Lecture 7: Planning and Models

Hado van Hasselt

UCL 2018

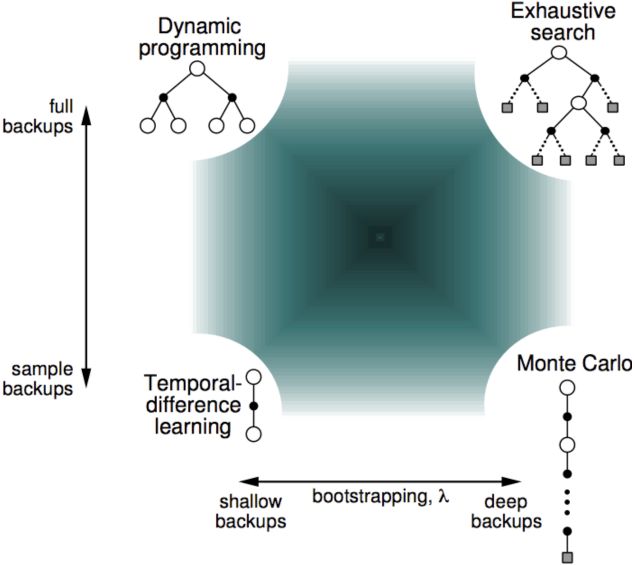
Model-Based Reinforcement Learning

- Last lecture: learn policy directly from experience
- Previous lectures: learn value function directly from experience
- This lecture:
  - Learn a model directly from experience (or be given a model)
  - Plan with the model to construct a value function or policy

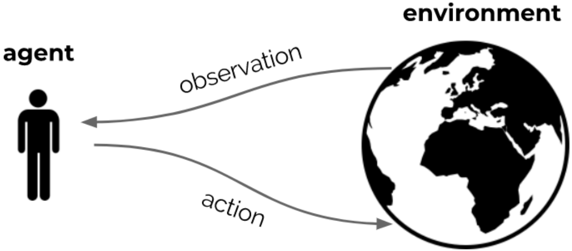
Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
- Model-Based RL
  - Learn a model from experience OR be given a model
  - Plan value function (and/or policy) from model

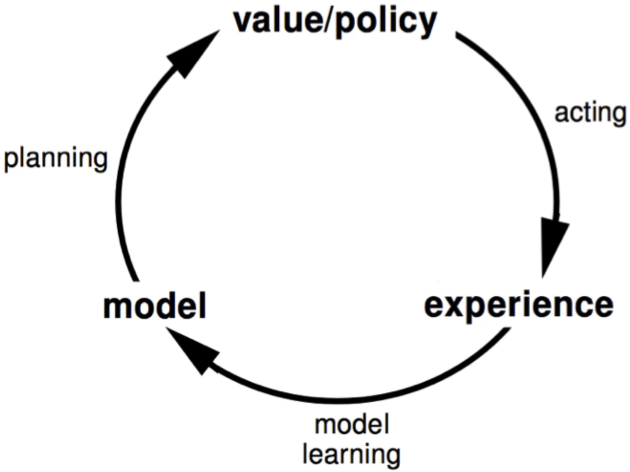
Filling in the middle of algorithm space



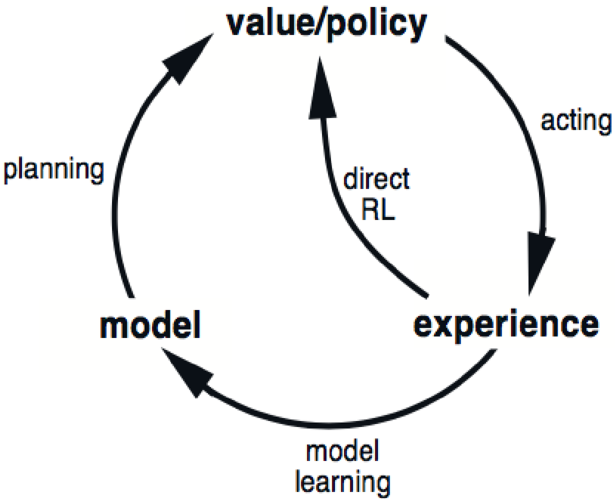
Model-Free RL



Model-Based RL



Model-Based RL



What is a Model?

