Model-based Reinforcement Learning

Course: Reinforcement Learning, Bachelor AI, Leiden University

Lecturer: Thomas Moerland

Content

I: Integrated view of planning and learning

II: Combination of planning and learning

Content

I: Integrated view of planning and learning

1. Access to the MDP dynamics
2. Planning versus learning
3. Types of back-ups

II: Combination of planning and learning

Content

I: Integrated view of planning and learning

1. Access to the MDP dynamics
2. Planning versus learning
3. Types of back-ups

II: Combination of planning and learning

4. Model-based reinforcement learning (MBRL)
5. Learning a model
6. Model-based RL algorithms

1) Access to the MDP dynamics

Access to the MDP dynamics

Dynamics function: $p(s'|s,a)$

Which next state s can we observe after taking action a in state s

Access to the MDP dynamics

Dynamics function: p(s'|s,a)

Which next state s can we observe after taking action a in state s

Two important considerations:

1) Is our access *reversible* or *irreversible*?

Access to the environment dynamics

- Reversible access
  - We can take try any action in any state we want
  - Similar to planning in our head
  - Call such access a **model**

Access to the environment dynamics

- Reversible access
  - We can take try any action in any state we want
  - Similar to planning in our head
  - Call such access a **model**

- Irreversible access
  - When we take an action, we have to continue from the next state
  - Similar to the real world
  - Call such access an **environment**

Access to the MDP dynamics

Dynamics function: p(s'|s,a)

Which next state s can we observe after taking action a in state s

Two important considerations:

1) Is our access *reversible* or *irreversible*? (**model** versus **environment**)

Access to the MDP dynamics

Dynamics function: p(s'|s,a)

Which next state s can we observe after taking action a in state s

Two important considerations:

1) Is our access *reversible* or *irreversible*? (**model** versus **environment**)

2) Do we get the *full distribution* of p(s'|s,a), or a *sample*?

Access to the MDP dynamics

Dynamics function: p(s'|s,a)

Which next state s can we observe after taking action a in state s

Two important considerations:

1) Is our access *reversible* or *irreversible*? (**model** versus **environment**)

2) Do we get the *full distribution* of p(s'|s,a), or a *sample*?

p(s'=1) = 0.1
p(s'=2) = 0.2
p(s'=3) = 0.7

s' = 3! (this time)

Access to the MDP dynamics

Dynamics function: p(s'|s,a)

Which next state s can we observe after taking action a in state s

Two important considerations:

1) Is our access *reversible* or *irreversible*? (**model** versus **environment**)

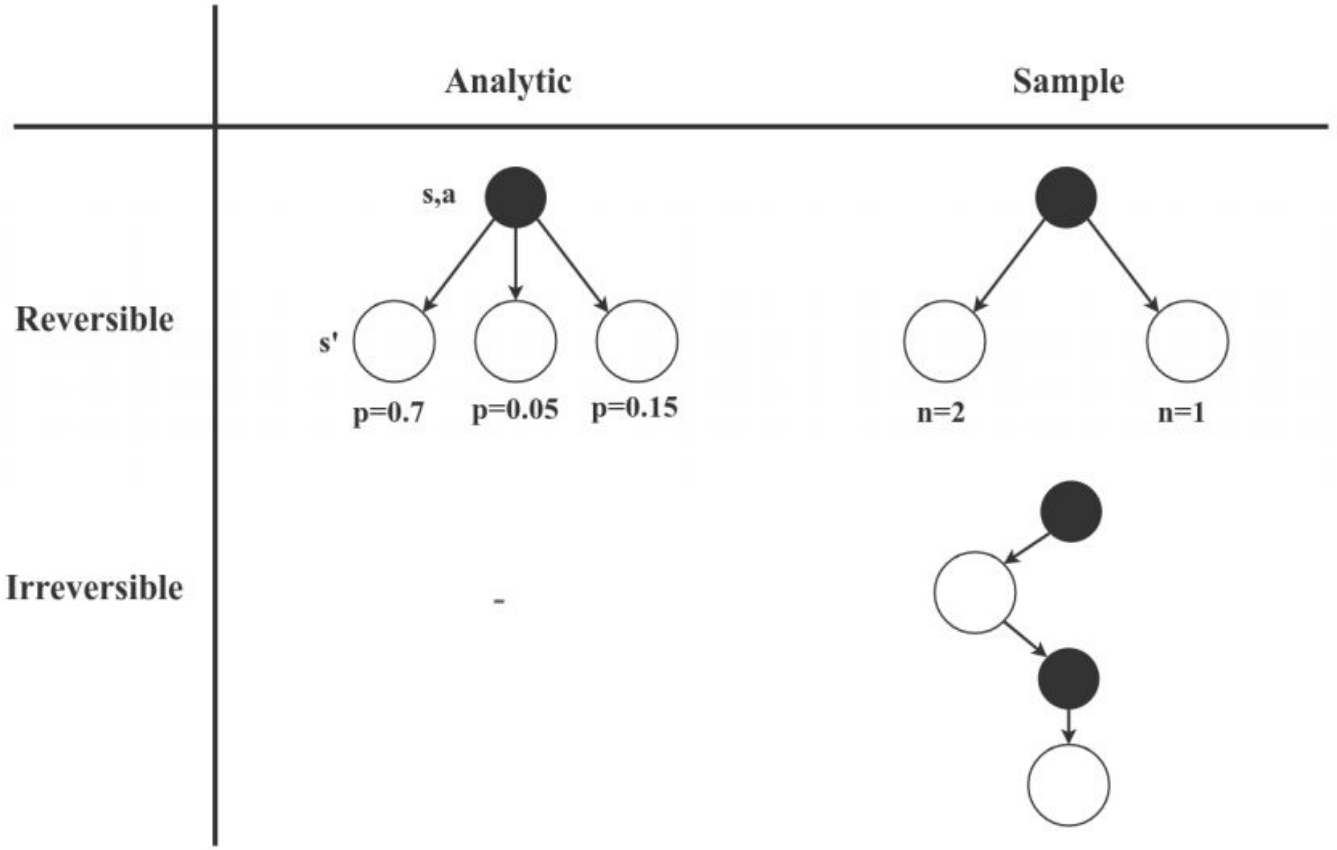
2) Do we get the *full distribution* of p(s'|s,a), or a *sample*?

p(s'=1) = 0.1
p(s'=2) = 0.2
p(s'=3) = 0.7

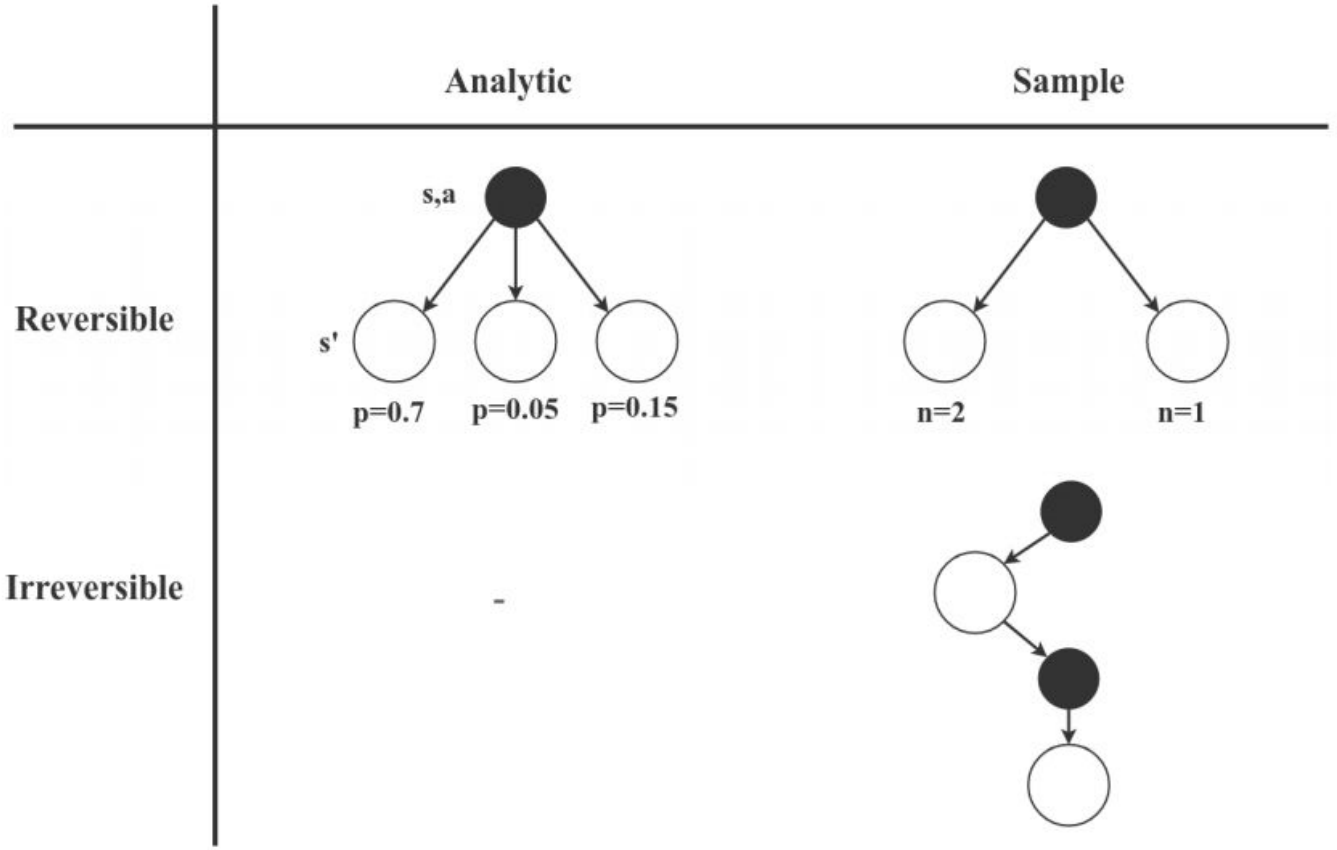
s' = 3! (this time)

