Model-based Reinforcement Learning

DAVIDE BACCIU – BACCIU@DI.UNIPI.IT

UNIVERSITÀ DI PISA

Outline

- ✓ Introduction
- ✓ Model-based reinforcement learning
- ✓ Integrated Architectures
  - ✓ Learning and Planning
  - ✓ Real and simulated experience
- ✓ Monte-Carlo Tree Search
- ✓ Go use case

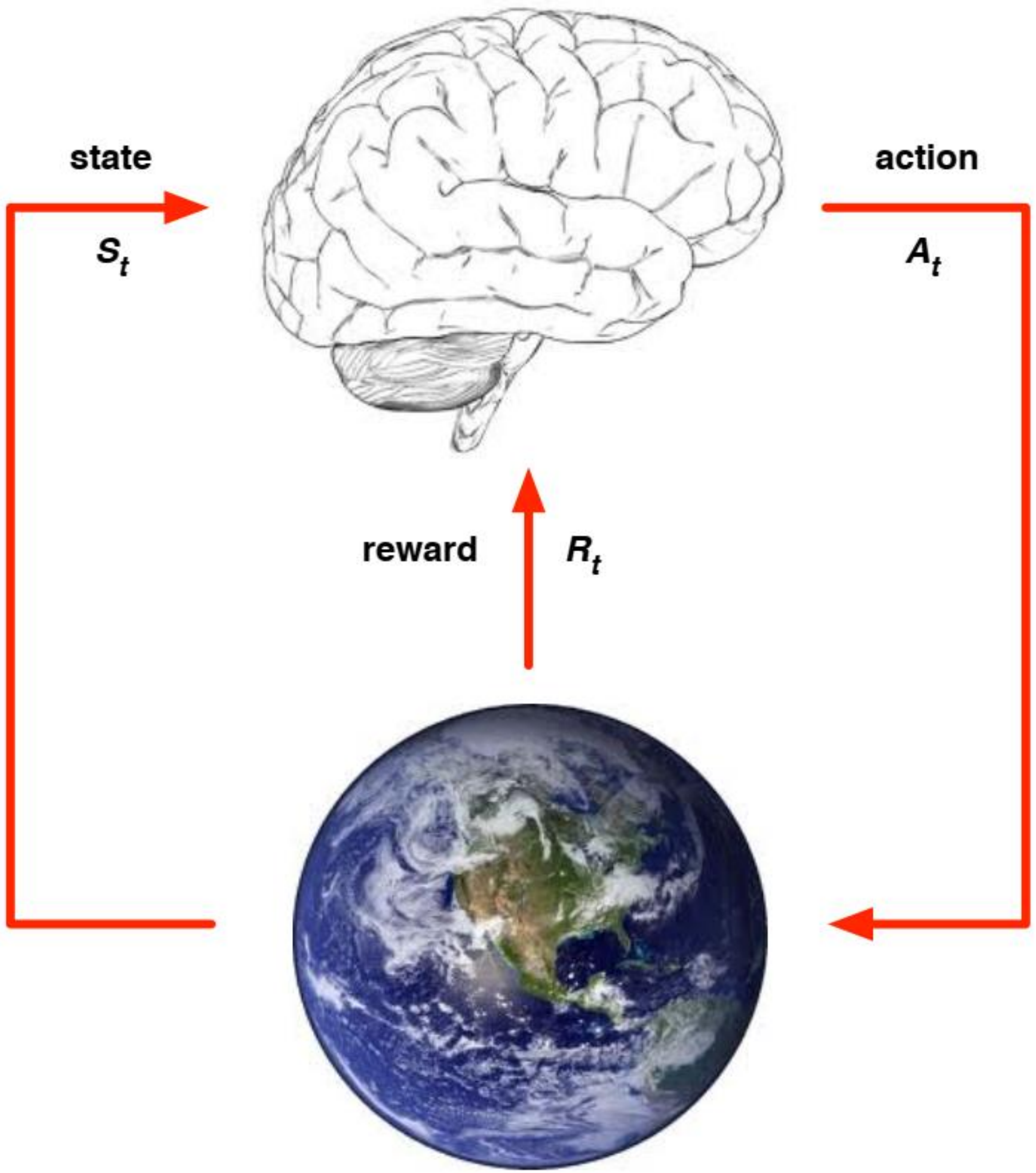
Introduction

Model-Based Reinforcement Learning

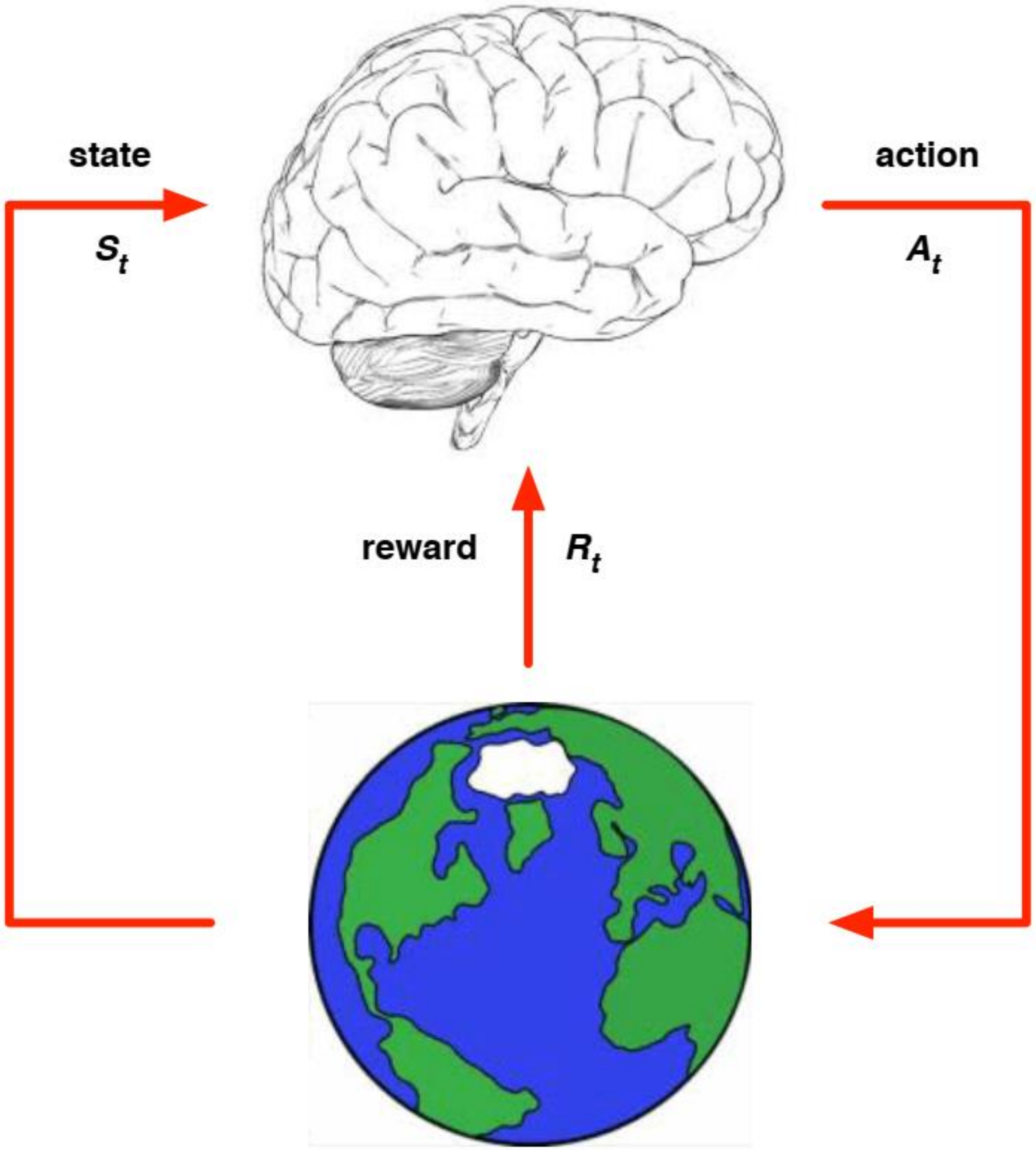
- Previous lectures
  - Learn value function directly from experience
  - Learn policy directly from experience
  - Model-free RL
- Today
  - Learn model directly from experience
  - Use planning to construct a value function or policy
  - Model-based RL
- Integrate learning and planning into a single architecture