- A model $\mathcal{M}_\eta$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \hat{p}_\eta \rangle$, parametrized by η
- For now, we will assume the states and actions are the same as in the real problem
- The model approximates the state transitions and rewards $\hat{p}_\eta \approx p$:

$$R_{t+1}, S_{t+1} \sim \hat{p}_\eta(r, s' \mid S_t, A_t)$$

- Optionally, we could model rewards and state dynamics separately

Model Learning

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

$$\begin{gathered} S_1, A_1 \rightarrow R_2, S_2 \\ \vdots \\ S_{T-1}, A_{T-1} \rightarrow R_T, S_T \end{gathered}$$

- Learn a function $f(s, a) = r, s'$
- Pick loss function (e.g. mean-squared error), and find parameters η that minimise empirical loss
- This would give an expectation model
- If $f(s, a) = r, s'$, then we would hope $s' \approx \mathbb{E}[S_{t+1} \mid s = S_t, a = A_t]$

Expectation Models

- Expectation models can have disadvantages:
  - Imaging that a (high-level) action randomly goes left or right past a well
  - The expectation model might interpolate and put you in the wall
- But with linear models and values, we are mostly alright
  - Consider model (=matrix) P with $\mathbb{E}[\phi_{t+1}] = P\phi_t$ and value function $v_\theta(S_t) = \theta^\top \phi_t$
  - Then

$$
\begin{aligned}
\mathbb{E}[v_\theta(S_{t+n}) \mid S_t = s] &= \mathbb{E}[\theta^\top \phi_{t+n} \mid S_t = s] \\
&= \mathbb{E}[\theta^\top P \phi_{t+n-1} \mid S_t = s] \\
&= \ldots \\
&= \mathbb{E}[\theta^\top P^n \phi_t \mid S_t = s] \\
&= \theta^\top P^n \phi(s) \\
&= v_\theta(P^n \phi(s)) \\
&= v_\theta(\mathbb{E}[\phi_{t+n} \mid S_t = s]) \,.
\end{aligned}
$$

Stochastic Models

- We may not want to assume everything is linear
- Then, expected states may not be right — they may not correspond to actual states, and iterating the model may do weird things
- Alternative: stochastic models (also known as generative models)

  $$\hat{R}_{t+1}, \hat{S}_{t+1} = \hat{p}(S_t, A_t, \omega)$$

  where ω is a noise term
- Stochastic models can be chained, even if the model is non-linear
- But they do add noise

Full Models

- Of course, we can try to model the complete transition dynamics, including stochasticity
- It can be hard to iterate these, because of the branching:

$$\mathbb{E}[v(S_{t+1}) \mid S_t = s] = \sum_a \pi(a \mid s) \sum_{s'} \hat{p}(s, a, s')(\hat{r}(s, a, s') + \gamma v(s'))$$

$$\begin{aligned}\mathbb{E}[v(S_{t+n}) \mid S_t = s] = \sum_a \pi(a \mid s) \sum_{s'} \hat{p}(s, a, s') \Bigg(& \hat{r}(s, a, s') + \\ & \gamma \sum_{a'} \pi(a' \mid s') \sum_{s''} \hat{p}(s', a', s'') \Bigg(\hat{r}(s', a', s'') + \\ & \gamma^2 \sum_{a''} \pi(a'' \mid s'') \sum_{s'''} \hat{p}(s'', a'', s''') \Bigg(\hat{r}(s'', a'', s''') + \dots \Bigg)\Bigg)\Bigg)\end{aligned}$$

- However, they can still be useful

Examples of Models

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Deep Neural Network Model
- ...