Next page: 2-by-2 overview of these considerations

Access to the MDP dynamics

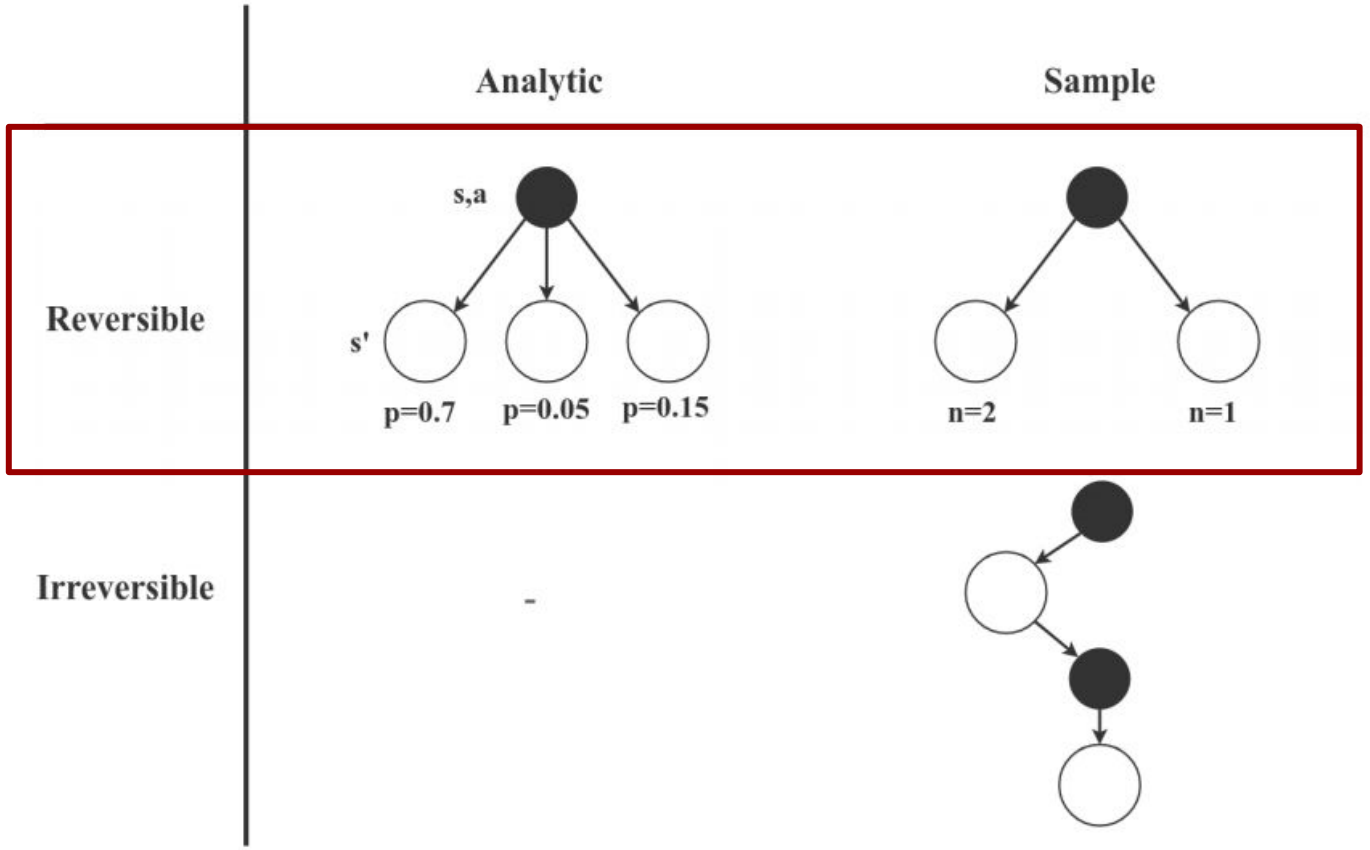


Access to the MDP dynamics



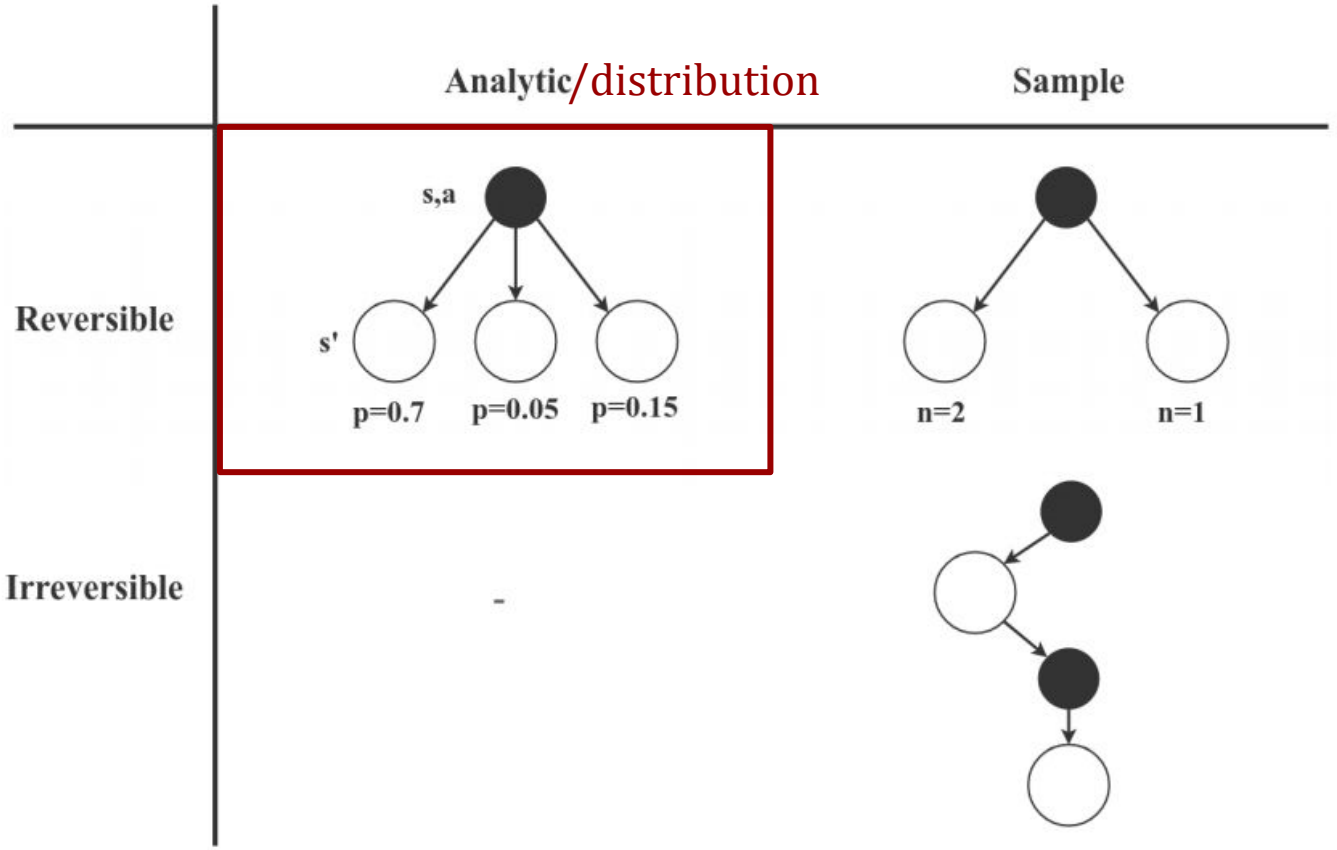
White nodes are states, black nodes are actions

