Reinforcement Learning
Lecture 6: Planning & Learning

Lecturer: Prof. Dr. Mathias Niepert

Institute for Artificial Intelligence
Machine Learning and Simulation Lab

University of Stuttgart
Germany

impers-is

June 6, 2024

Outline

1. Recap
2. Using models
3. Dyna algorithm
4. Prioritized Sweeping
5. Simulation-based Search

Lecture to Book Chapter Mapping

The exam will cover topics covered in lectures 1-9

1. Multi-armed bandits; chapters: 2.1, 2.2, and 2.4 (+ softmax policy)
2. MDPs (definition, values function, etc.); chapters: 3.1-3.6
3. Policy improvement with dynamic programming; chapters 4.1-4.4
4. Monte-Carlo methods; chapter 5.1-5.7
5. Temporal difference methods; 6.1, 6.2, and 6.4-6.6
6. Planning and Learning; chapters: 8.1-8.4, 8.10-8.11
7. Function approximation; chapter: 9.1-9.4, 9.5.4
8. n-step bootstrapping (no eligibility traces!); chapters: 7.1-7.3
9. Policy gradient methods; chapters: 13.1-13.6

Recap

Transition Function and Reward

Transition function:
Choosing action a in state s, what is the **probability of transitioning to state** s'?

$$p(s' \mid s, a) = \Pr\left\{S_{t+1} = s' \mid S_t = s, A_t = a\right\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a)$$

Reward function:
Choosing action a in state s and transitioning to s', what is the **immediate reward**?

$$r(s, a, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

Important: $r(s, a, s')$ is a function but an expectation (average) over all possible rewards – typically and unless otherwise specified, we assume there is a single reward for each (s, a, s') and we can drop $\mathbb{E}$

Reward definitions

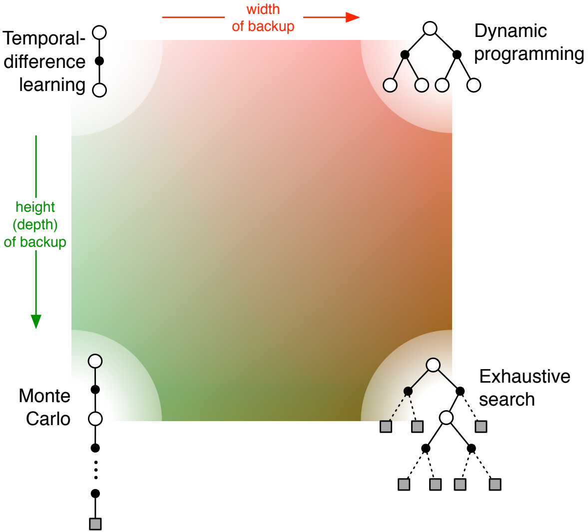
- $r(s, a, s')$: expected immediate reward on transition from s to s' under action a
- $r(s, a)$: expected immediate reward starting in s and choosing action a

$$r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a)$$

- $r(s)$: expected immediate reward for *being* in state s
  - "bag of treasure" sitting on a grid-world square

Using models

Unified view



Models

- **Model:** anything the agent can use to predict how the environment will respond to its actions
- **Distribution model:** Full probability distribution with all possible combinations and their probabilities
$$p(s', r \mid s, a) \text{ for all } s, a, r, s'$$
- **Sample model:** We can only take samples, no access to full distribution
  - produces sample experiences for given s, a
  - often much easier to come by
- Both types of models can be used to produce *hypothetical experience*

Planning

- **Planning:** any computational process that uses a model to create or improve a policy

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

- Planning in AI:
  - state-space planning: search through the state space to find optimal path/policy
  - plan-space planning: search through the space of plans (not our focus)
- We take the following (unusual) view:
  - all state-space planning methods involve computing value functions, either explicitly or implicitly to improve the policy
  - they compute value functions through backups on simulated experience

$$\text{model} \rightarrow \text{sim. experience} \xrightarrow{\text{backups}} \text{value} \rightarrow \text{policy}$$