Table Lookup Model

- Model is an explicit MDP
- Count visits $N(s, a)$ to each state action pair

$$\frac{(s,a,s')}{(s,a)}$$

$$\hat{p}_t(s' \mid s, a) = \frac{1}{N(s, a)} \sum_{k=0}^{t-1} I(S_k = s, A_k = a, S_{k+1} = s')$$

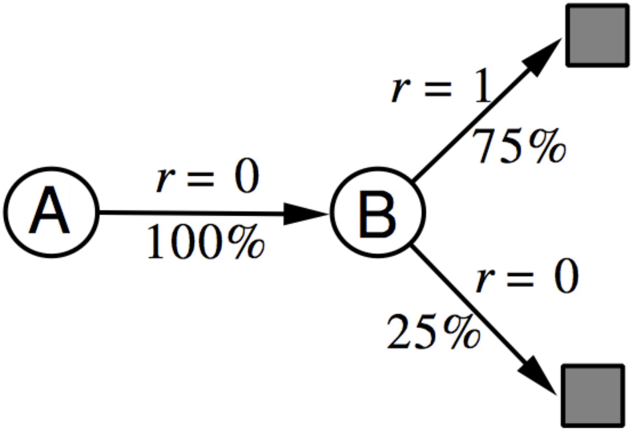
$$\mathbb{E}_{\hat{p}_t}[R_{t+1} \mid S_t = s, A_t = a] = \frac{1}{N(s, a)} \sum_{k=0}^{t-1} I(S_k = s, A_k = a) R_{k+1}$$

- Alternatively, use non-parametric 'replay' model
  - 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 $\hat{p}_\eta$
- Solve the MDP $\langle \mathcal{S}, \mathcal{A}, \hat{p}_\eta \rangle$
- 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

$$S, R \sim \hat{p}_\eta(\cdot \mid s, a)$$

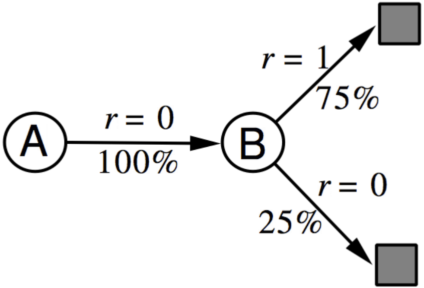
- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning

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		

e.g. Monte-Carlo learning: $V(A) = 1, V(B) = 0.75$

Conventional model-based and model-free methods

Traditional RL algorithms did not explicitly store their experiences, and were often placed into one of two groups.

- Model-free methods update the value function and/or policy and do not have
清楚的 explicit dynamics models.
- Model-based methods update the transition and reward models, and compute a value function or policy from the model.

Moving beyond model-based and model-free labels

The sharp distinction between model-based and model-free methods is becoming somewhat less useful.

1. For tabular RL there is an exact output equivalence between some conventional model-based and model-free algorithms.
2. When the agent stores transitions in an *experience replay buffer* and learns from it (as in DQN), we can think of this stored experience as an implicit model.
3. More generally, an agent can store its experience in other forms. In those cases it is unclear if either or both labels apply.

The terms are still used to describe whether an algorithm is explicitly modeling the environmental dynamics (to a greater or lesser extent), and how the agent is generalizing from past experience.

Using experience in the place of a model

Recall prioritized sweeping from tabular dynamic programming.

- Update the value function of the states with the largest magnitude Bellman errors using a priority queue.

A related idea is prioritized experience replay (Schaul et al, 2015) which works from experience for general function approximation.

- The experience replay buffer maintains a priority for each transition, with the priority given by the magnitude of the Bellman error.
- Minibatches are sampled using this priority to quickly reduce errors.
- Weighted importance sampling corrects for bias from non-uniform sampling.

Limits of Planning with an Inaccurate Model

- Given an imperfect model $\hat{p}_\eta \neq p$
- Performance is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \hat{p}_\eta \rangle$
- Model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a suboptimal policy (not covered in these slides)
  - Approach 1: when model is wrong, use model-free RL
  - Approach 2: reason explicitly about model uncertainty over η (e.g. Bayesian methods)
  - Approach 3: Combine model-based and model-free methods in a safe way.