Model-Based & 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



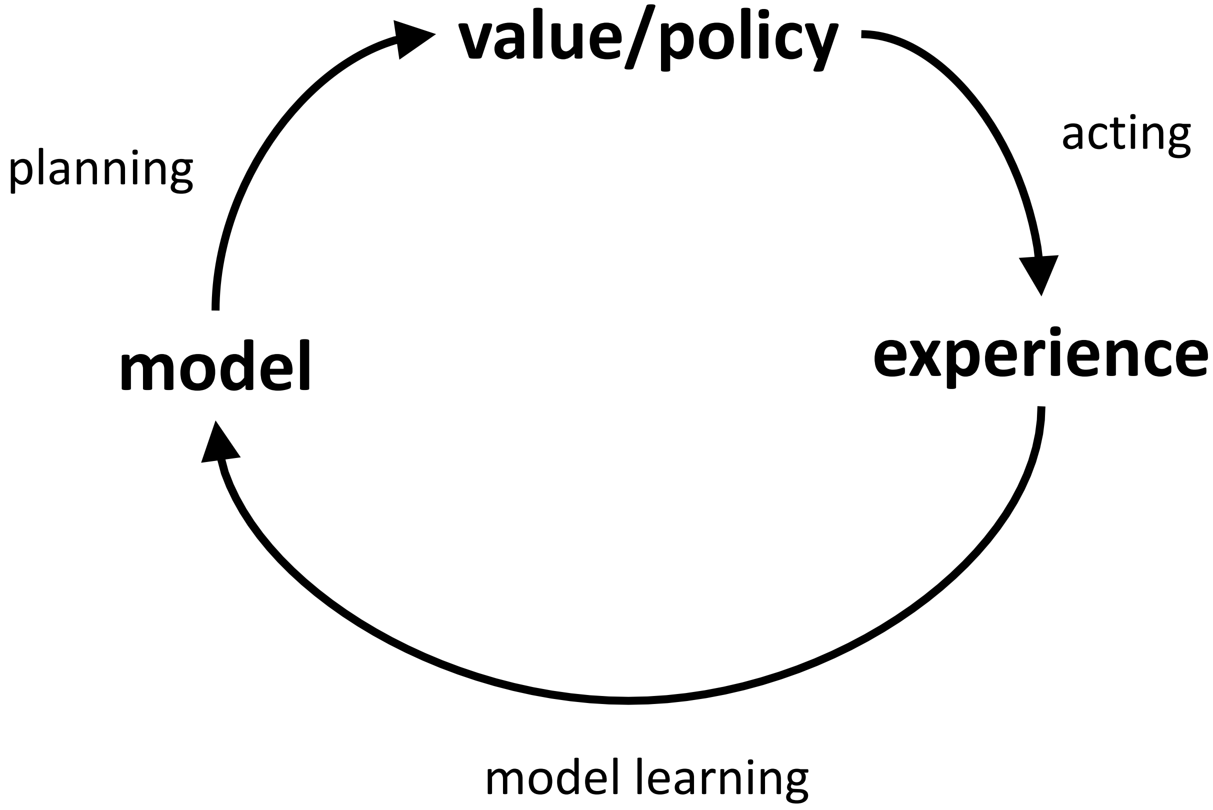
Model-Free RL



Model-based RL

Model-based RL

Model-based RL - Lifecycle



Model-Based RL – Pros and Cons

- ✓ Advantages
  - ✓ Can efficiently learn model by supervised learning methods
  - ✓ Can reason about model uncertainty
- ✓ Disadvantages
  - ✓ First learn a model, then construct a value function
  - ✓ I.e. two sources of approximation error

What is a Model?

- A model $\mathcal{M}_\eta$ is a representation of an MDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{P}, \mathcal{R} \rangle$ parametrized by η
- Assume state space $\mathcal{S}$ and action space $\boldsymbol{\mathcal{A}}$ are known
- So a model $\mathcal{M}_\eta = \langle P_\eta, \mathcal{R}_\eta \rangle$ represents state transitions $P_\eta \approx \boldsymbol{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$

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

- Typically assume conditional independence between state transitions and rewards

$$P(S_{t+1}, R_{t+1} | S_t, A_t) = P(S_{t+1} | S_t, A_t) P(R_{t+1} | S_t, A_t)$$

Model Learning

- ✓Estimate model $\mathcal{M}_\eta$ from experience $\{S_1; A_1; R_2; \dots, S_T\}$
- ✓Supervised learning task

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

- ✓Learning $s, a \rightarrow r$ is a regression problem
- ✓Learning $s, a \rightarrow s'$ is a density estimation problem
- ✓Pick an adequate loss function, e.g. mean-squared error, KL divergence, ...
- ✓Find parameters η that minimise empirical loss

Some Learning Models

- ✓ Table Lookup Model
- ✓ Linear Expectation Model
- ✓ Linear Gaussian Model
- ✓ Gaussian Process Model
- ✓ Deep Belief Network Model
- ✓ …

Model Learning with Table Lookup

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

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

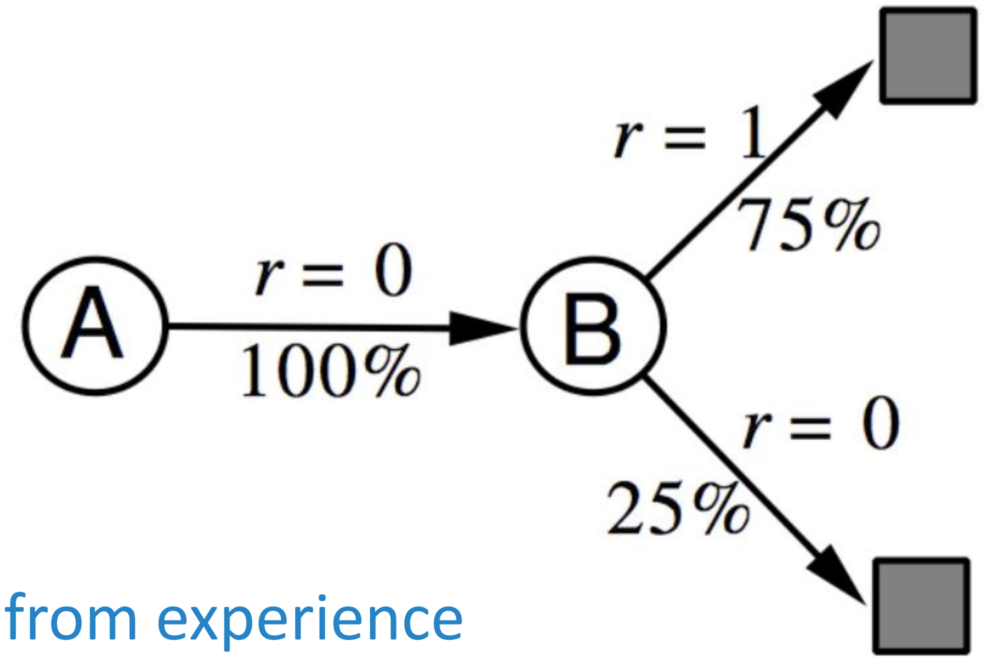
$$\hat{R}^{a}_{s} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t; s, a) R_t$$

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

The Return of AB Example

✓Two states A; B; no discounting; 8 episodes of experience

1. A, 0, B, 0
2. B, 1
3. B, 1
4. B, 1
5. B, 1
6. B, 1
7. B, 1
8. B, 0



✓We have constructed a table lookup from experience

Planning with a Model

- ✓ Given a model $\mathcal{M}_\eta = \langle P_\eta, \mathcal{R}_\eta \rangle$
- ✓ Solve the MDP $\langle \mathcal{S}, \mathcal{A}, P_\eta, \mathcal{R}_\eta \rangle$
- ✓ Using your favourite planning algorithm
  - ✓ Value iteration
  - ✓ Policy iteration
  - ✓ Tree search
  - ✓ …

Sample-Based Planning

- ✓ Simple but powerful approach to planning
- ✓ Use the model only to generate samples
- ✓ Sample experience from model

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

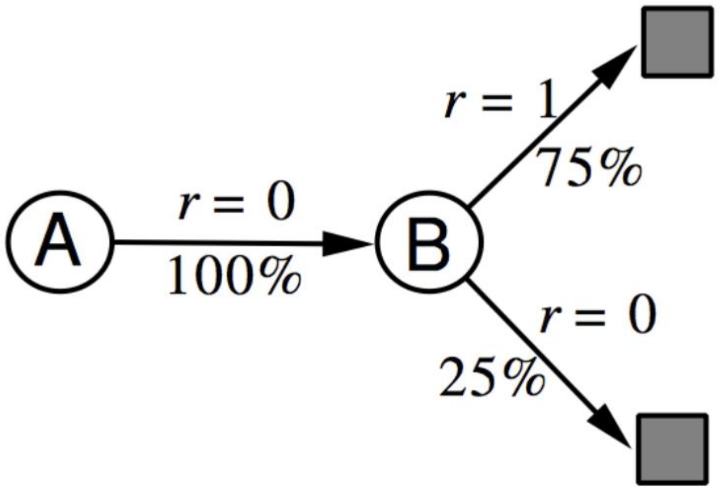
- ✓ Apply model-free RL to samples, e.g.:
  - ✓ MC control
  - ✓ Sarsa
  - ✓ Q-learning
- ✓ Sample-based planning methods are often more efficient

The AB Example - Again

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

Real experience

1. A, 0, B, 0
2. B, 1
3. B, 1
4. B, 1
5. B, 1
6. B, 1
7. B, 1
8. B, 0



e.g. Monte-Carlo learning: V (A) = 1; V (B) = 0:75

Sampled experience

1. B, 1
2. B, 0
3. B, 1
4. A, 0, B, 1
5. B, 1
6. A, 0, B, 1
7. B, 1
8. B, 0

Planning with an Inaccurate Model

- ✓Given an imperfect model $\langle P_\eta, \mathcal{R}_\eta \rangle \neq \langle P, \mathcal{R} \rangle$
  - ✓Performance of model-based RL is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, P_\eta, \mathcal{R}_\eta \rangle$
  - ✓i.e. Model-based RL is only as good as the estimated model
- ✓When the model is inaccurate, planning process will compute a suboptimal policy
  - ✓Solution 1 - When model is wrong, use model-free RL
  - ✓Solution 2 - Reason explicitly about model uncertainty