Access to the MDP dynamics



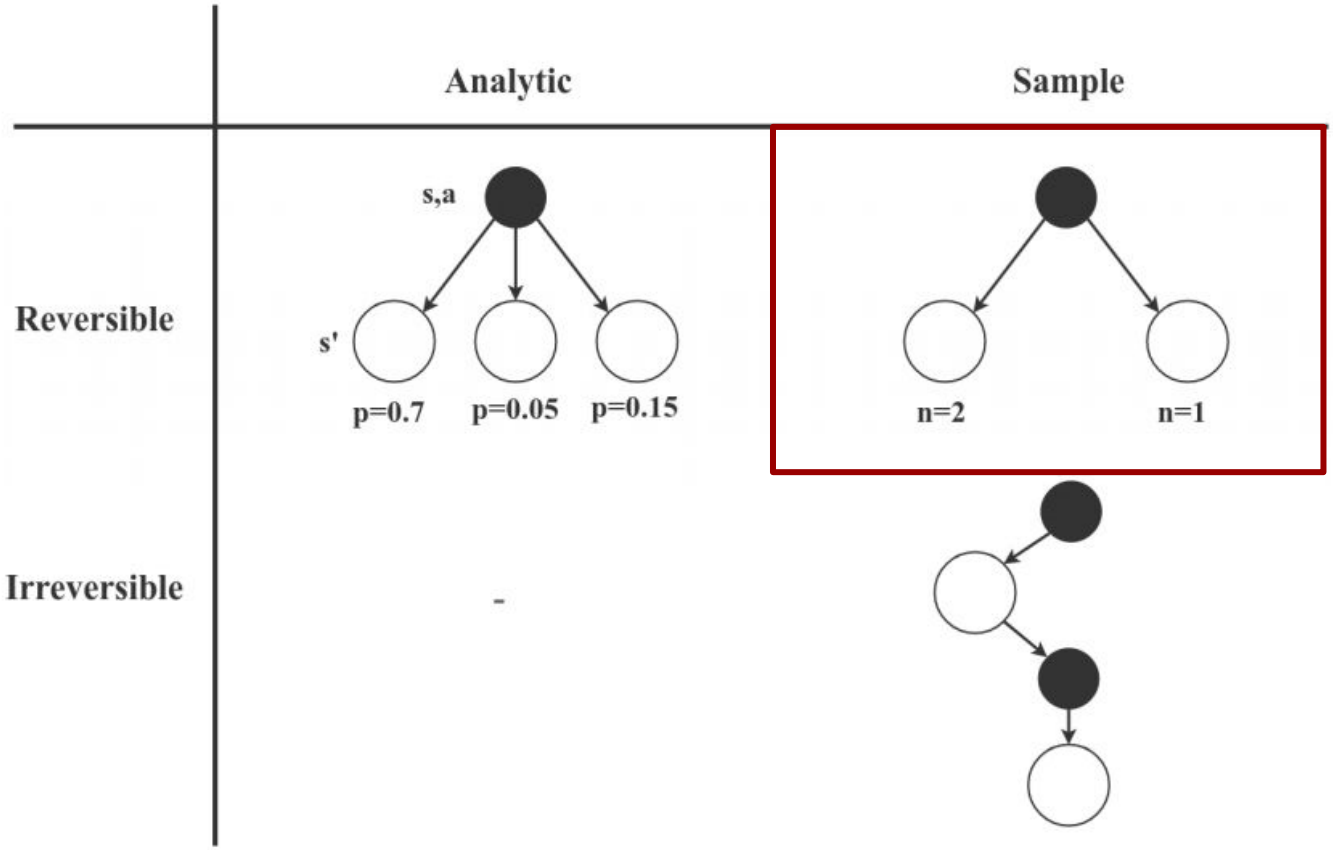
'Model'