Real and Simulated Experience

We consider two sources of experience

Real experience Sampled from environment (true MDP)

$$r, s' \sim p$$

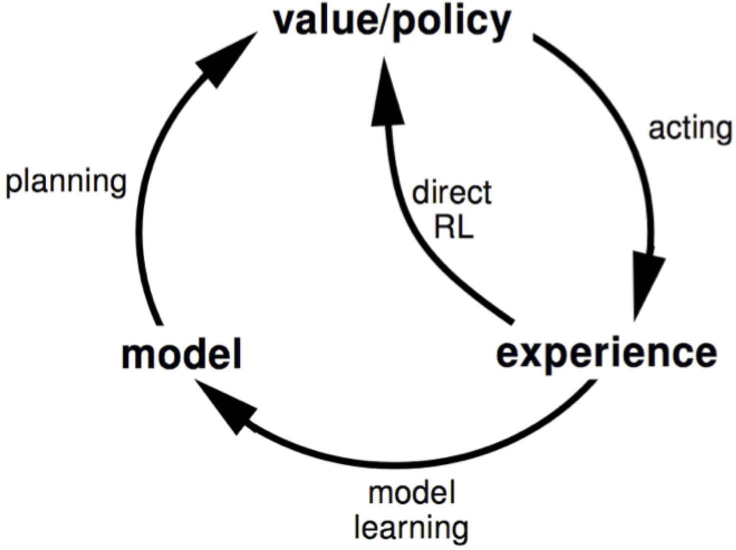
Simulated experience Sampled from model (approximate MDP)

$$r, s' \sim \hat{p}_\eta$$

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
  - Treat real and simulated experience equivalently. Conceptually, the updates from learning or planning are not distinguished.

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 state, s', and reward, r

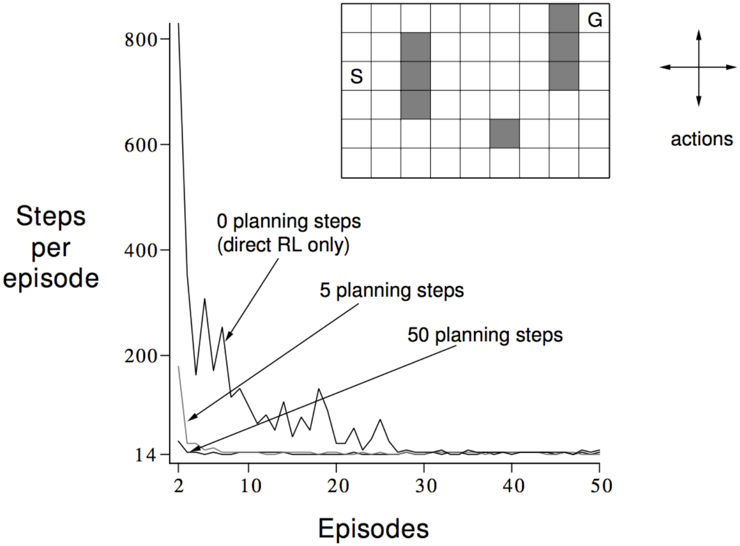
(d) $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

(e) $Model(s, a) \leftarrow s', r$ (assuming deterministic environment)

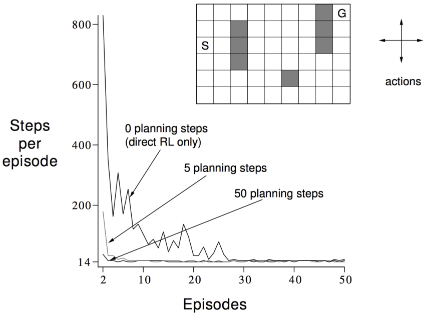
(f) Repeat N times:

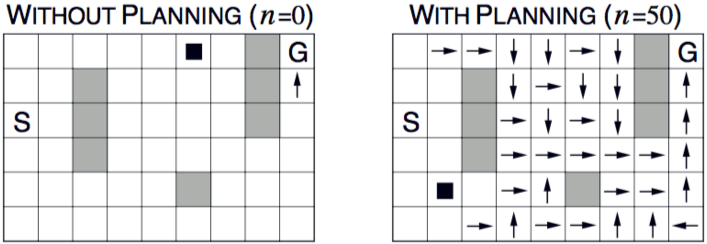
- $s \leftarrow$ random previously observed state
- $a \leftarrow$ random action previously taken in s
- $s', r \leftarrow Model(s, a)$
- $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

Dyna-Q on a Simple Maze



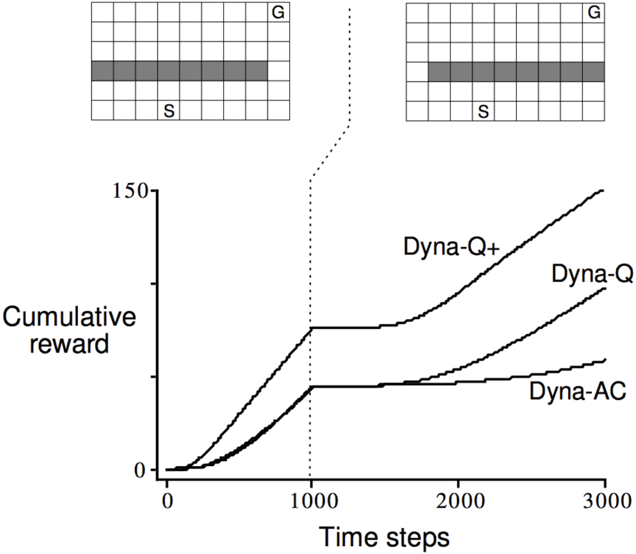
Dyna-Q on a Simple Maze





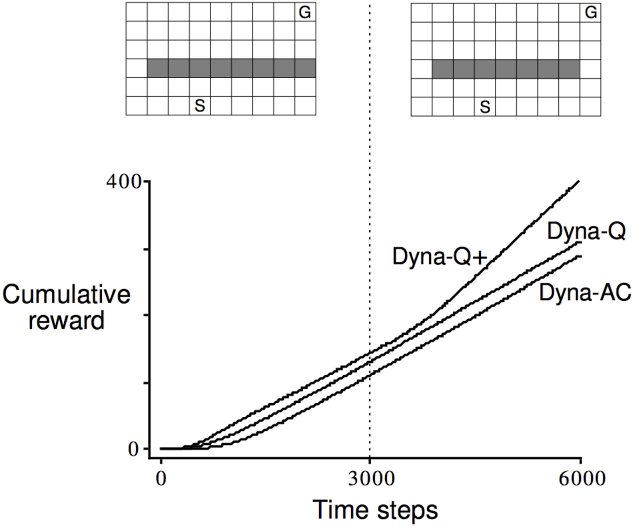
Dyna-Q with an Inaccurate Model

- The changed environment is harder



Dyna-Q with an Inaccurate Model (2)

- The changed environment is easier



Dyna with Function Approximation

- How can an agent plan when the actual environmental states are not known?
- Can directly approximate probability distributions of the transitions and the rewards.
- Probability distribution models in high dimensional feature spaces are computationally expensive and often inaccurate!