Integrated Architectures

Real and Simulated Experience

- ✓ Consider two sources of experience
- ✓ Real experience - Sampled from environment (true MDP)

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

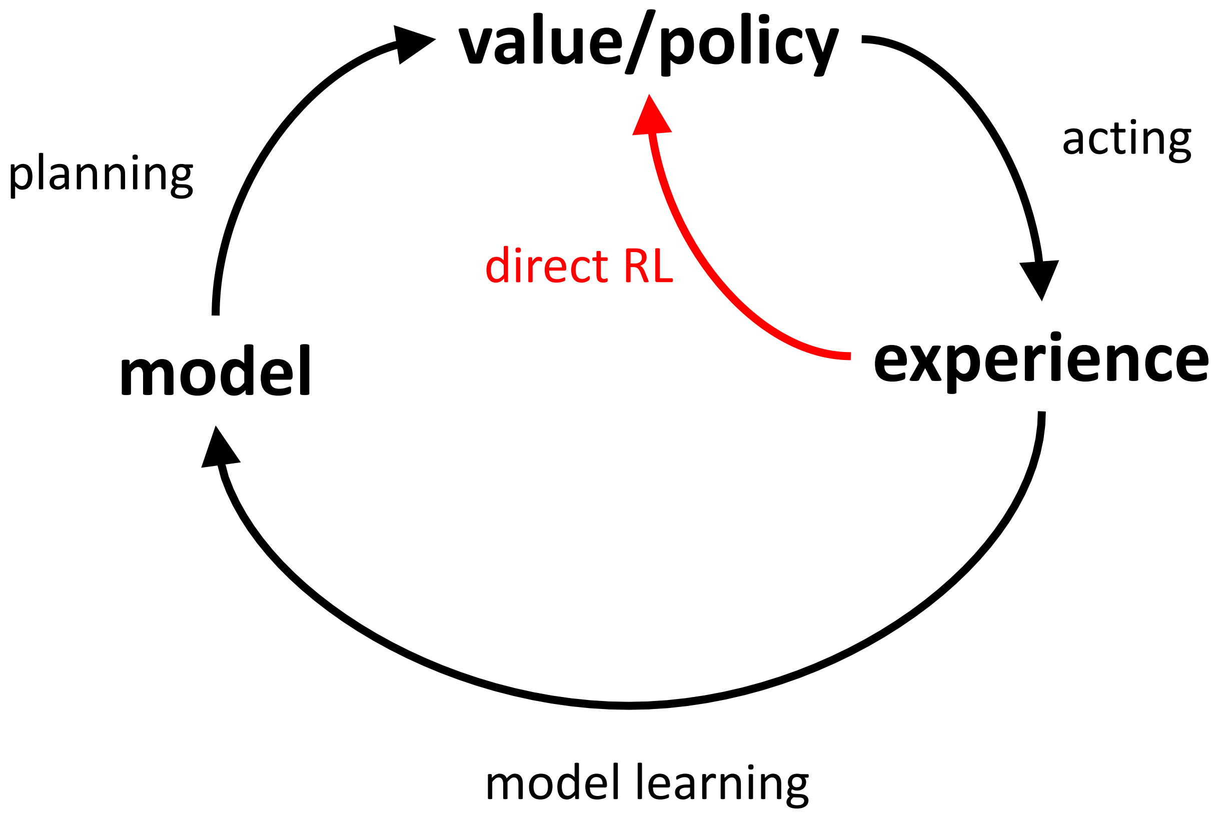
- ✓ Simulated experience - Sampled from model (approximate MDP)

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

Integrating Learning and Planning

- ✓Model-Free RL
  - ✓No model
  - ✓Learn value function (and/or policy) from real experience
- ✓Model-Based RL (using Sample-Based Planning)
  - ✓Learn a model from real experience
  - ✓Plan value function (and/or policy) from simulated experience
- ✓Dyna
  - ✓Learn a model from real experience
  - ✓Learn and plan value function (and/or policy) from real and simulated experience

Dyna Architecture

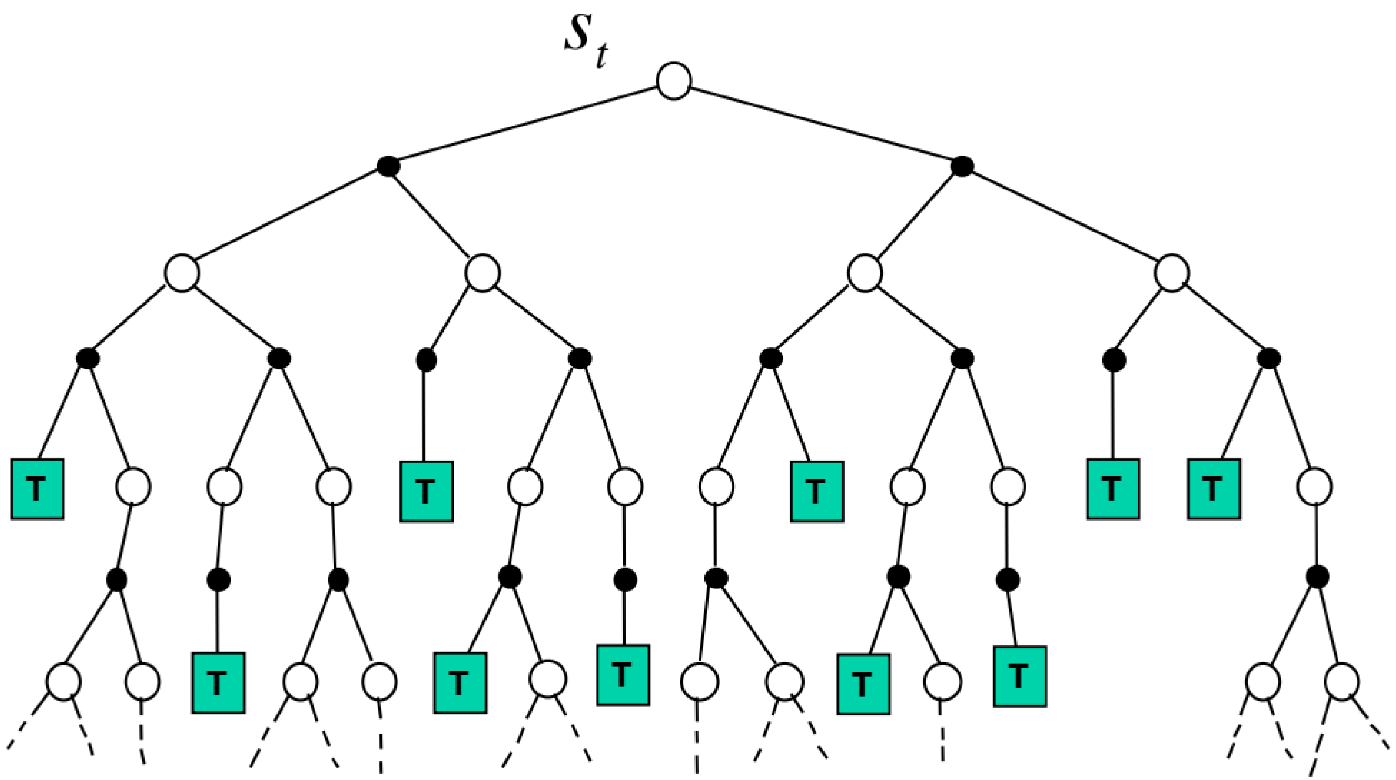


Dyna-Q Algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow \varepsilon$-greedy(S, Q)
  (c) Execute action A; observe resultant reward, R, and state, S'
  (d) $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
  (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
    $R, S' \leftarrow Model(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

