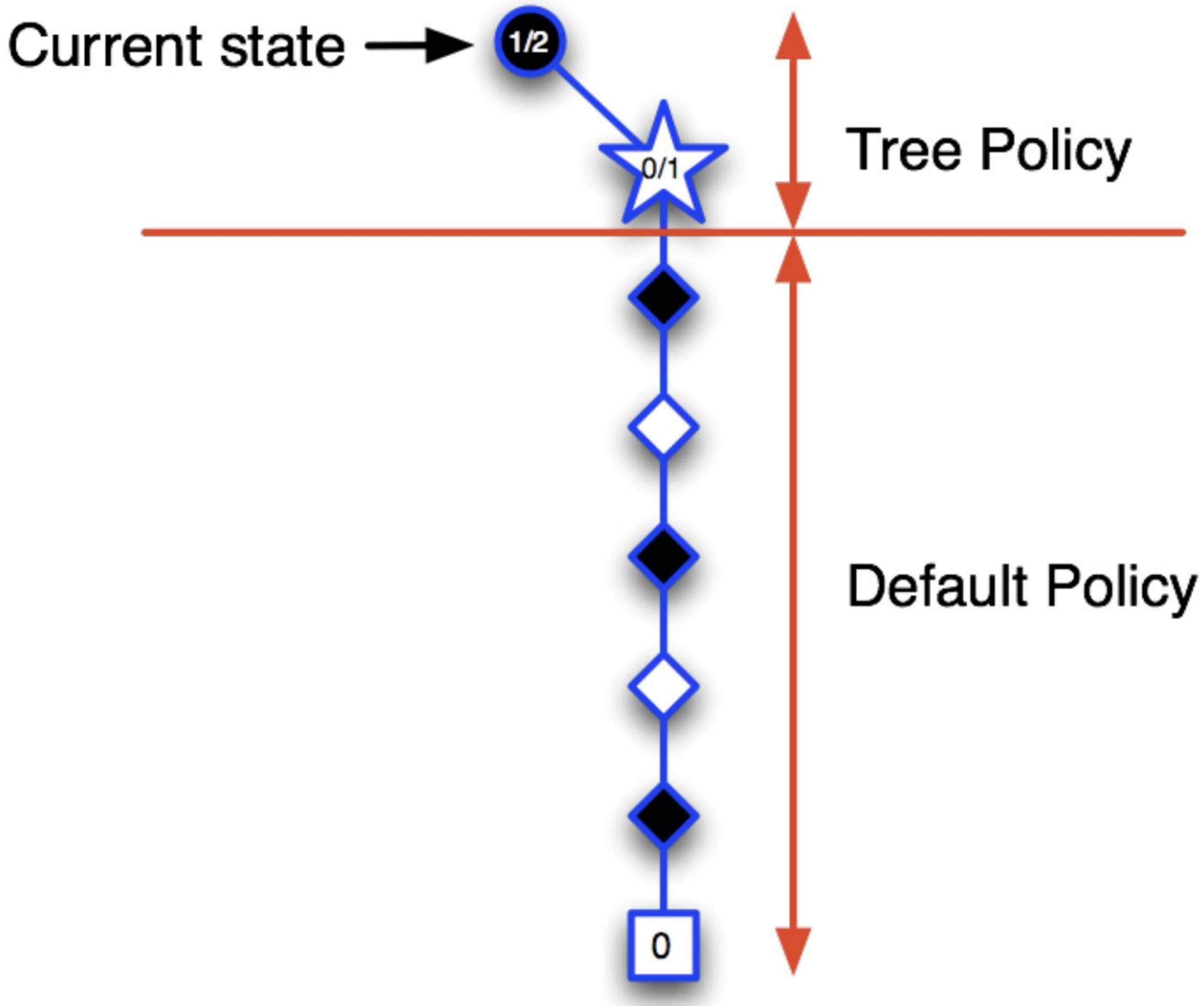
$$v_\pi(s) = E_\pi [R_T \mid S = s] = P [\text{Black wins} \mid S = s]$$