Access to the MDP dynamics



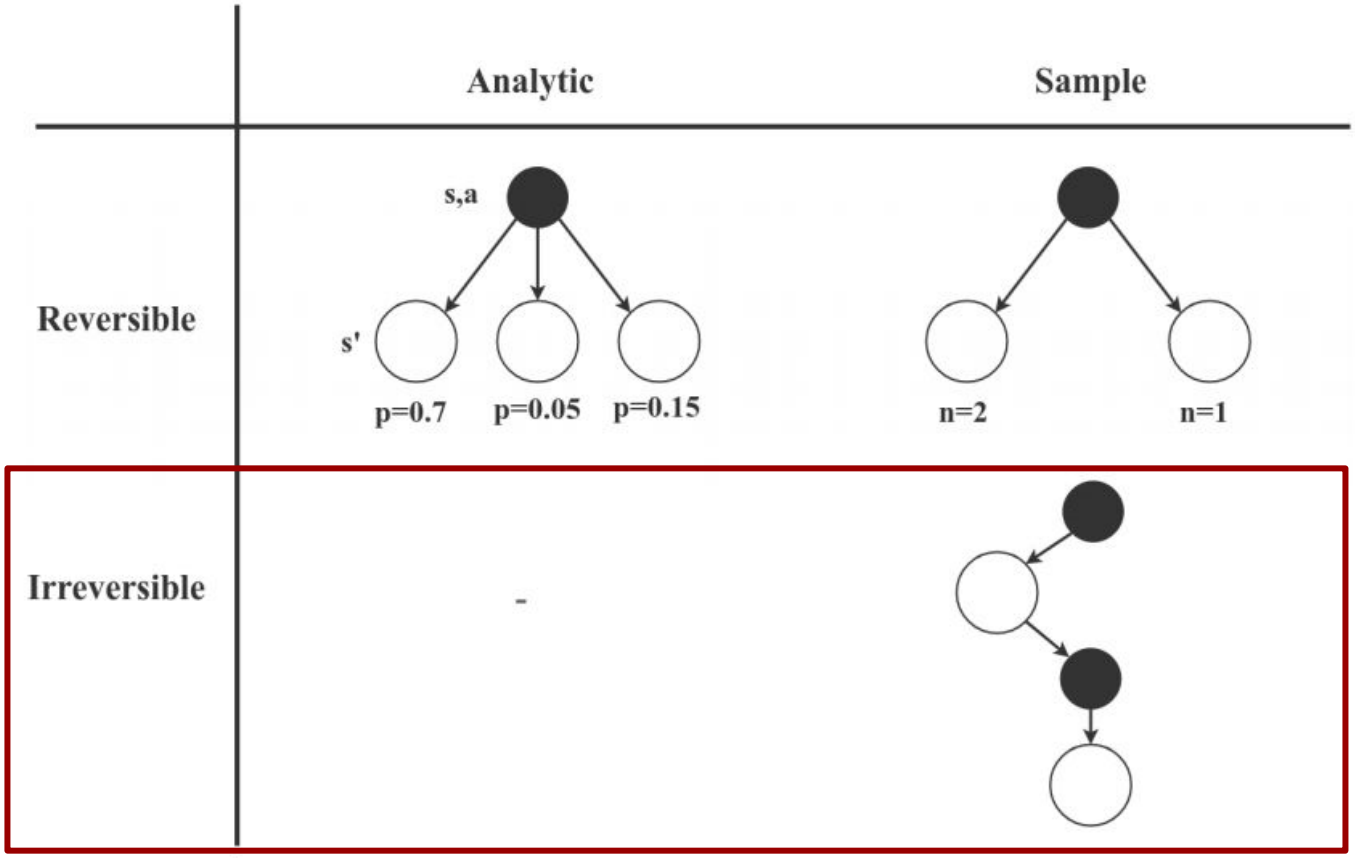
Exact probability of each next state

Access to the MDP dynamics



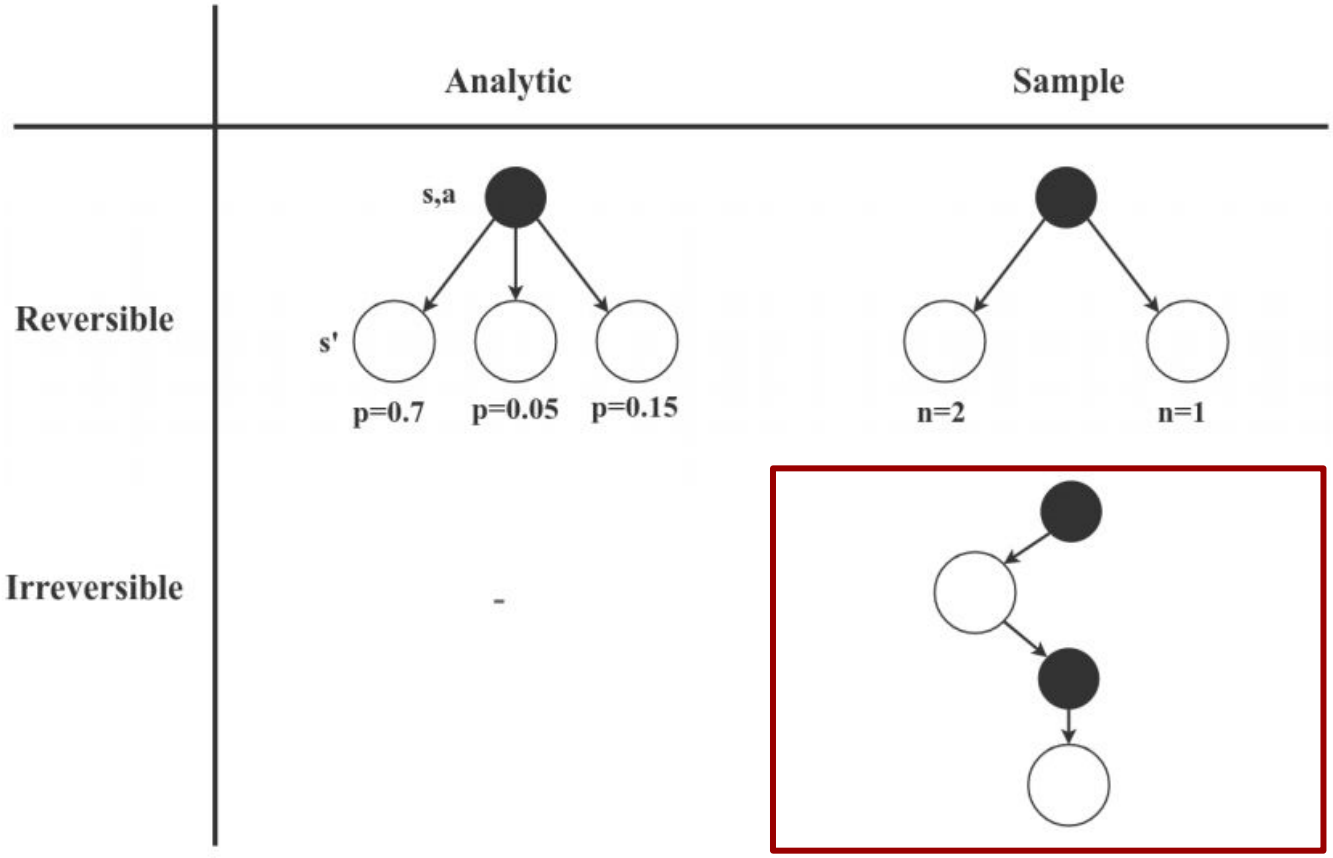
Only sample, but repeatedly

Access to the MDP dynamics



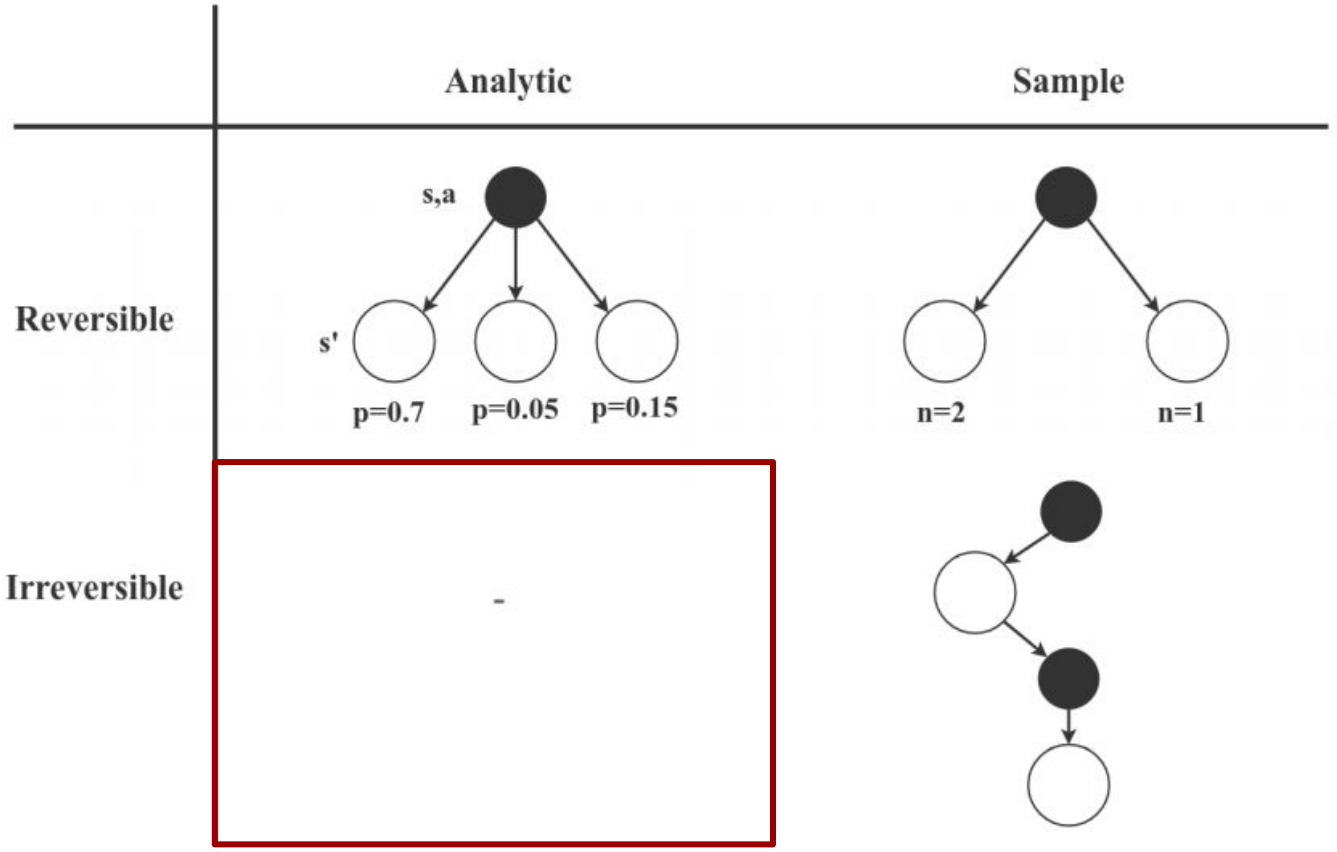
'Environment'

Access to the MDP dynamics



Only sample once, then continue

Access to the MDP dynamics



In practice does not occur

Summary 1: Access to MDP dynamics

1. Reversible versus irreversible access

2. Distributional versus sample models

2) Difference between planning and learning

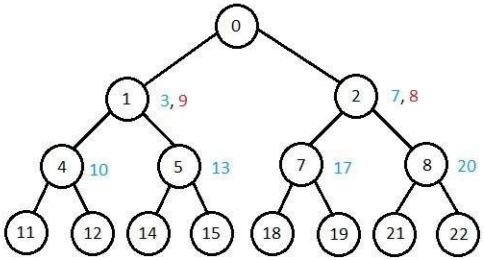
Planning versus learning

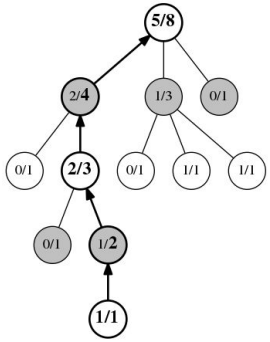
Planning

Reinforcement learning

Planning versus learning

Planning

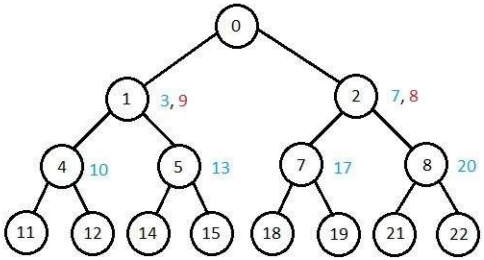


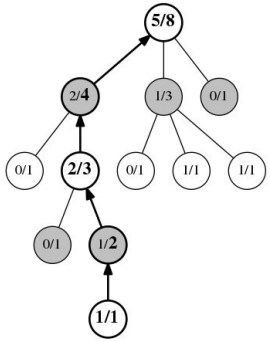


Reinforcement learning

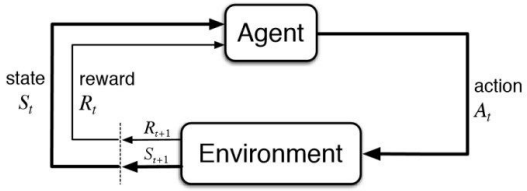
Planning versus learning

Planning





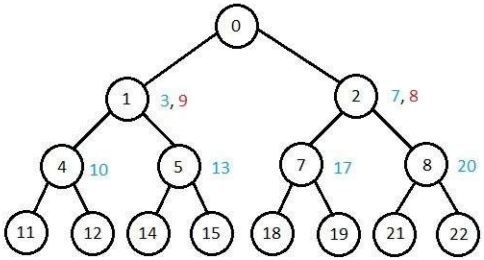
Reinforcement learning

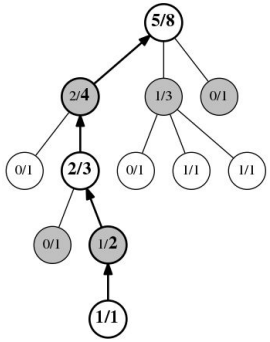


$$Q = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Planning versus learning