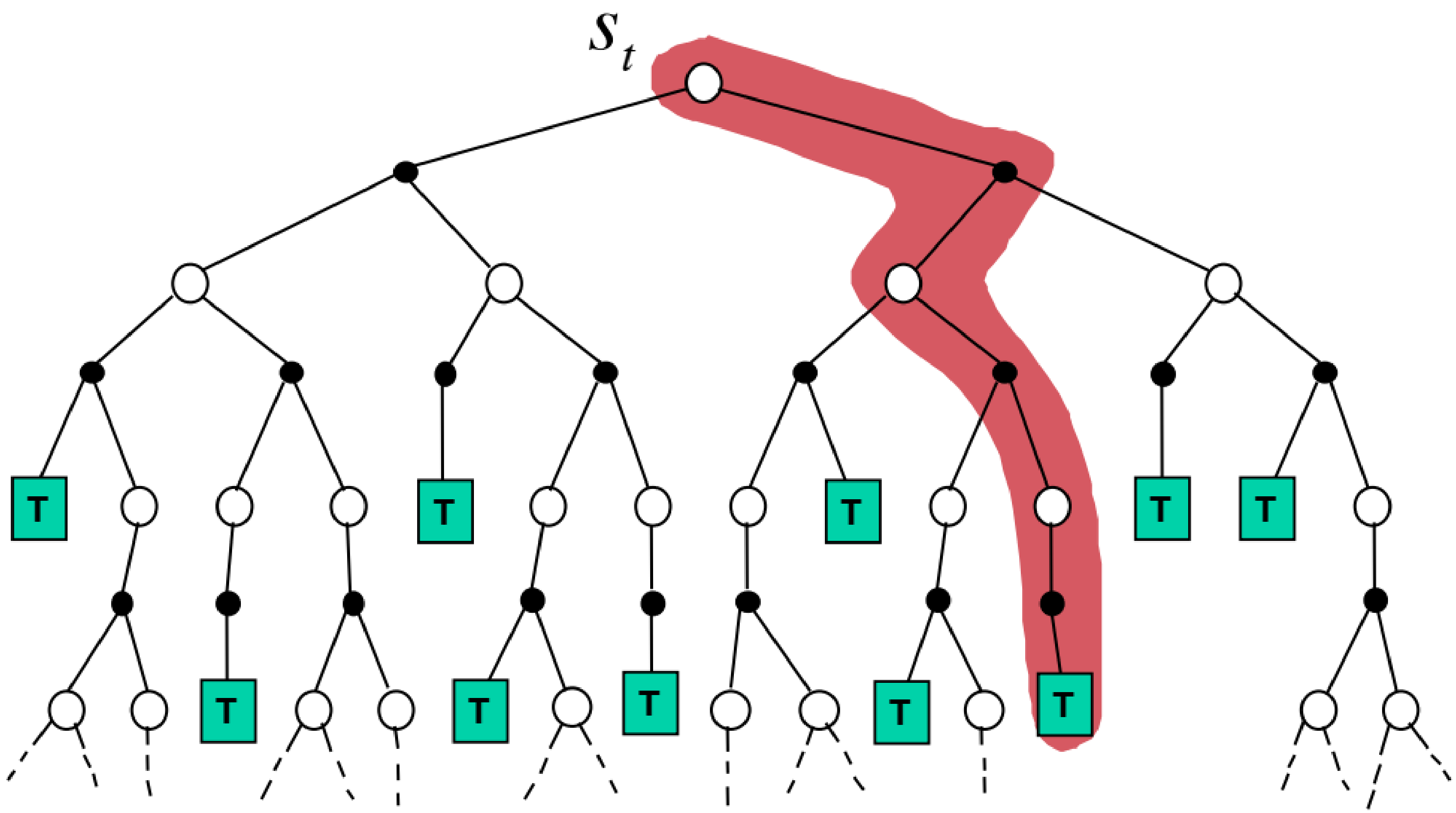
Learning with Simulation

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 (I)

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

Simulation-Based Search (II)

- ✓ Simulate episodes of experience from now with the model

$$\{s_t^k, A_t^k, R_{t+1}^k; \ldots, S_T^k\} \sim \mathcal{M}_\nu$$

- ✓ Apply model-free RL to simulated episodes
  - ✓ Monte-Carlo control $\Longrightarrow$ Monte-Carlo search
  - ✓ SARSA $\Longrightarrow$ 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

$$\{s_t, a, R_{t+1}^k; S_{t+1}^k, A_{t+1}^k \dots, S_T^k\} \sim \mathcal{M}_\nu, \pi$$

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

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

- ✓Select current (real) action with maximum value

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

Monte-Carlo Tree Search (MCTS) - Evaluate

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

$$\{s_t, A_t^k, R_{t+1}^k; S_{t+1}^k, A_{t+1}^k \dots, S_T^k\} \sim \mathcal{M}_\nu, \pi$$

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

$$Q(s,a) = \frac{1}{\mathrm{N(s,a)}} \sum_{k=1}^{K} \sum_{\mathrm{u=t}}^{\mathrm{T}} \mathbf{1}(S_u, A_u; s, a)\, G_{\mathrm{u}} \rightarrow^{P} q_\pi(\mathrm{s}, a)$$

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

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

Monte-Carlo Tree Search - Simulate

- ✓In MCTS the simulation policy π improves
- ✓Each simulation consists of two phases (in-tree, out-of-tree)
  - ✓Tree policy (improves): pick actions to maximise $Q(S, \mathrm{A})$
  - ✓Default policy (fixed): pick actions randomly
- ✓Repeat (each simulation)
  - ✓Evaluate states $Q(S, \mathrm{A})$ 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, $Q(S, \mathrm{A}) \rightarrow q_*(S, A)$

Advantages of MCTS

- ✓ 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

Temporal-Difference Search

- ✓ Simulation-based search
- ✓ Using TD instead of MC (bootstrapping)
- ✓ MCTS 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 $Q(\mathrm{s}, \mathrm{a})$
- ✓For each step of simulation, update action-values by SARSA

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

- ✓Select actions based on action-values $Q(\mathrm{s}, \mathrm{a})$ (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

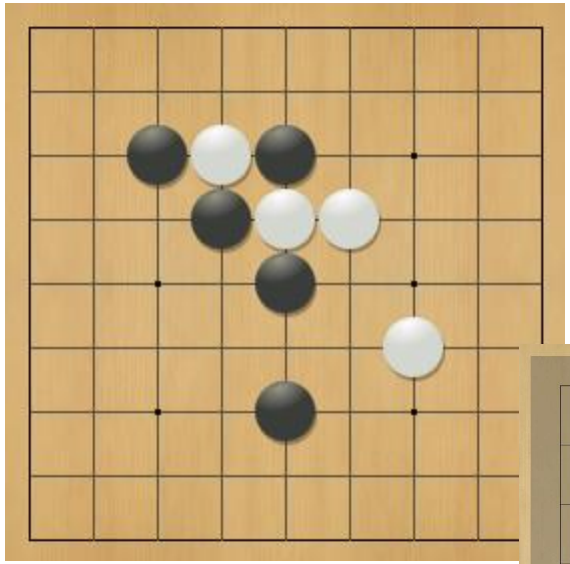
GO Case Study

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 (since McCarthy)
- ✓Traditional game-tree search has long failed in Go