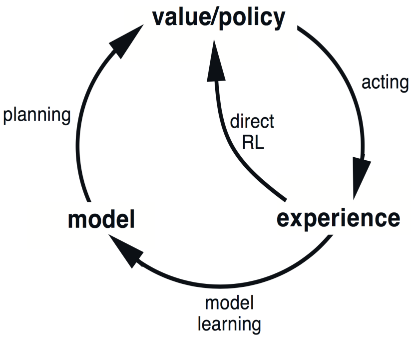
Planning

- Classical DP methods are state-space planning methods
- Heuristic search methods are state-space planning methods
- A planning method based on Q-learning:

repeat
  Select a state $S \in \mathcal{S}$ and action $A \in \mathcal{A}(S)$ at random
  Send S, A to a sample model, obtain a sampled next reward R and next state S'
  Apply one-step tabular Q-learning to S, A, R, S':
  $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
until termination criterion reached

Learning, planning, and acting

- Two uses of real experience:
  - **model learning:** to improve the model
  - **direct RL:** to directly improve the value function and policy
- Improving value function and/or policy via a model is sometimes called **indirect RL**, here, we call it **planning**



Direct (model-free) vs. indirect (model-based) RL

Direct methods

- simpler
- not affected by bad model

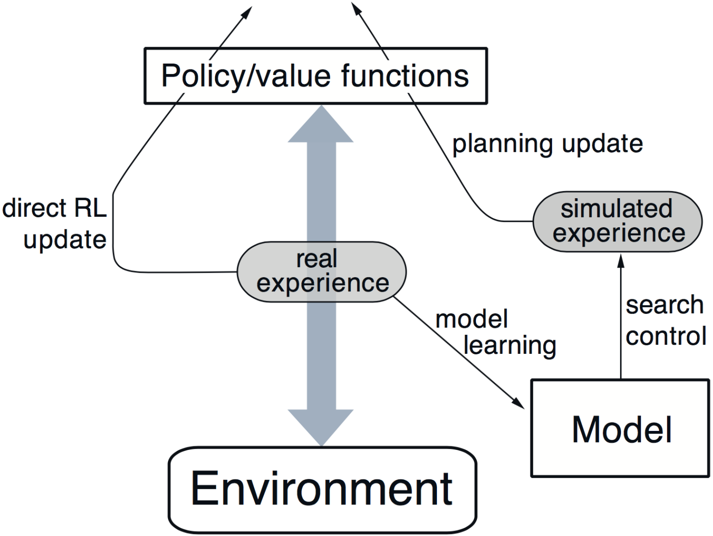
Indirect methods

- make fuller use of experience
- get better policy with fewer environment interactions

Both are very closely related and can be usefully combined: planning, acting, model learning, and direct RL can occur *simultaneously* and in *parallel*