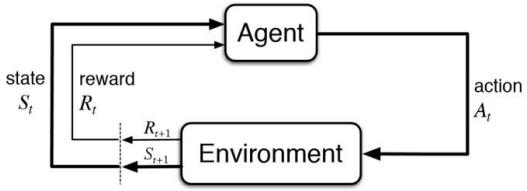
Planning





Both can solve the same MDP optimization problem

Reinforcement learning



Action

State	0	1	2	3	4	5
0	−1	−1	−1	−1	0	−1
1	−1	−1	−1	0	−1	100
2	−1	−1	−1	0	−1	−1
3	−1	0	0	−1	0	−1
4	0	−1	−1	0	−1	100
5	−1	0	−1	−1	0	100

Q =

Planning versus learning

So what discriminates planning from reinforcement learning?

Planning versus learning

So what discriminates planning from reinforcement learning?

Two factors:

1)	Access to the MDP dynamics:	reversible	or	irreversible
2)	Storage of the solution:	local	or	global

Planning versus learning

So what discriminates planning from reinforcement learning?

Two factors:

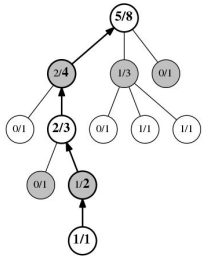
1)	Access to the MDP dynamics:	reversible	or	irreversible
2)	Storage of the solution:	local	or	global

Local versus global solution

Question: can you guess what the difference between a local and global solution might be?

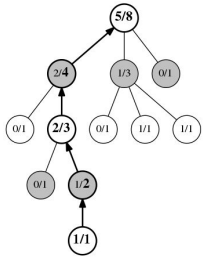
Local versus global solution

- A **local** solution temporarily stores a solution for a subset of all states
  - focus on current state
  - discarded after execution
  - e.g., a planning tree



Local versus global solution

- A **local** solution temporarily stores a solution for a subset of all states
  - focus on current state
  - discarded after execution
  - e.g., a planning tree



- A **global** solution permanently stores estimates for all states
  - e.g., a value table
  - only option in (model-free) RL, because we have to move forward and do not know when we get back to a state

$$Q = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \multicolumn{6}{c}{\text{Action}} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Planning versus learning

Problem: we have two possible distinctions

Planning versus learning

Problem: we have two possible distinctions

	Local solution	Global solution
Reversible MDP access		
Irreversible MDP access		

Planning versus learning

Problem: we have two possible distinctions

	Local solution	Global solution
Reversible MDP access	**Planning** e.g., MCTS	
Irreversible MDP access		

Planning versus learning

Problem: we have two possible distinctions

	Local solution	Global solution
Reversible MDP access	**Planning** e.g., MCTS	
Irreversible MDP access		**Model-free RL** e.g., Q-learning

Planning versus learning

Problem: we have two possible distinctions

	Local solution	Global solution
Reversible MDP access	**Planning** e.g., MCTS	**Borderline/Model-based RL** e.g., Dynamic Programming
Irreversible MDP access		**Model-free RL** e.g., Q-learning

Planning versus learning

Problem: we have two possible distinctions

	Local solution	Global solution
Reversible MDP access	**Planning** e.g., MCTS	**Borderline/Model-based RL** e.g., Dynamic Programming
Irreversible MDP access	(impossible)	**Model-free RL** e.g., Q-learning

Q: Why is it impossible to use a local solution when we have irreversible access?

Planning versus learning

Problem: we have two possible distinctions

	Local solution	Global solution
Reversible MDP access	**Planning** e.g., MCTS	**Borderline/Model-based RL** e.g., Dynamic Programming
Irreversible MDP access	(impossible)	**Model-free RL** e.g., Q-learning

Q: Why is it impossible to use a local solution when we have irreversible access?
A: Local solutions get discarded after execution of the real action, but if the environment is irreversible, we directly throw away our new solution after the first sample.

Summary 2: Planning versus learning

1.	Planning:	reversible model	+	local solution
2.	Model-free RL:	irreversible model	+	global solution
3.	Borderline/model-based RL:	reversible model	+	global solution

3) Comparison of back-ups

Expected versus sample back-ups

Our access to the MDP dynamics also influences the way we can back-up information!

Expected versus sample back-ups

Our access to the MDP dynamics also influences the way we can back-up information!

Need to distinguish:

- **Expected** back-ups (mostly planning)
- **Sample** back-ups (mostly RL)

Expected versus sample back-ups

Our access to the MDP dynamics also influences the way we can back-up information!

Need to distinguish:

- **Expected** back-ups (mostly planning)
- **Sample** back-ups (mostly RL)

We will illustrate this difference with back-up diagrams

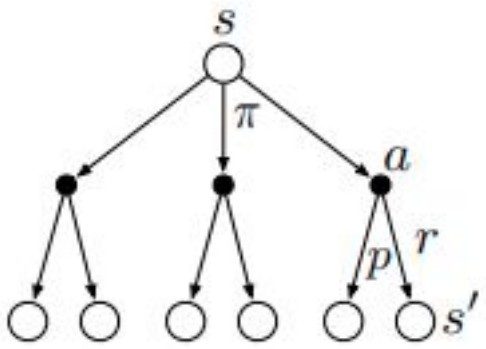
Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Bellman equation



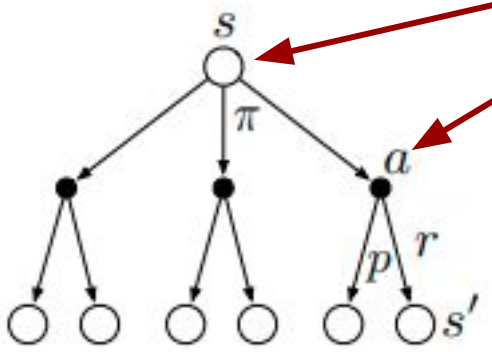
policy evaluation

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Bellman equation

States are circles, actions are dots

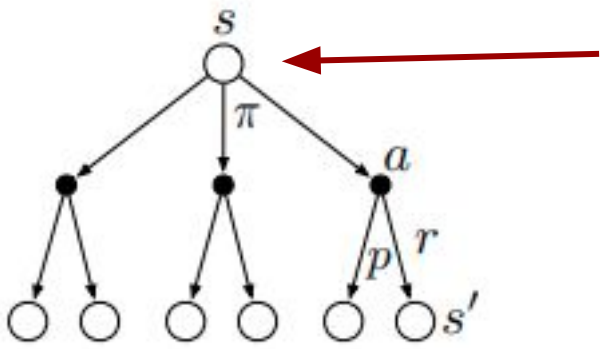


policy evaluation

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Bellman equation



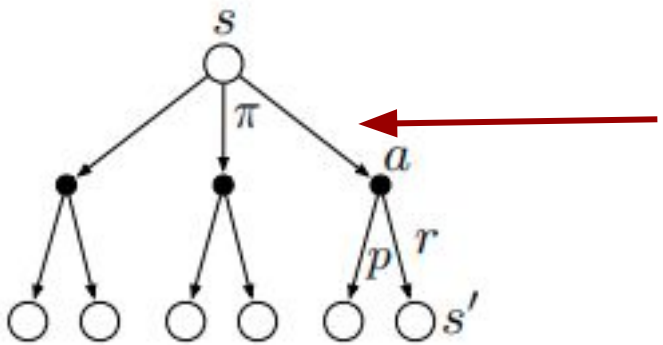
policy evaluation

To compute a new estimate of V(s)