Go Rules

- ✓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



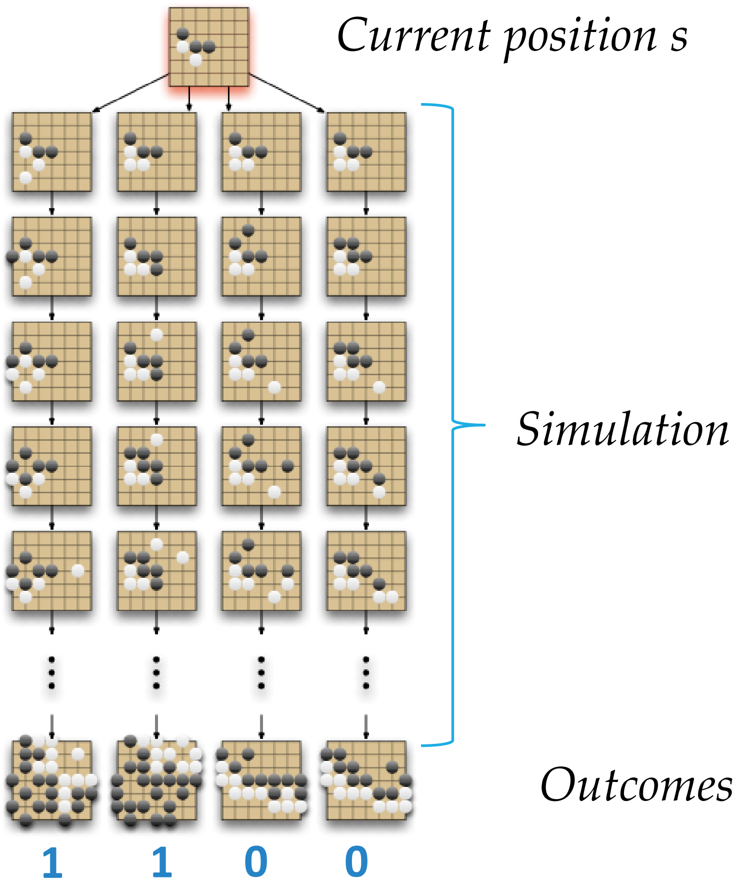
Position Evaluation in Go

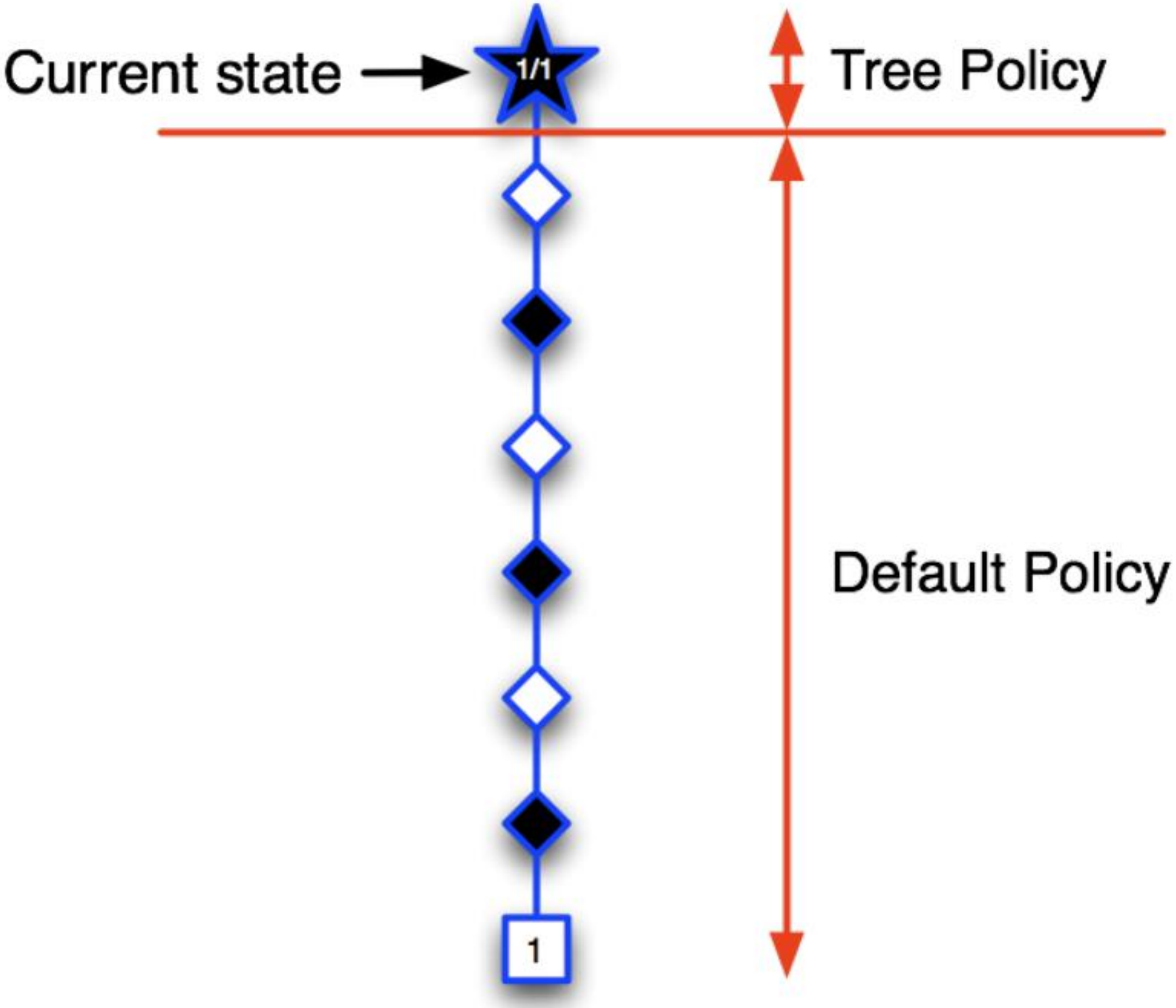
- How good is a position s?
- Reward function (undiscounted):
  - $R_t \;=\; 0$ for all non-terminal steps $t < T$
  - $R_T \;= 1$ if Black wins
  - $R_T \;= 0$ if White wins
- Policy $\pi \;=\; \langle \pi_B, \pi_W \rangle$ selects moves for both players (B,W)
- Value function (how good is position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