Simulation-Based Search

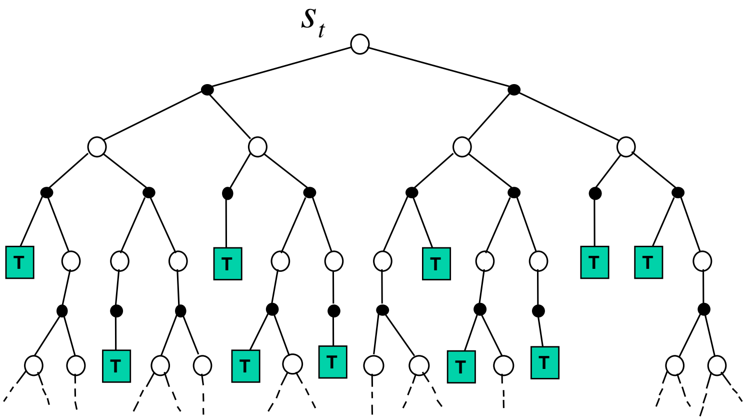
- We have been learning a model and planning with it.
- Now consider that setting where the model is given (fixed), and we want to use it.

Planning for Action Selection

- We considered the case where planning is used to improve a global value function
- Now consider planning for the near future, to select the next action
- The distribution of states that may be encountered from now can differ from the distribution of states encountered from a starting state
- The agent may be able to make a more accurate local value function (for the states that will be encountered soon) than the global value function

Forward Search

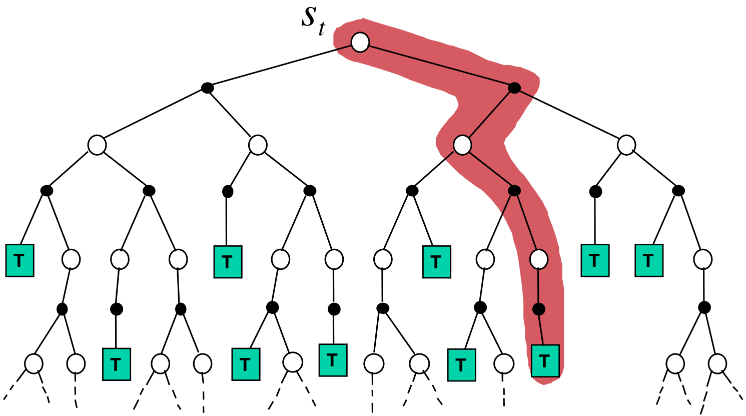
- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead



- 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



Simulation-Based Search (2)

- Simulate episodes of experience from now with the model

$$\{S_t^k, A_t^k, R_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \hat{p}_\eta$$

- Apply model-free RL to simulated episodes
  - Monte-Carlo control $\rightarrow$ Monte-Carlo search
  - Sarsa $\rightarrow$ TD search

Search tree vs. value function approximation

- Search tree is a table lookup approach
- Based on a partial instantiation of the table
- For model-free reinforcement learning, table lookup is naive
  - Can't store value for all states
  - Doesn't generalise between similar states
- For simulation-based search, table lookup is less naive
  - Search tree stores value for easily reachable states
  - But still doesn't generalise between similar states
  - In huge search spaces, value function approximation is helpful

Monte-Carlo Simulation

- Given a parameterized model $\mathcal{M}_\eta$ and a simulation policy π
- Simulate K episodes from current state S_t

$$\{S_t^k = S_t, A_t^k, R_{t+1}^k, S_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \hat{p}_\eta, \pi$$

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

$$v(S_t) = \frac{1}{K} \sum_{k=1}^{K} G_t^k \rightsquigarrow v_\pi(S_t)$$

Simple Monte-Carlo Search

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

$$\{S_t^k = s, A_t^k = a, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

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

$$q(s, a) = \frac{1}{K} \sum_{k=1}^{K} G_t^k \rightsquigarrow q_\pi(s, a)$$

- Select current (real) action with maximum value

$$A_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\; q(S_t, a)$$

Monte-Carlo Tree Search (Evaluation)

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

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

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

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

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

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q(S_t, a)$$

Monte-Carlo Tree Search (Simulation)