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Bellman equation



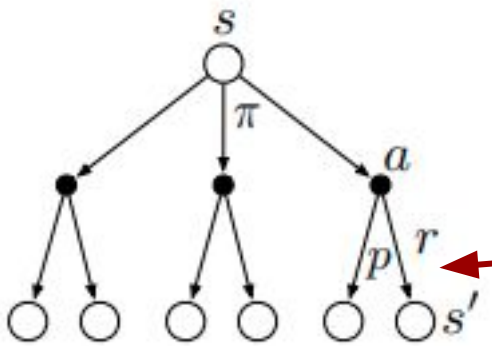
policy evaluation

We sum over all actions

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Bellman equation



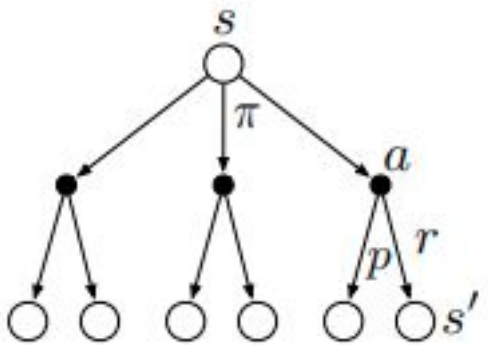
policy evaluation

For each action we sum over all possible next states

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Bellman equation



policy evaluation

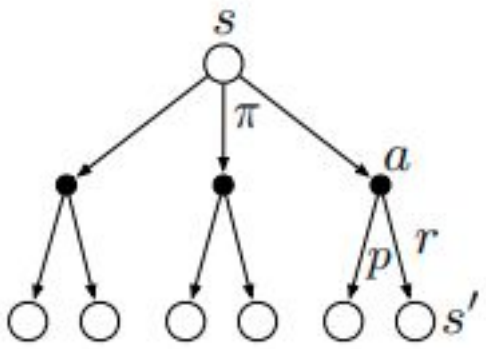
And for each possibility back-up the reward plus gamma times the next value V(s')

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Bellman equation

= **full expectation** over the actions (policy) and next states (dynamics)

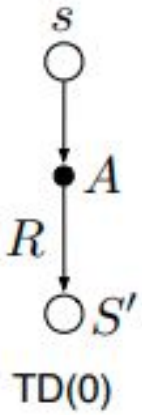


policy evaluation

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Temporal Difference Learning

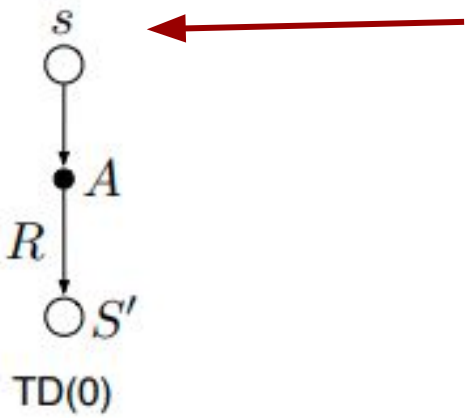


TD(0)

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Temporal Difference Learning

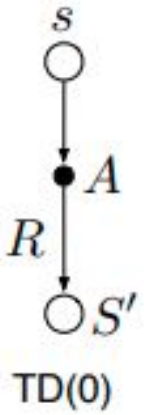


To compute a new estimate of V(s)

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Temporal Difference Learning



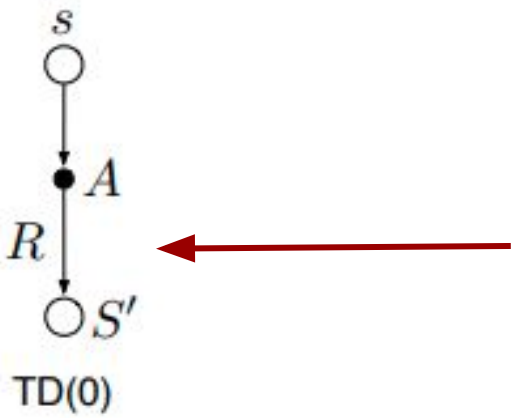
TD(0)

We sample a single action

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Temporal Difference Learning

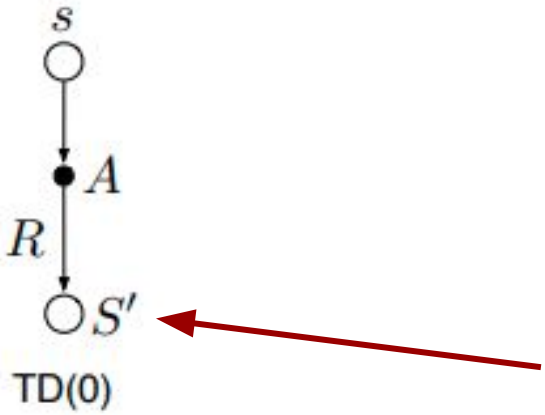


For this action sample a next state s'

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Temporal Difference Learning



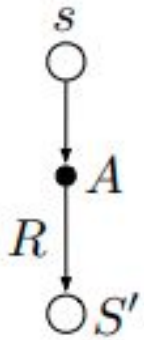
And back-up r + γ * V(s')

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Temporal Difference Learning

This is a **sample** back-up over the actions (policy) and next states (dynamics)



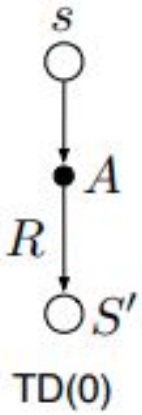
TD(0)

Back-up diagrams

A back-up diagram graphically represents a one-step back-up equation

Back-up diagram for the Temporal Difference Learning

This is a **sample** back-up over the actions (policy) and next states (dynamics)

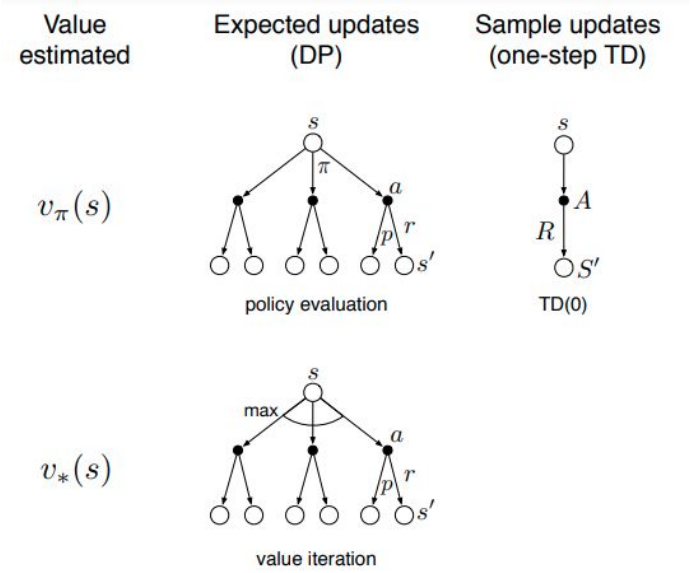


TD(0)

Sample estimates will converge to the correct equation over multiple updates/samples

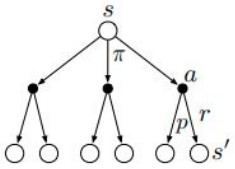
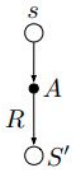
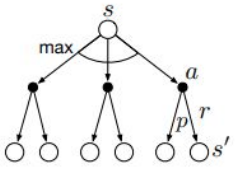
Spectrum of possible 1-step back-ups

Spectrum of possible 1-step back-ups



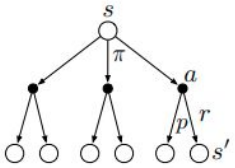
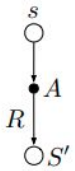
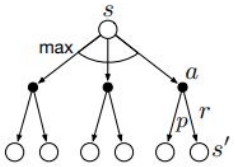
State values V(s)

Spectrum of possible 1-step back-ups

	Expected	**Sample**
Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$v_\pi(s)$	 policy evaluation	 TD(0)
$v_*(s)$	 value iteration	

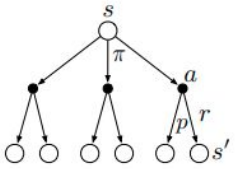
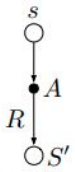
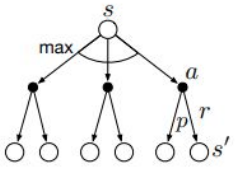
State values V(s)

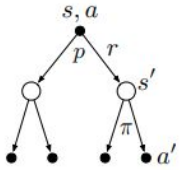
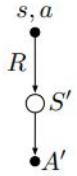
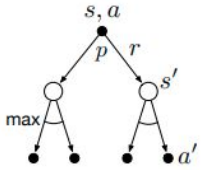
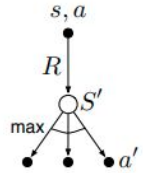
Spectrum of possible 1-step back-ups

	Expected	Sample
	Value estimated	Expected updates (DP)	Sample updates (one-step TD)
On-policy $v_\pi(s)$	 policy evaluation	 TD(0)
Off-policy $v_*(s)$	 value iteration	

State values V(s)

Spectrum of possible 1-step back-ups

	Value estimated	Expected: Expected updates (DP)	Sample: Sample updates (one-step TD)
On-policy	$v_\pi(s)$	 policy evaluation	 TD(0)
Off-policy	$v_*(s)$	 value iteration	

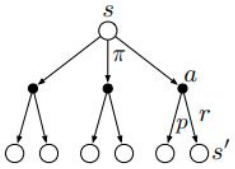
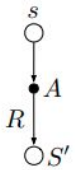
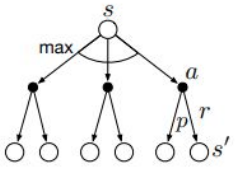
	Value estimated	Expected: Expected updates (DP)	Sample: Sample updates (one-step TD)
On-policy	$q_\pi(s,a)$	 q-policy evaluation	 Sarsa
Off-policy	$q_*(s,a)$	 q-value iteration	 Q-learning

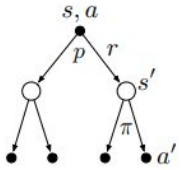
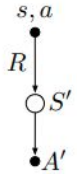
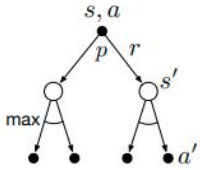
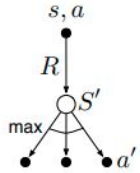
State values V(s)

State-action values Q(s,a)

Q: Categorize Q-learning.