$$v^*(s) = \max_{\pi B} \min_{\pi W} v_\pi(s)$$

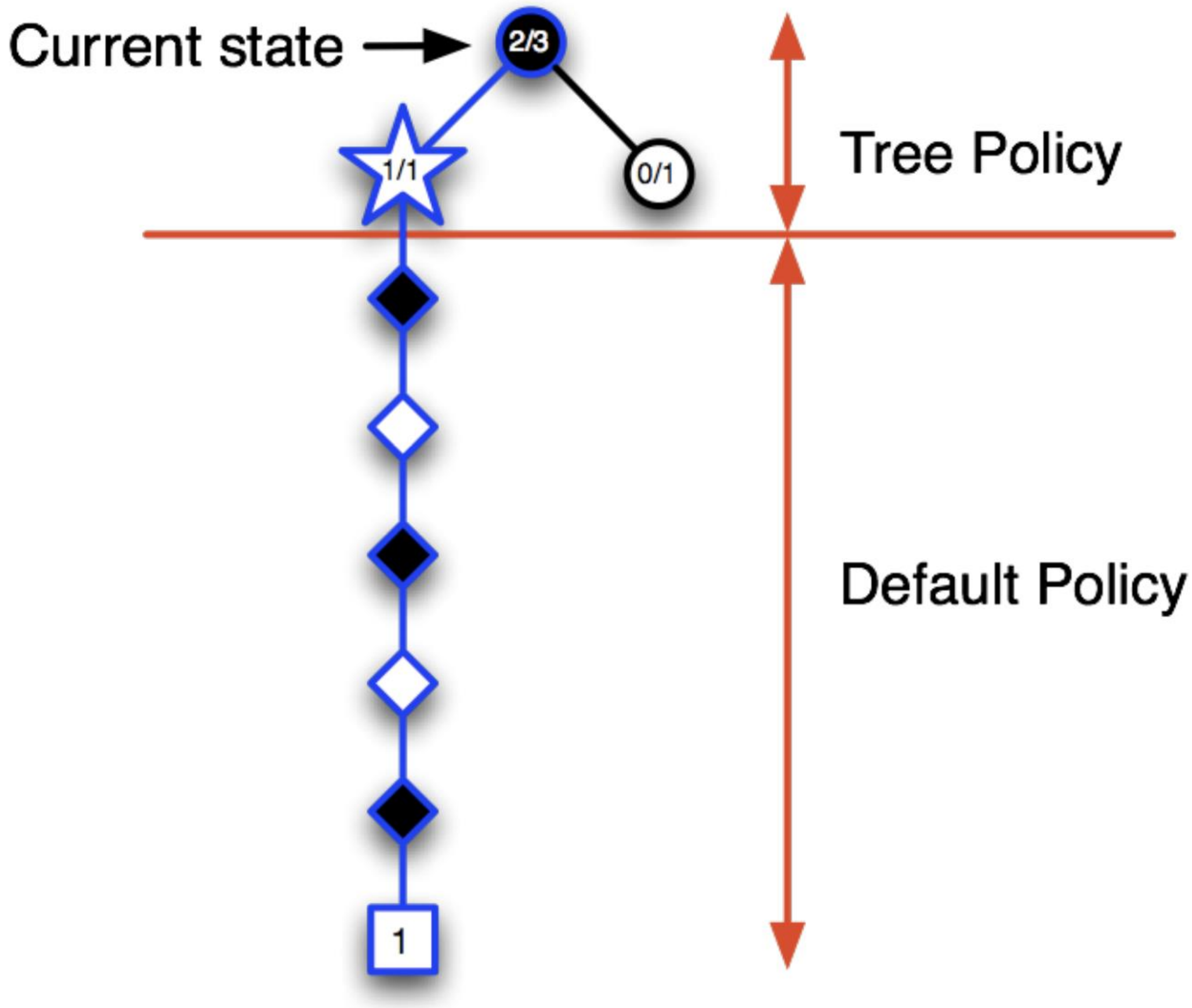


$V(s) = 2/4 = 0.5$
Current position s
Simulation
1 1 0 0
Outcomes

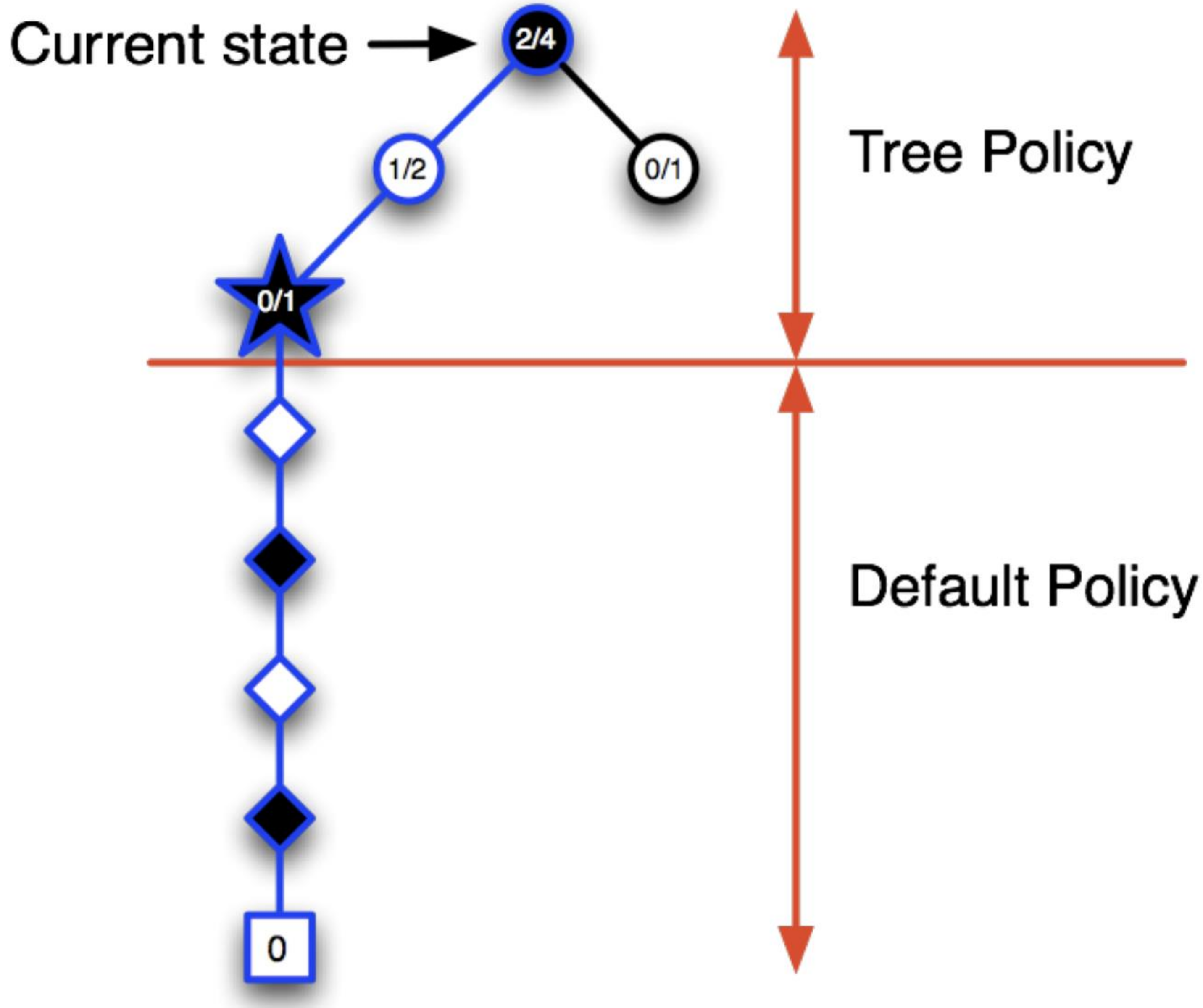

Current state
1/1
Tree Policy
Default Policy
1



Current state
1/2
0/1
Tree Policy
Default Policy
0



Current state
2/3
1/1
0/1
Tree Policy
Default Policy
1



Current state
2/4
1/2
0/1
0/1
0
Tree Policy
Default Policy

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

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

- Select actions based on action-values Q(s, a) e.g. -greedy
- May also use function approximation for Q

AlphaGo

- Same exact MC method as what we just talked about
- Just use neural nets to learn the probabilities using self play and outcome rewards
- Needed a lot of human games to train the initial value networks
- Also had some hand crafted features to bake in knowledge about the game

AlphaZero

- Relaxed the constraint of requiring a lot of human data and constraints up front by just scaling
- Just do pure online RL

Model Free vs Model Based 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

What is a Model?

- A model M is a representation of an MDP <S, A,T, R>, parametrized by η
- We will assume state space S and action space A are known
- So a model $M = \langle T_\eta, R_\eta \rangle$ represents state transitions $T_\eta \approx T$ and rewards $R\eta \approx R$

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

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

- Typically assume conditional independence between state transitions and rewards