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

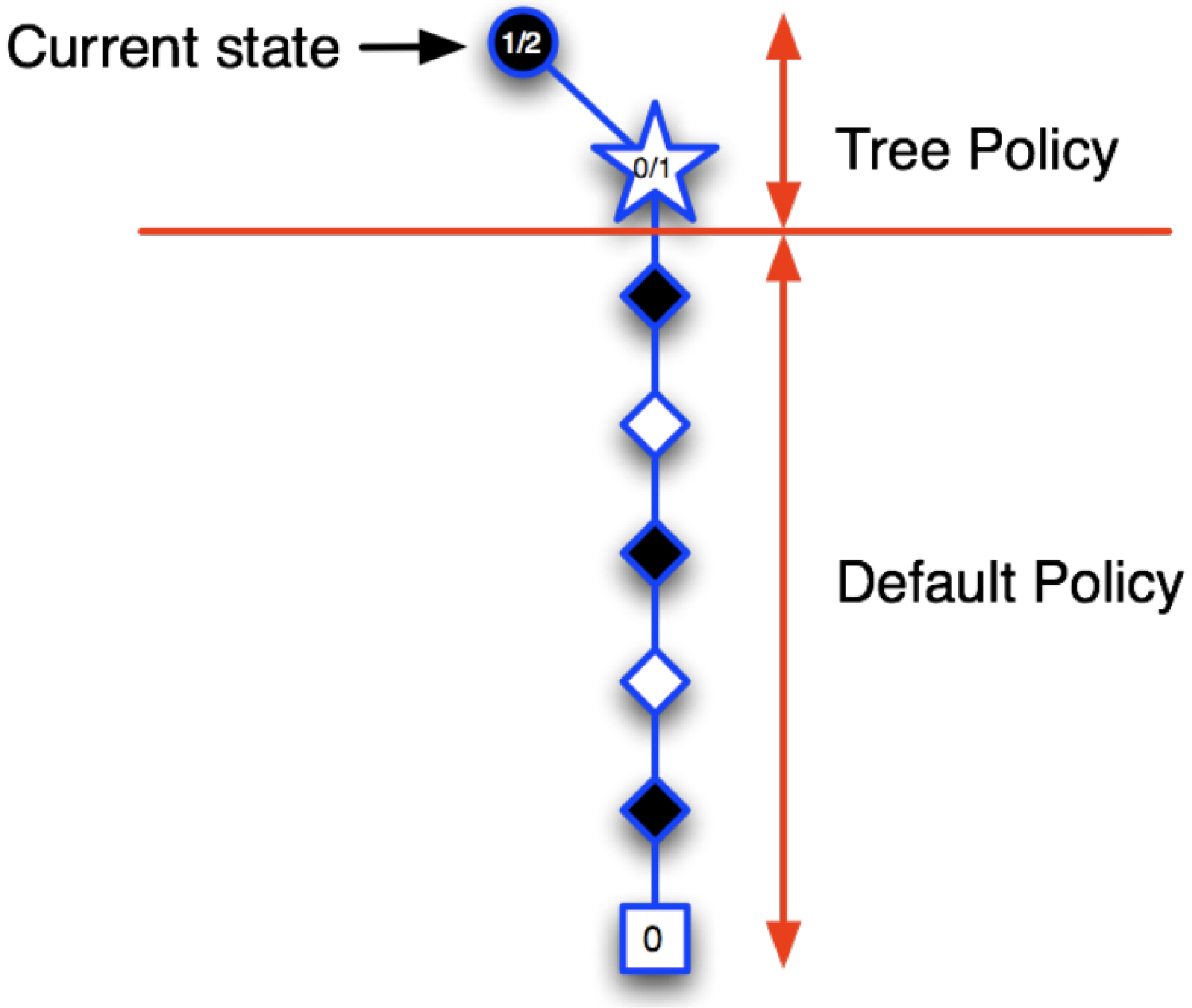
Go – Monte-Carlo Evaluation

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



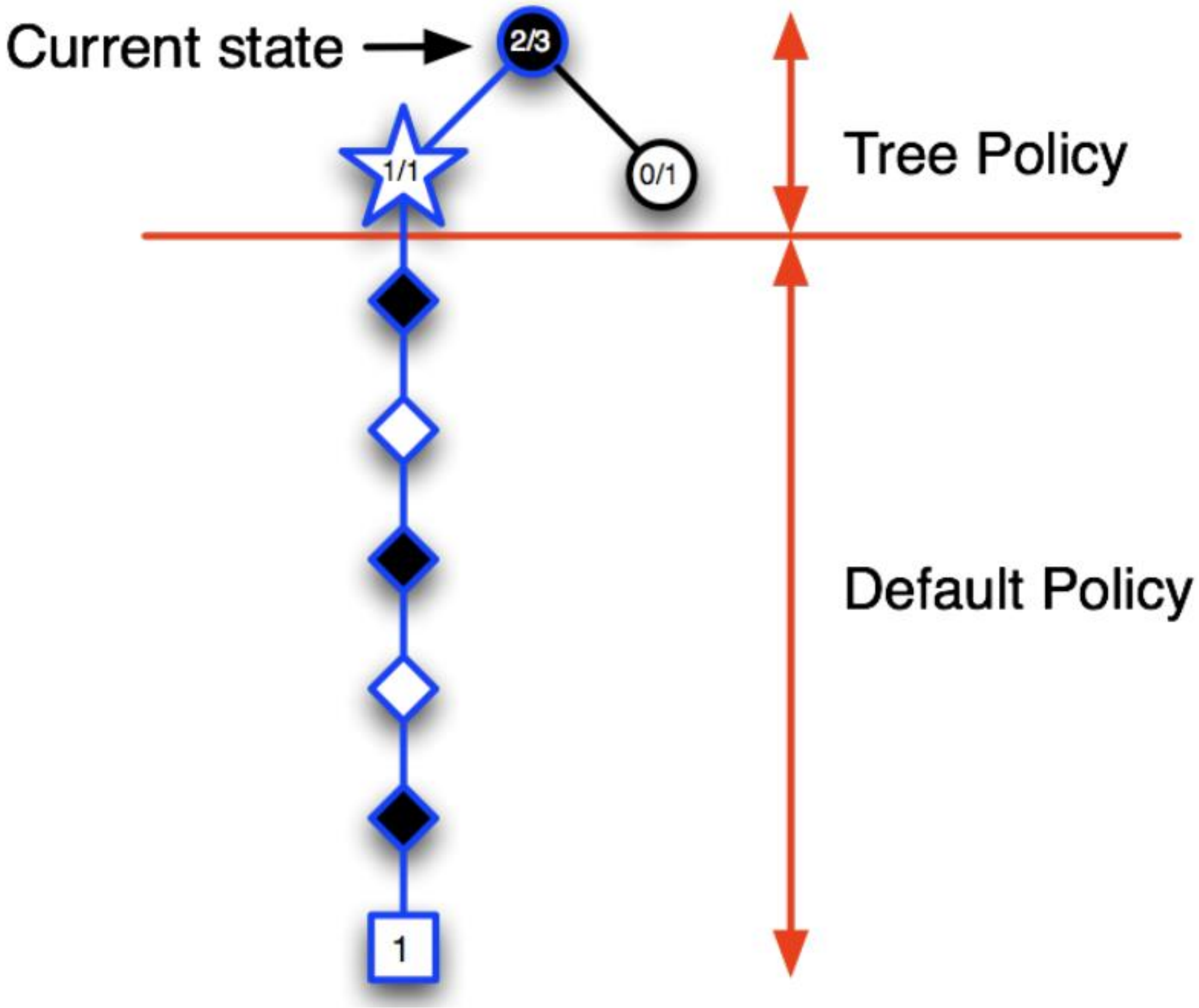


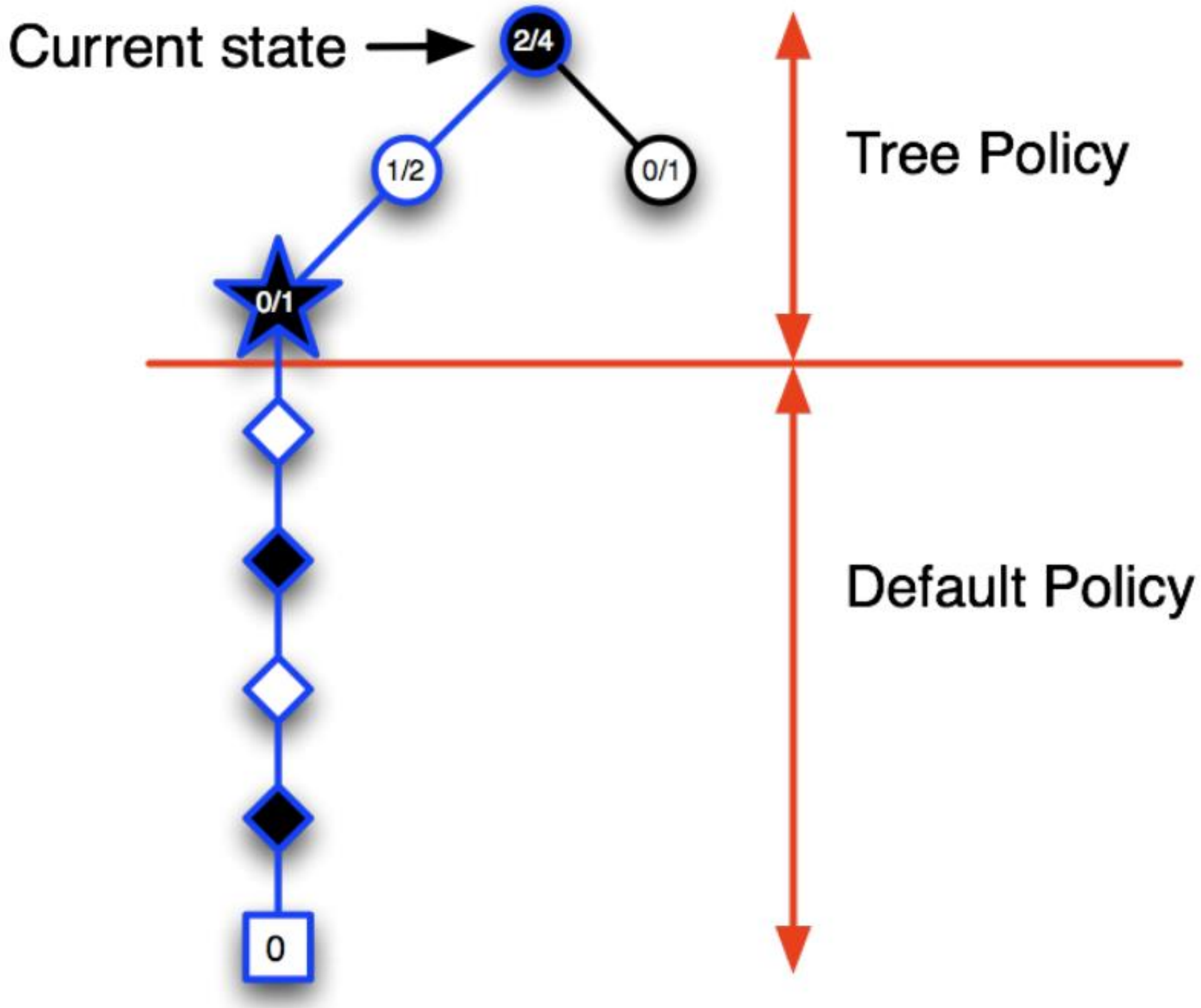
Applying Monte-Carlo Tree Search (I)



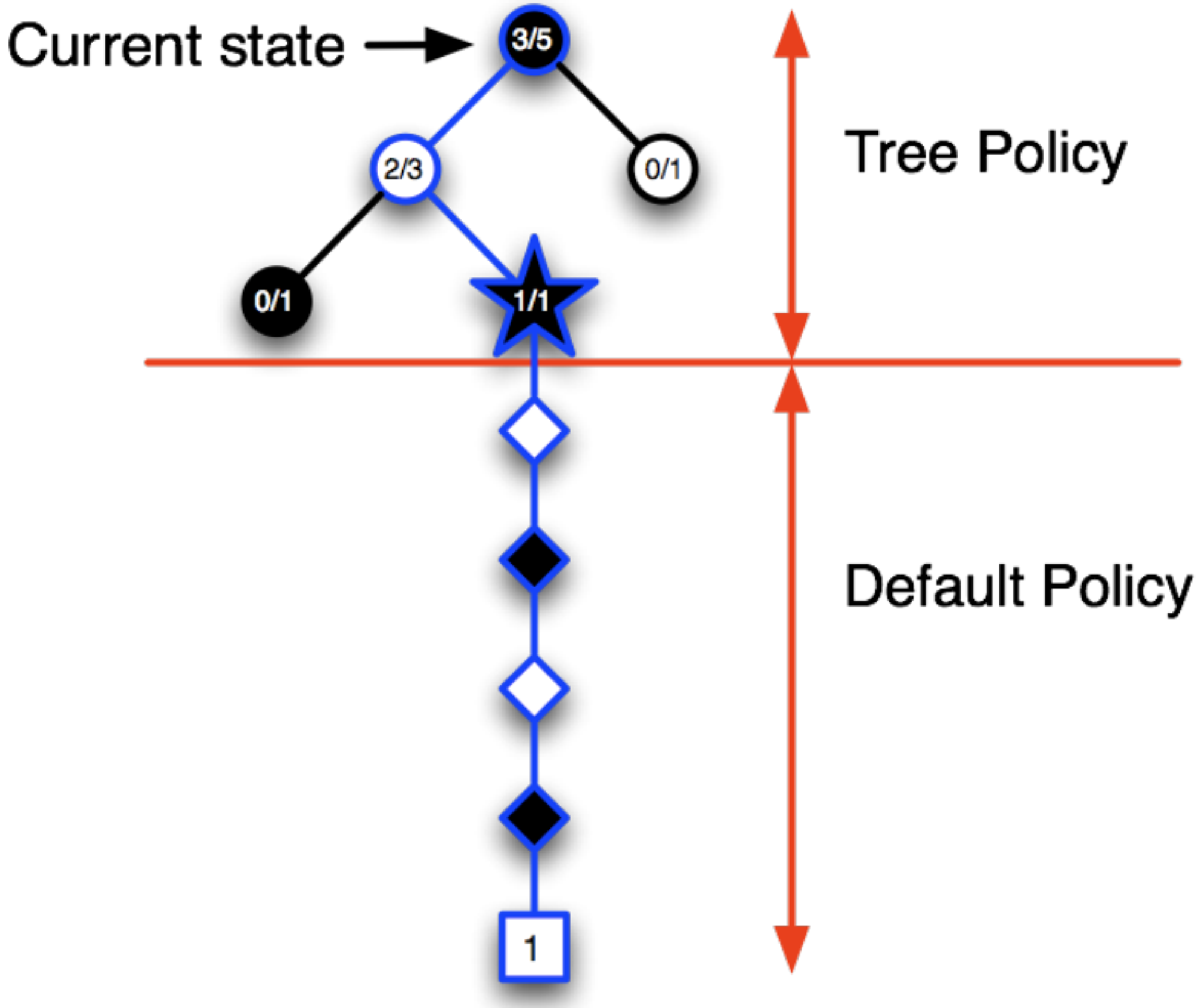
Applying Monte-Carlo Tree Search (II)

Applying Monte-Carlo Tree Search (III)