Dyna algorithm

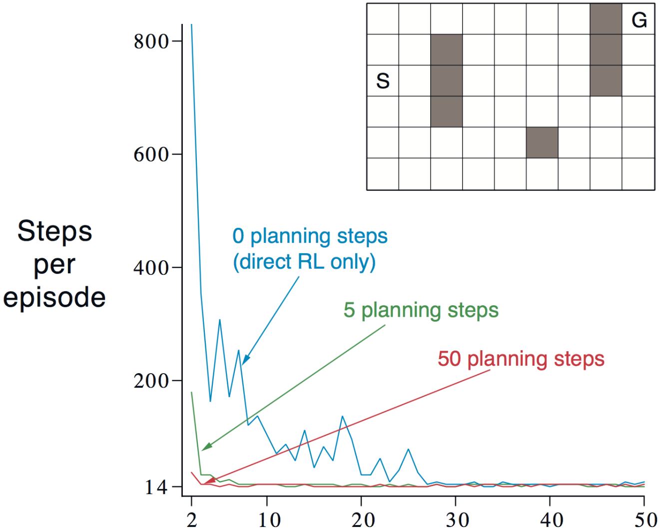
Dyna architecture



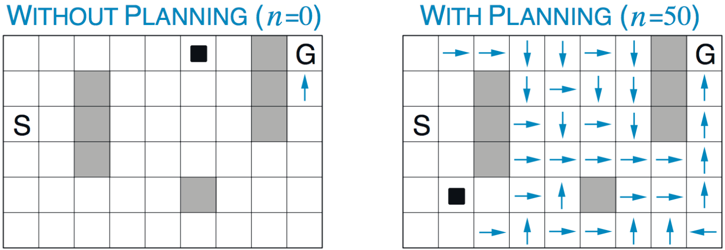
Dyna-Q algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
repeat
  $S \leftarrow$ current (nonterminal) state
  $A \leftarrow \epsilon$-greedy(S, Q)
  Take action A; observe reward R and state S'
  $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ ▷ direct RL
  $Model(S, A) \leftarrow R, S'$ ▷ (deterministic) model learning
  loop repeat n times: ▷ Planning
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in S
    $R, S' \leftarrow Model(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
  end loop
until termination criterion reached

Dyna-Q example: simple maze



Dyna-Q example: snapshots



Halfway through the second episode.

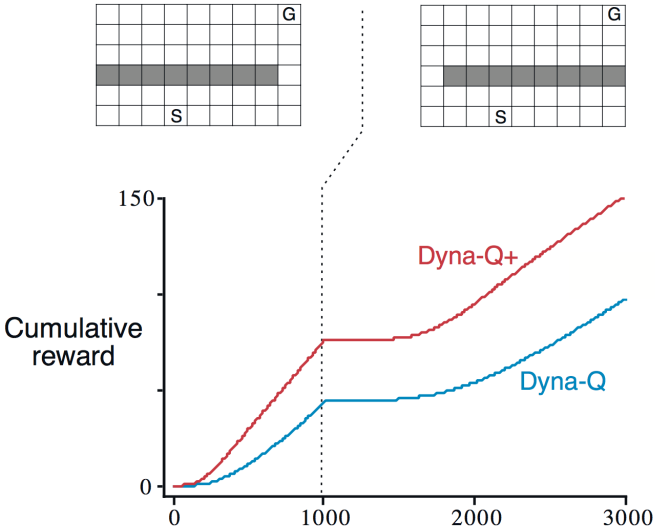
Dyna-Q+

- Uses an *exploration bonus*
- Keeps track of time τ since each state-action pair was tried for real
- Extra reward is added for transitions caused by state–action pairs related to how long ago they were tried
- The longer unvisited, the more reward for visiting:

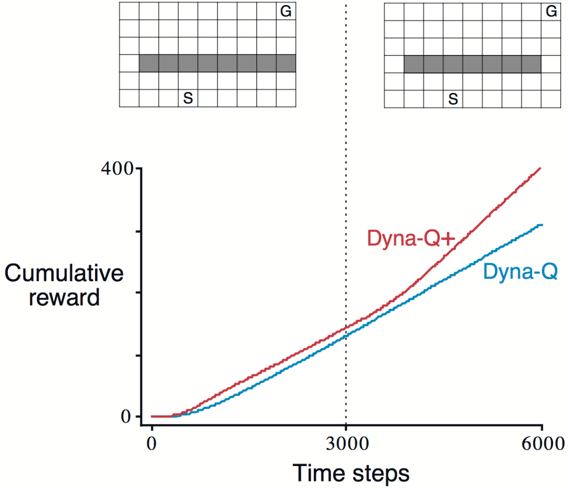
$$R + \kappa\sqrt{\tau}$$

- Agent actually *plans* how to visit long unvisited state–action pairs

Dyna-Q: model errors / blocking maze



Dyna-Q: model errors / shortcut maze



Prioritized Sweeping

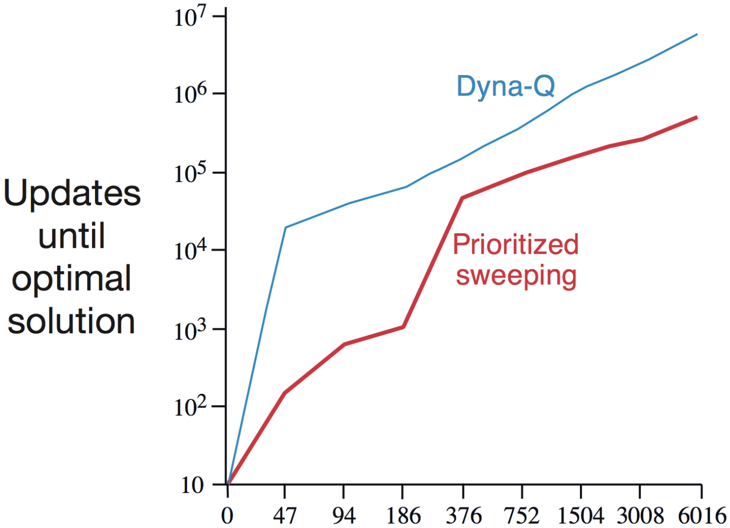
Prioritized sweeping

- Which states or state-action pairs should be generated during planning?
- Work backwards from states whose values have just changed:
  - maintain a queue of state-action pairs whose values would change a lot if backed up, prioritized by the size of the change
  - when a new backup occurs, insert predecessors according to their priorities
  - always perform backups from first in queue

Prioritized sweeping

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ and $PQueue$ to empty
repeat
  $S \leftarrow$ current (nonterminal) state
  $A \leftarrow policy(S,Q)$
  Take action A; observe reward R and state S'
  $Model(S,A) \leftarrow R, S'$
  $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$
  if $P > \theta$ **then** insert S, A into $PQueue$ with priority P
  loop repeat n times, while $PQueue$ is not empty:
    $S, A \leftarrow first(PQueue)$
    $R, S' \leftarrow Model(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    for all $\overline{S}, \overline{A}$ predicted to lead to S **do**
      $\overline{R} \leftarrow$ predicted reward for $\overline{S}, \overline{A}, S$
      $P \leftarrow |\overline{R} + \gamma \max_a Q(S,a) - Q(\overline{S}, \overline{A})|$
      if $P > \theta$ **then** insert $\overline{S}, \overline{A}$ into $PQueue$ with priority P
    end for
  end loop
until termination criterion reached

Prioritized sweeping vs. Dyna-Q



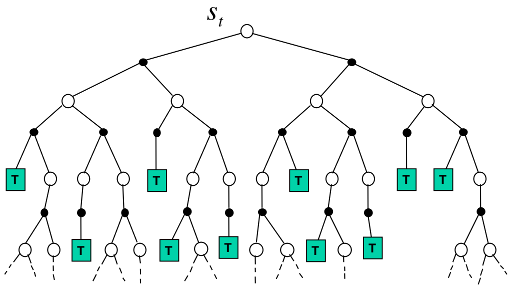
Simulation-based Search

Decision Time Planning

- So far: planning to improve the overall policy
- Decision time planning: Planning after encountering a new state S_t
- Objective of planning here is to determine the next action A_t
- The Simplest version is a one-step look-ahead (compare value of states one can reach)

Forward Search

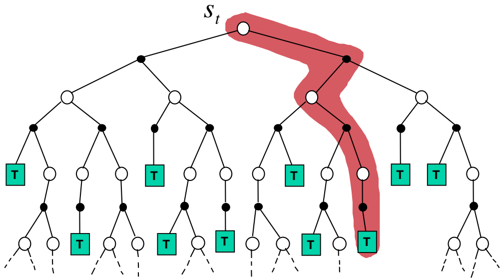
- Forward search algorithms select the best action by lookahead
- They build a **search tree** with the **current state** s_t at the root
- Using a model of the MDP to look ahead



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

Rollout Algorithms

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



Rollout Algorithms (2)