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

Learning a Model

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

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

- Learning s, a → r is a regression problem
- Learning s, a → s' is a density estimation problem
- Pick loss function, e.g. mean-squared error, KL divergence, … Find parameters η that minimizes empirical loss

Model Based RL

- Pick your fav simulation search algo from before and do planning with your model
- Key difference here is that the Model has errors, uncertainty
- What does this mean for how many steps you need to take in an env?

Model Based RL

- Pick your fav simulation search algo from before and do planning with your model
- Key difference here is that the Model has errors, uncertainty
- It will take a lot longer! (Why?)
- This is the overall concept behind MuZero, simultaneously learn both model and policy

Models and Simulation and Reality

- Traditionally we consider two sources of experience
- Real experience: Sampled from environment (true MDP)

  $S' \sim T^a_{s,s'}$

  $R = R^a_s$

- Simulated experience: Sampled from model (approximate MDP)

  $S' \sim T_\eta(S' \mid S, A)$

  $R = R_\eta(R \mid S, A)$

- What's the issue with World Models learned inside a simulation?

Pros and Cons of MBRL

- Pros
  - Can do all the (self, un) supervised learning tricks to learn from large scale data
  - Can reason about uncertainty
- Cons
  - Need model of T first
  - Will build estimate of value from that
  - Two(+) sources of error