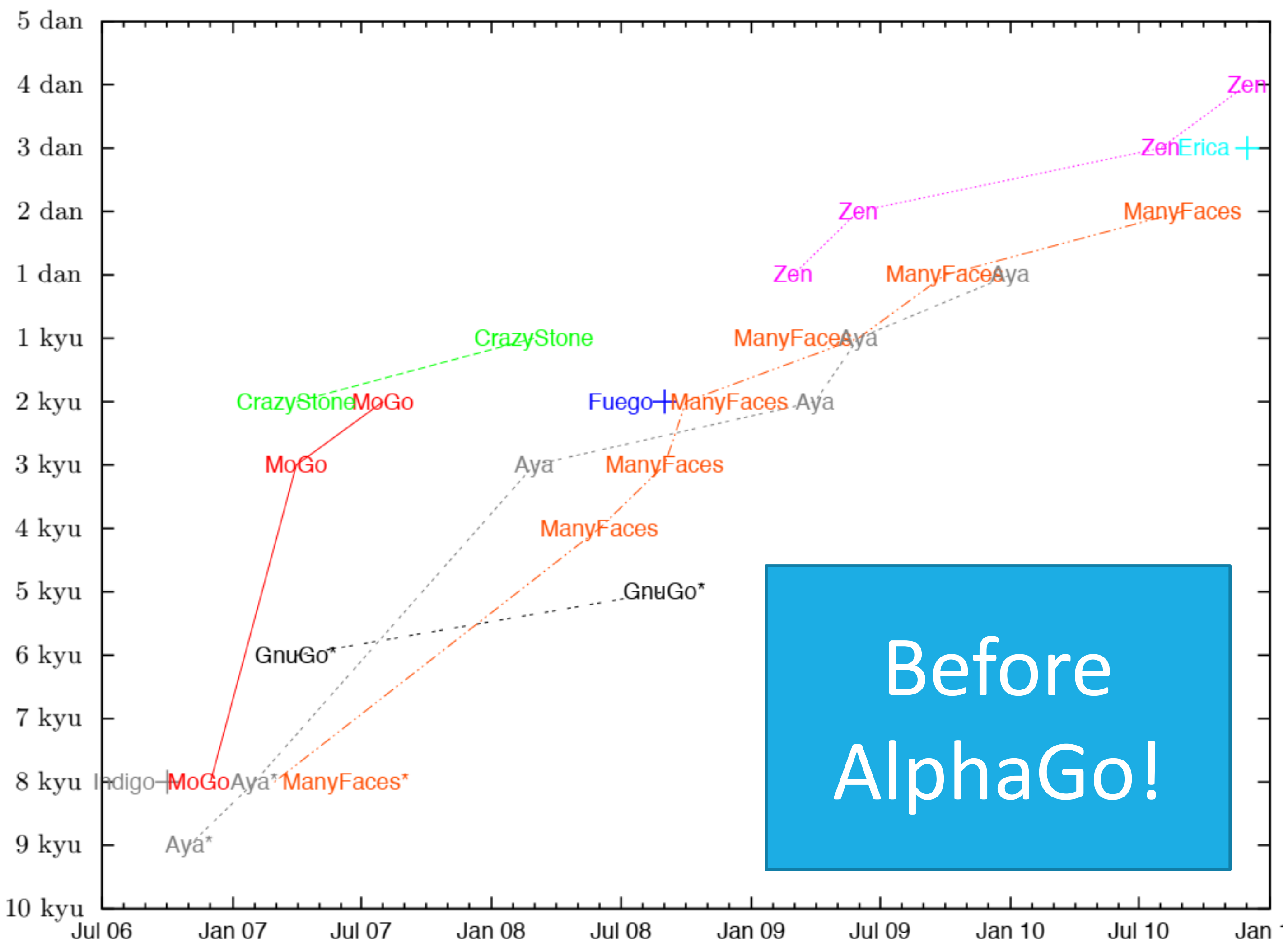




Applying Monte-Carlo Tree Search (IV)



Applying Monte-Carlo Tree Search (V)



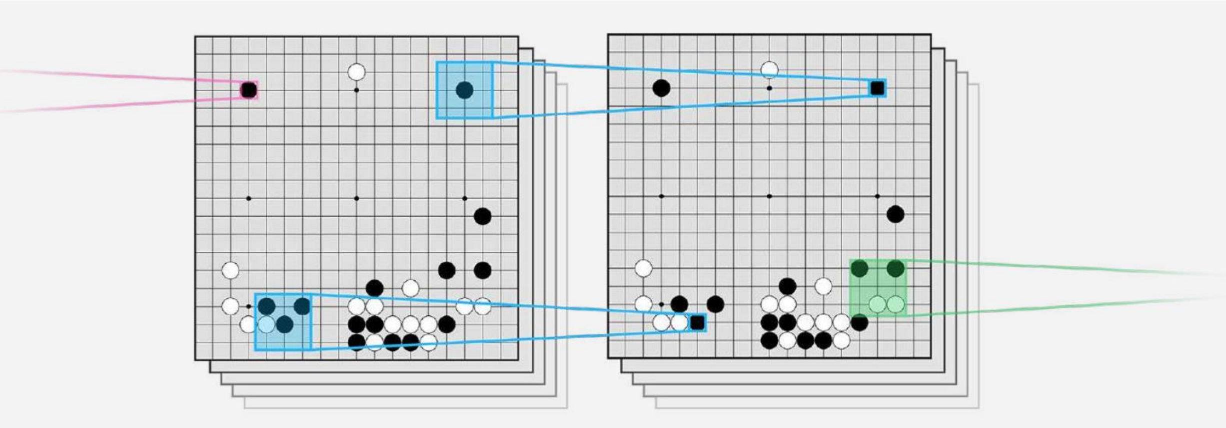
Before AlphaGo!

MCTS in Computer Go

Alpha-Go

- Combining what we have seen so far
  - Value function learning
  - Policy gradient
  - Monte-Carlo Tree Search

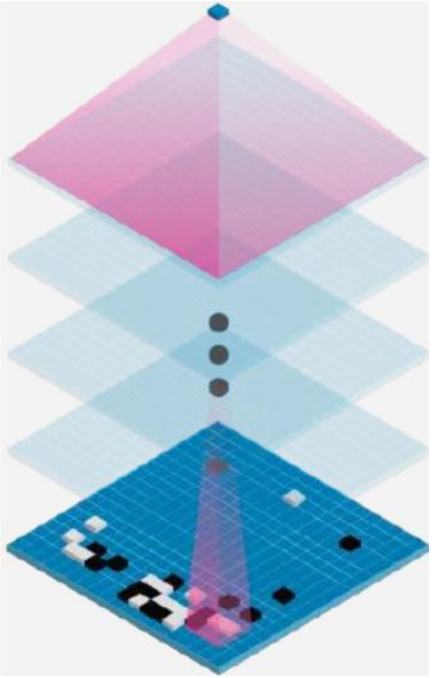
Convolutional neural networks to extract a meaningful state representation



Deep Learning in Alpha-Go

Value network

$V(s)$

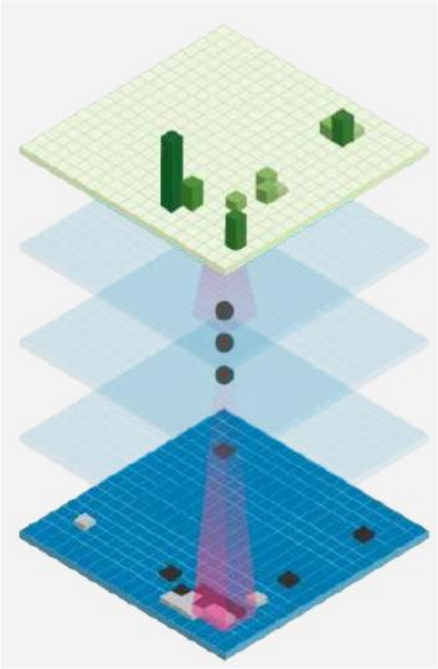


s

Is the player going to win with the current board?

$P(a|s)$

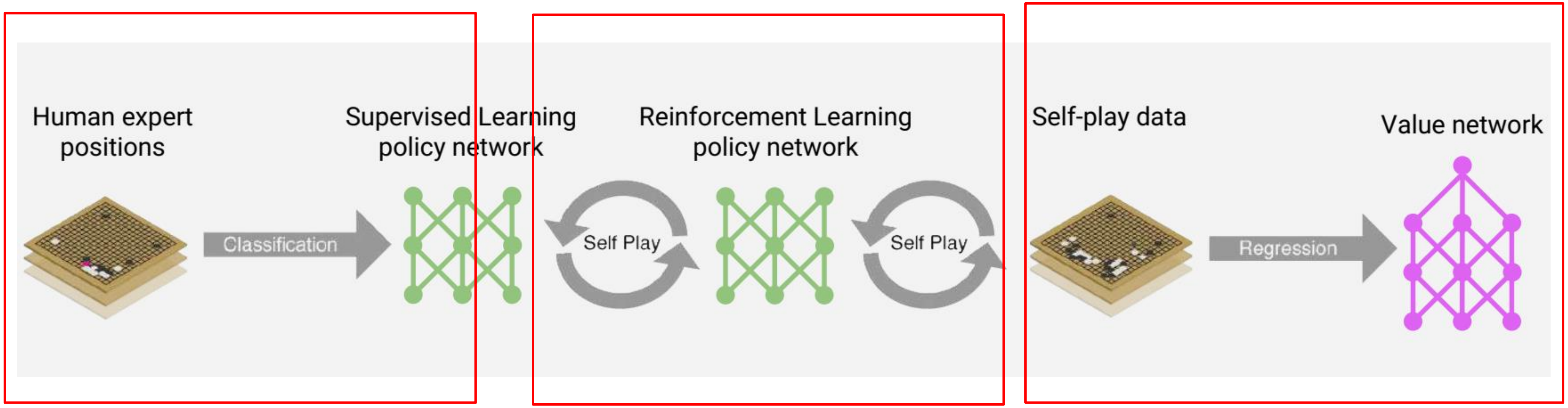
Policy network



s

How much preference for a specific move in the current board?