- Simulate episodes of experience from now with the model

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

- 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 policy π
- For each action $a \in \mathcal{A}$
  - Simulate K episodes from current (real) state s_t

$$\{s_t, a_t, R^k_{t+1}, S^k_{t+1}, A^k_{t+1}, \ldots, S^k_T\}_{k=1}^K \sim \mathcal{M}_v$$

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

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

- Select current (real) action with maximum value

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

Monte-Carlo Tree Search

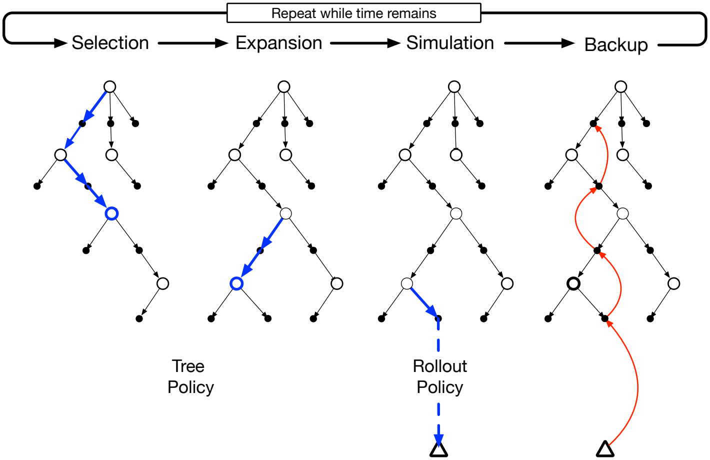
- **Selection.** Starting at root node, a tree policy based on the action values of the edges traverses tree to select a node that can still be expanded
- **Expansion.** On some iterations, the tree is expanded from the selected node by adding one or more children
- **Simulation.** From a selected node (or some of its newly added children), continue with the rollout policy (a simple policy such as random)
- **Backup.** The return generated by the simulation is used to update the action values on the existing tree, that is, the tree policy is updated.

When planning time has run out, select next action based on computed action values of tree. Largest action value from root node of the tree or most visited action.

Monte-Carlo Tree Search (Simulation)

- In MCTS, the tree policy π improves in each time step
- Each simulation consists of two phases (in-tree, out-of-tree)
  - **Tree policy** (improves): For instance ϵ-greedy(Q)
  - **Rollout policy** (fixed): sample actions using this policy
- Repeat (each simulation)
  - Evaluate states $Q(S, A)$ by Monte-Carlo evaluation
  - Improve tree policy using MC control
- Monte-Carlo control applied to simulated experience
- Converges on the optimal search tree, $Q(S, A) \rightarrow q_*(S, A)$

Monte-Carlo Tree Search (Algorithm)

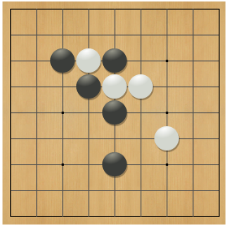


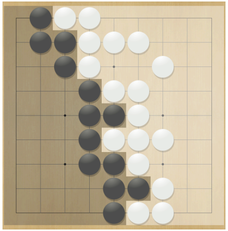
Case Study: the Game of Go

- 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





Position Evaluation in Go

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

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

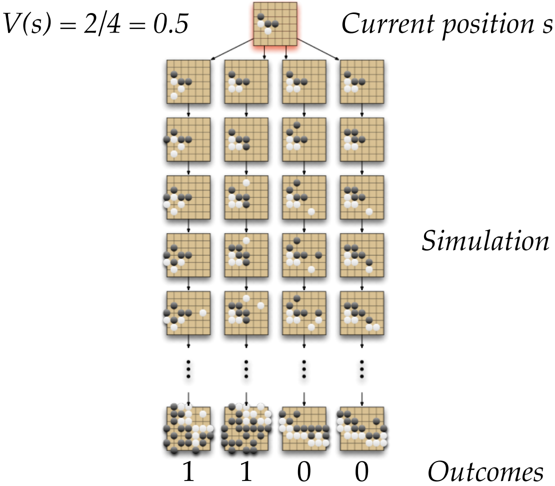
$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