Spectrum of possible 1-step back-ups

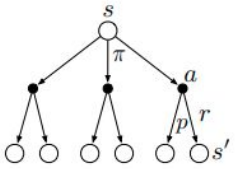
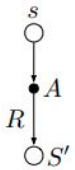
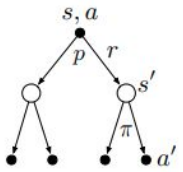
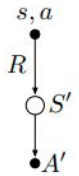
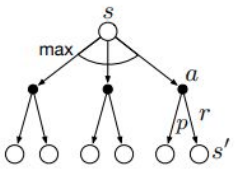
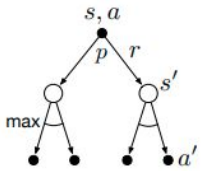
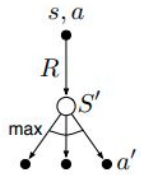
	Value estimated	Expected updates (DP)	Sample updates (one-step TD)
On-policy	$v_\pi(s)$	 policy evaluation	 TD(0)
Off-policy	$v_*(s)$	 value iteration	

	Value estimated	Expected updates (DP)	Sample updates (one-step TD)
On-policy	$q_\pi(s,a)$	 q-policy evaluation	 Sarsa
Off-policy	$q_*(s,a)$	 q-value iteration	 Q-learning

State values V(s)

State-action values Q(s,a)

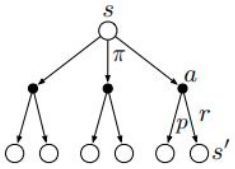
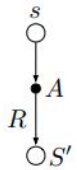
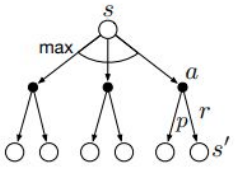
Spectrum of possible 1-step back-ups

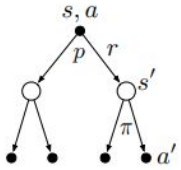
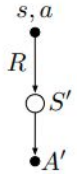
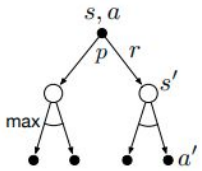
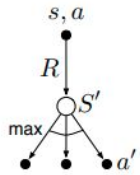
	Expected	Sample	Expected	**Sample**
	Value estimated	Expected updates (DP)	Sample updates (one-step TD)	Value estimated	Expected updates (DP)	Sample updates (one-step TD)
On-policy	$v_\pi(s)$	 policy evaluation	 TD(0)	$q_\pi(s,a)$	 q-policy evaluation	 Sarsa
Off-policy	$v_*(s)$	 value iteration		$q_*(s,a)$	 q-value iteration	 Q-learning

State values V(s)

State-action values Q(s,a)

Spectrum of possible 1-step back-ups

	Value estimated	Expected: Expected updates (DP)	Sample: Sample updates (one-step TD)
On-policy	$v_\pi(s)$	 policy evaluation	 TD(0)
Off-policy	$v_*(s)$	 value iteration	

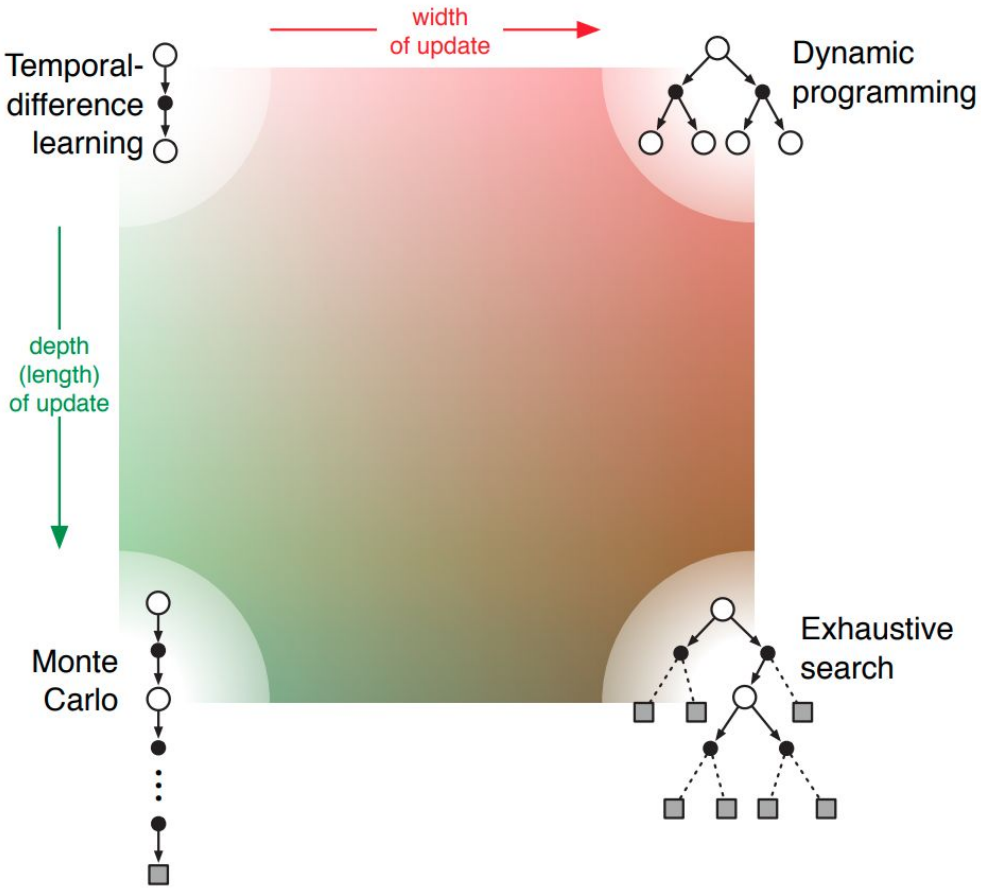
	Value estimated	Expected: Expected updates (DP)	**Sample**: Sample updates (one-step TD)
On-policy	$q_\pi(s,a)$	 q-policy evaluation	 Sarsa
Off-policy	$q_*(s,a)$	 q-value iteration	 Q-learning

Advice: carefully study these 1-step back-up diagrams, they provide a lot of insight/overview