Supervised-Reinforcement Learning Pipeline (Offline phase)

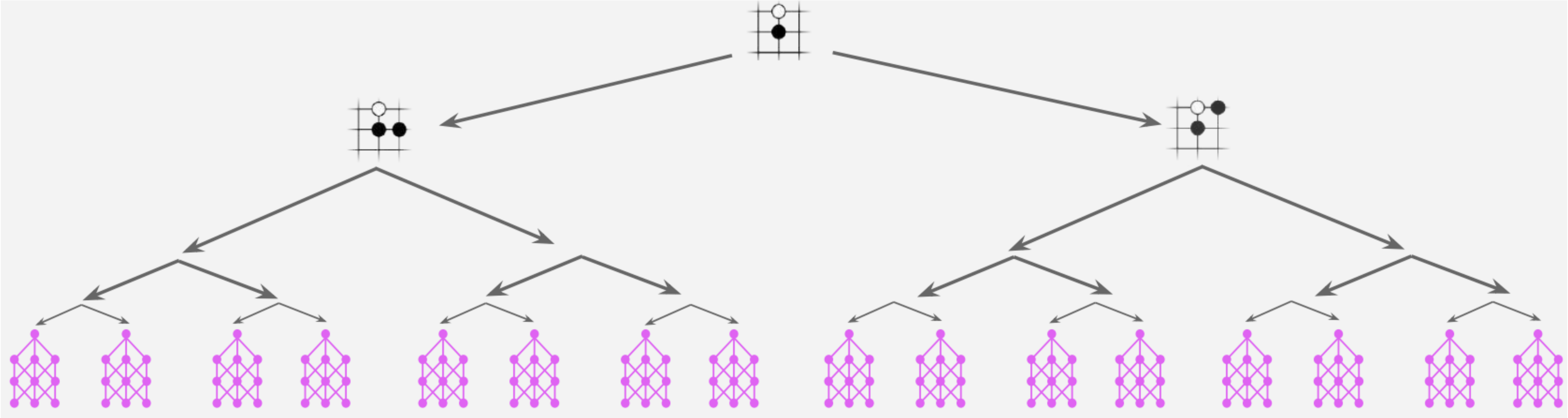


Max likelihood policy on human expert games

Policy gradient on self-play

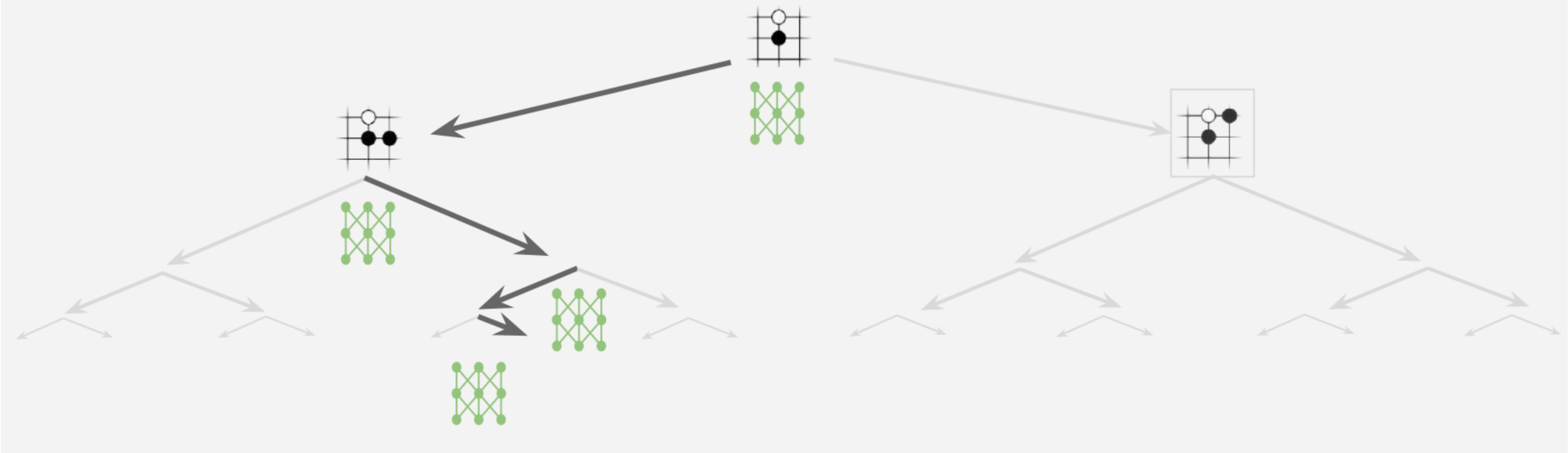
Value function learning on self-play (by policy)

Simulation Phase - Reducing Search Depth



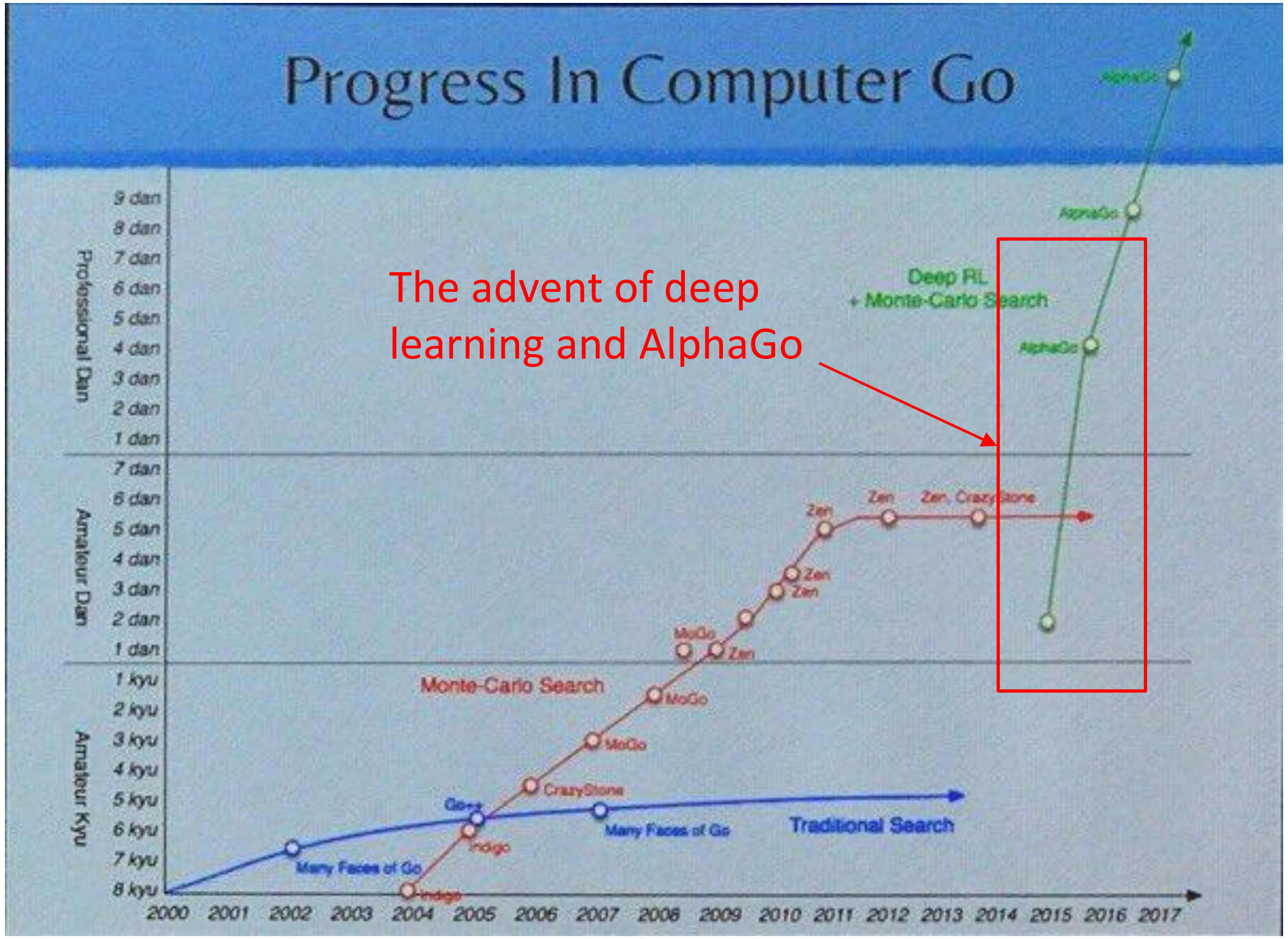
Using learned value network

Simulation Phase - Reducing Search Breadth



Bias MCTS towards most promising branches by policy network

Progress in Computer Go (revised)



Wrap-up

Take home messages

- ✓ Model-based RL is effective
  - ✓ If you know the rules of the world (game) you can use those to simulate experience
  - ✓ Can use the model of the environment to simulate experiences
  - ✓ Can integrate simulated experiences with real world experiences (Dyna)
- ✓ Monte-Carlo Tree Search
  - ✓ Assess the value of an actual state by looking ahead in episodes sampled in simulation
  - ✓ Works for black-box models and it is highly efficient
- ✓ TD Search
  - ✓ Update action-state values by SARSA on simulated episodes
  - ✓ As usual reduces variance w.r.t. MCTS but increases bias
  - ✓ Possibly more efficient than MCTS

Coming up

- ✓ Exploration and Exploitation
  - ✓ Simple naïve exploration (ϵ-greedy)
  - ✓ Optimistic approaches
  - ✓ Probability matching & Information Value
- ✓ Multi-armed bandits
- ✓ Imitation Learning
  - ✓ Demonstration techniques
  - ✓ Inverse reinforcement learning
  - ✓ Reinforcement learning with generative models