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

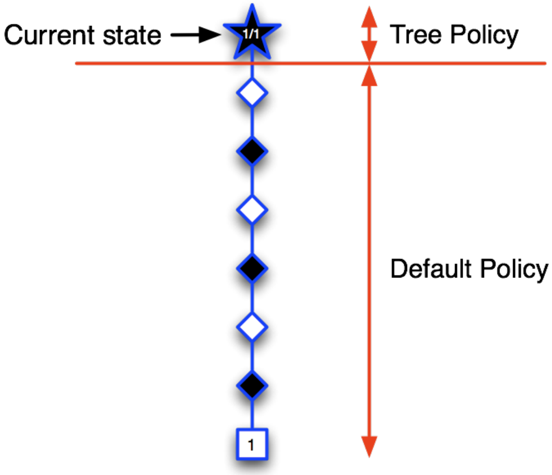
$$v_\pi(s) = \mathbb{E}_\pi[R_T \mid S = s] = \mathbb{P}[\text{Black wins} \mid S = s]$$

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

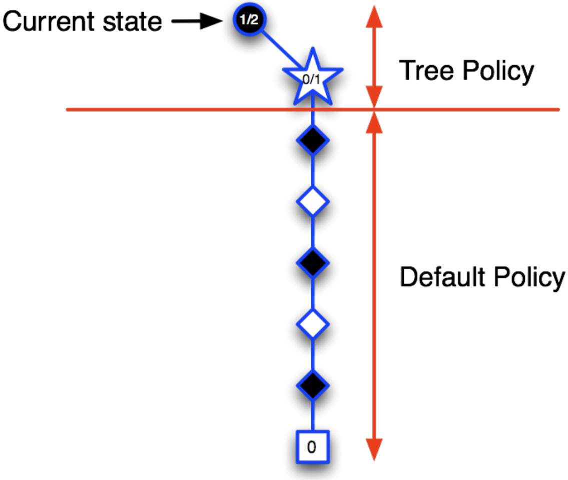
Monte-Carlo Evaluation in Go



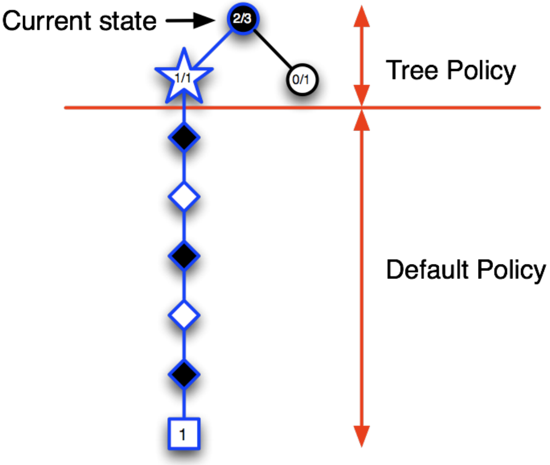
Applying Monte-Carlo Tree Search (1)



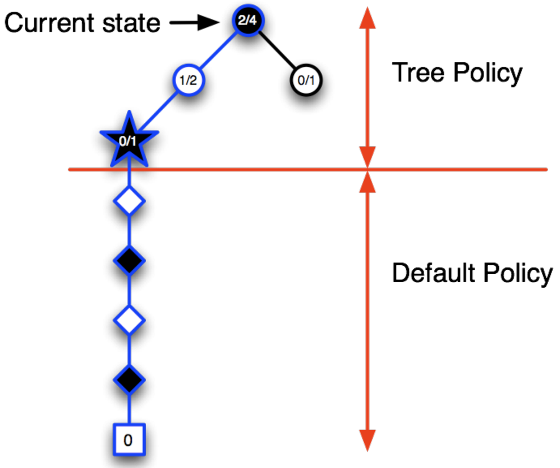
Applying Monte-Carlo Tree Search (2)