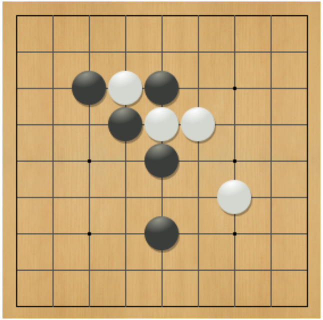
- In MCTS, the simulation policy π improves
- The simulation policy π has two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions from $q(s, a)$ (e.g. $\epsilon -$ greedy($q(s, a)$))
  - Rollout policy (fixed): e.g., pick actions randomly
- Repeat (for each simulated episode)
  - Select actions in tree according to tree policy.
  - Expand search tree by one node
  - Rollout to termination with default policy
  - Update action-values q(s,a) in the tree
- Output best action when simulation time runs out.
- With some assumptions, converges to the optimal values, $q(s, a) \rightarrow q_*(s, a)$

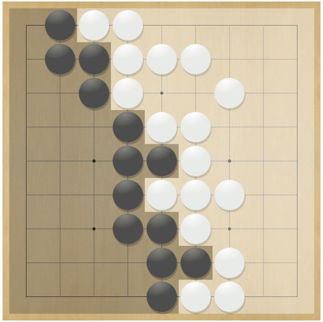
Case Study: the Game of Go

- 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
- Traditional game-tree search 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

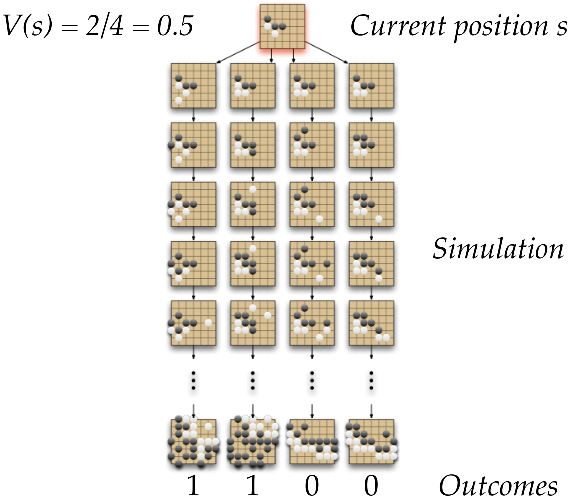
- How good is a position s?
- Reward function (undiscounted):

$$R_t = 0 \text{ for all non-terminal steps } t < T$$
$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

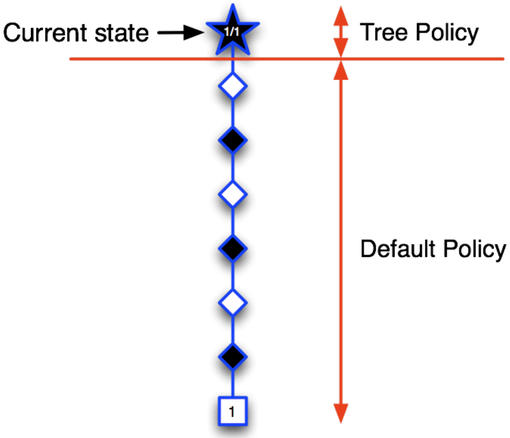
- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

$$v_\pi(s) = \mathbb{E}_\pi \left[R_T \mid s \right] = p(\text{Black wins} \mid s)$$
$$v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