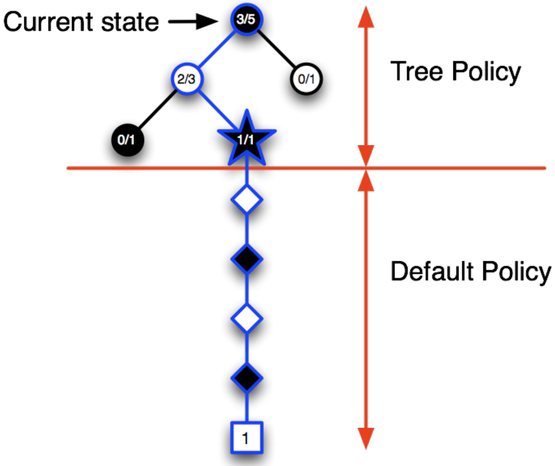
Applying Monte-Carlo Tree Search (3)



Applying Monte-Carlo Tree Search (4)



Applying Monte-Carlo Tree Search (5)

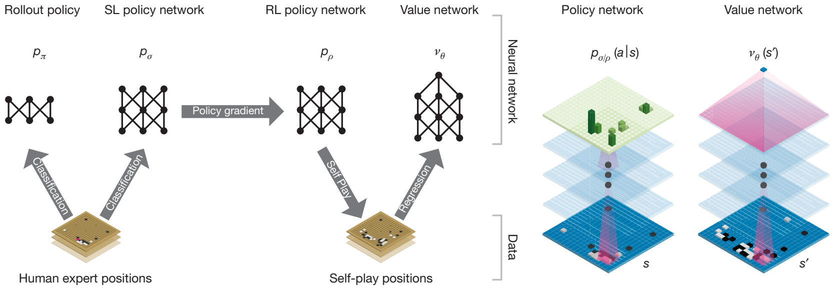


Advantages of MC Tree Search

- 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

Success AlphaGo

- AlphaGo [Silver 16]
  - Train a policy $p_\sigma(\mathbf{a}|\mathbf{s})$ network
  - Improve the policy by RL and self-play
  - Train a value network $v_\theta(\mathbf{s}')$
- $p_\sigma(\mathbf{a}|\mathbf{s})$ is a 13-layer DNN with alternating convolutions and ReLUs, with output soft-max layer (probabilities over $\mathbf{a}$)

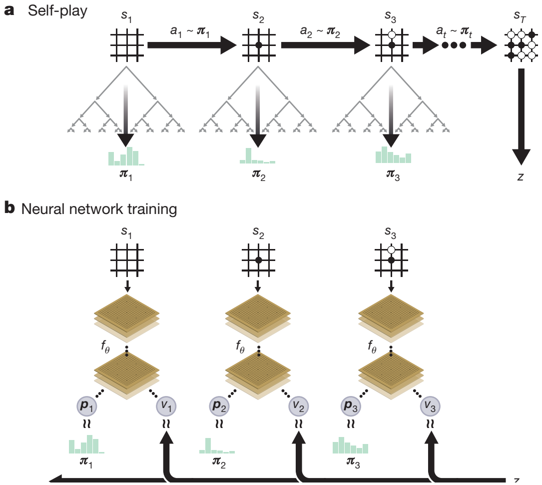


[source Nature Silver 16]

Success AlphaGo Zero

AlphaGo Zero [Silver 17]

- Start *tabula rasa*
- Policy improvement: MCTS
- Policy evaluation: Self Play
  Learn From Win/Loss



[source Nature Silver 17]

Summary

- Emphasized close relationship between planning and learning
- Important distinction between *distribution models* and *sample models*
- Looked at some ways to integrate planning and learning
  - synergy among planning, acting, model learning
- Distribution of backups: focus of the computation
  - prioritized sweeping
  - small backups
  - sample backups
  - (trajectory sampling)
  - (heuristic search)
- Size of backups:
  - full/sample
  - deep (n–step) / shallow (one–step)