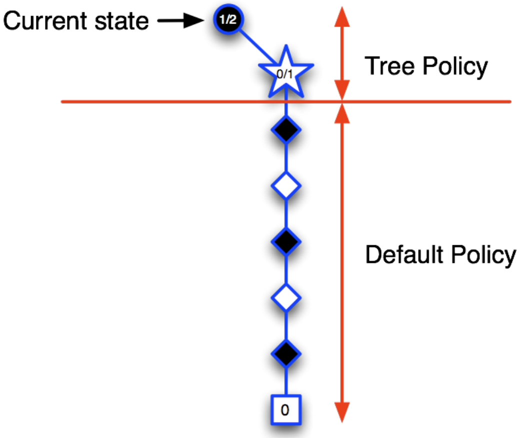
Monte-Carlo Evaluation in Go



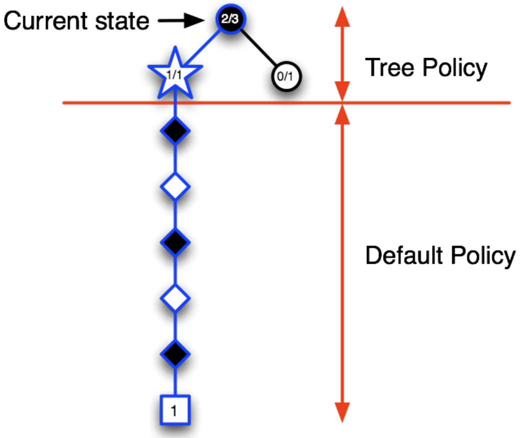
Applying Monte-Carlo Tree Search (1)



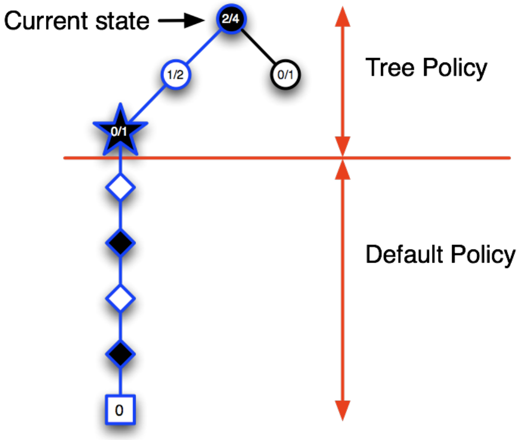
Applying Monte-Carlo Tree Search (2)



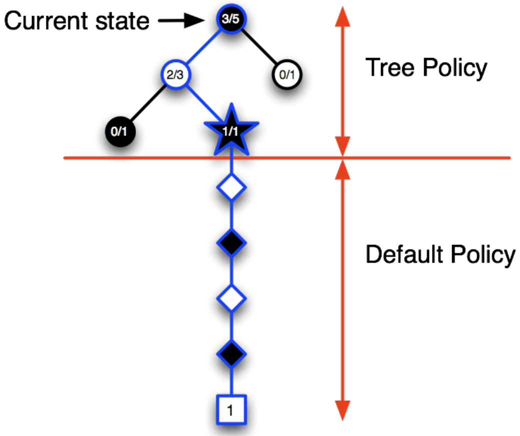
Applying Monte-Carlo Tree Search (3)



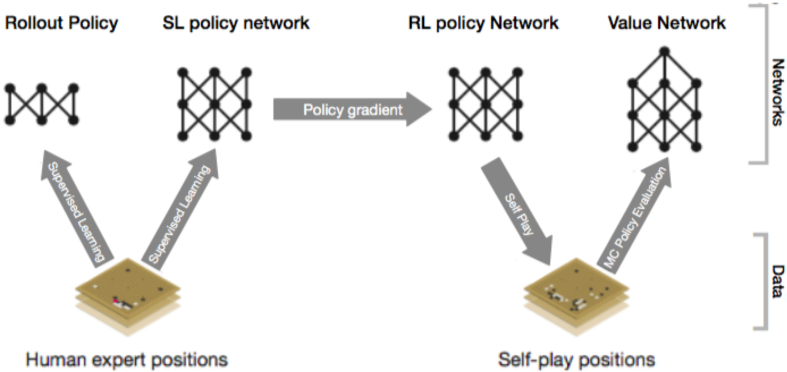
Applying Monte-Carlo Tree Search (4)



Applying Monte-Carlo Tree Search (5)



MCTS and AlphaGo



AlphaGo Zero

- AlphaGo combines several components
- But is that necessary?
- In AlphaGo Zero: no rollouts, no human data
- Results in an even stronger player

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

Summary

- Learning a model of the environment and planning with it
- Integrating planning and learning with Dyna
- Planning for the now with MCTS