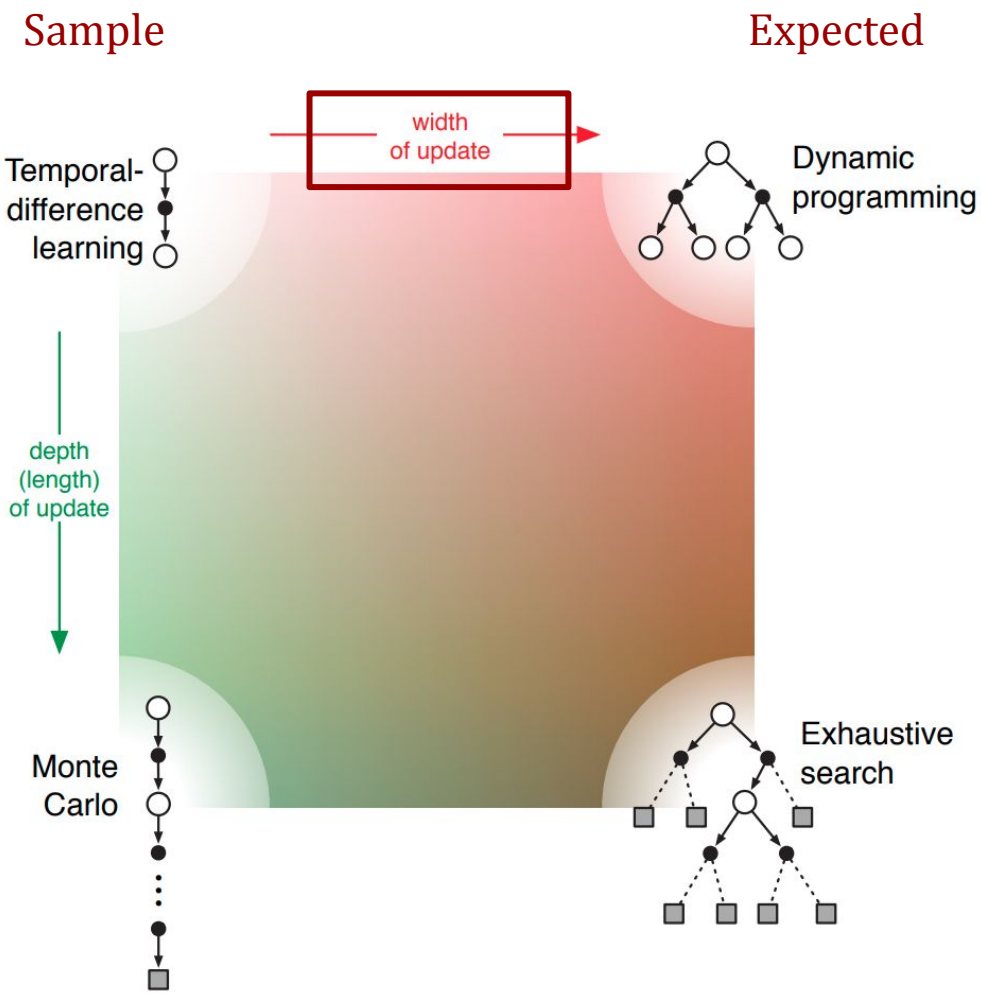
Multi-step back-up diagrams

Can also extend the back-up over multiple-steps

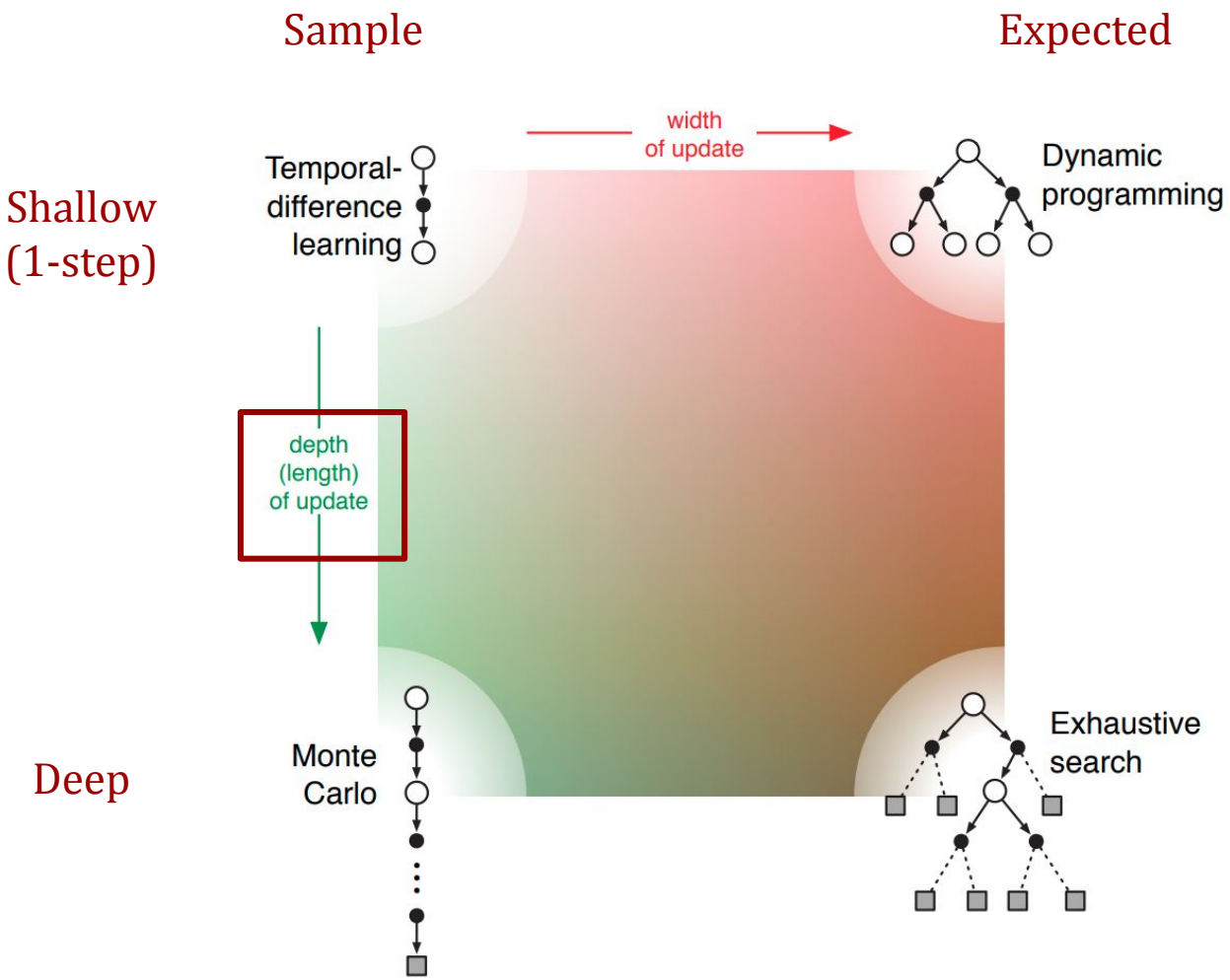
Multi-step back-up diagrams



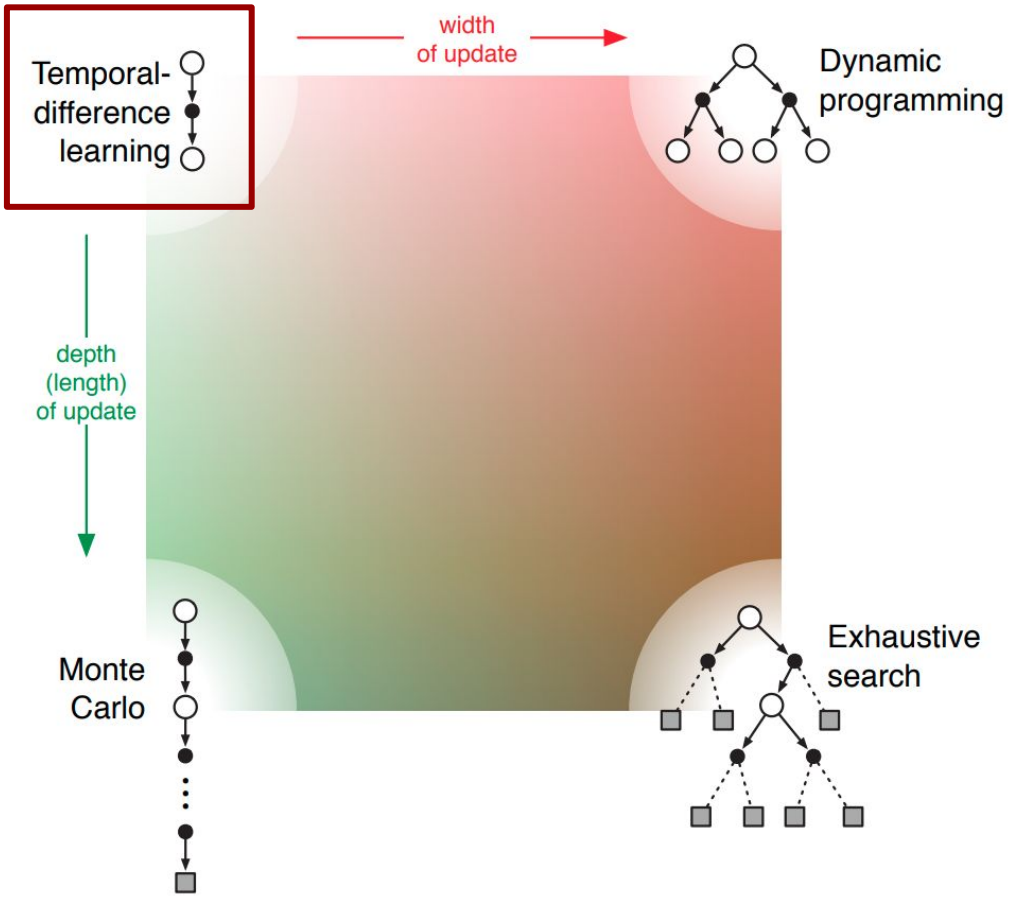
Multi-step back-up diagrams



Multi-step back-up diagrams

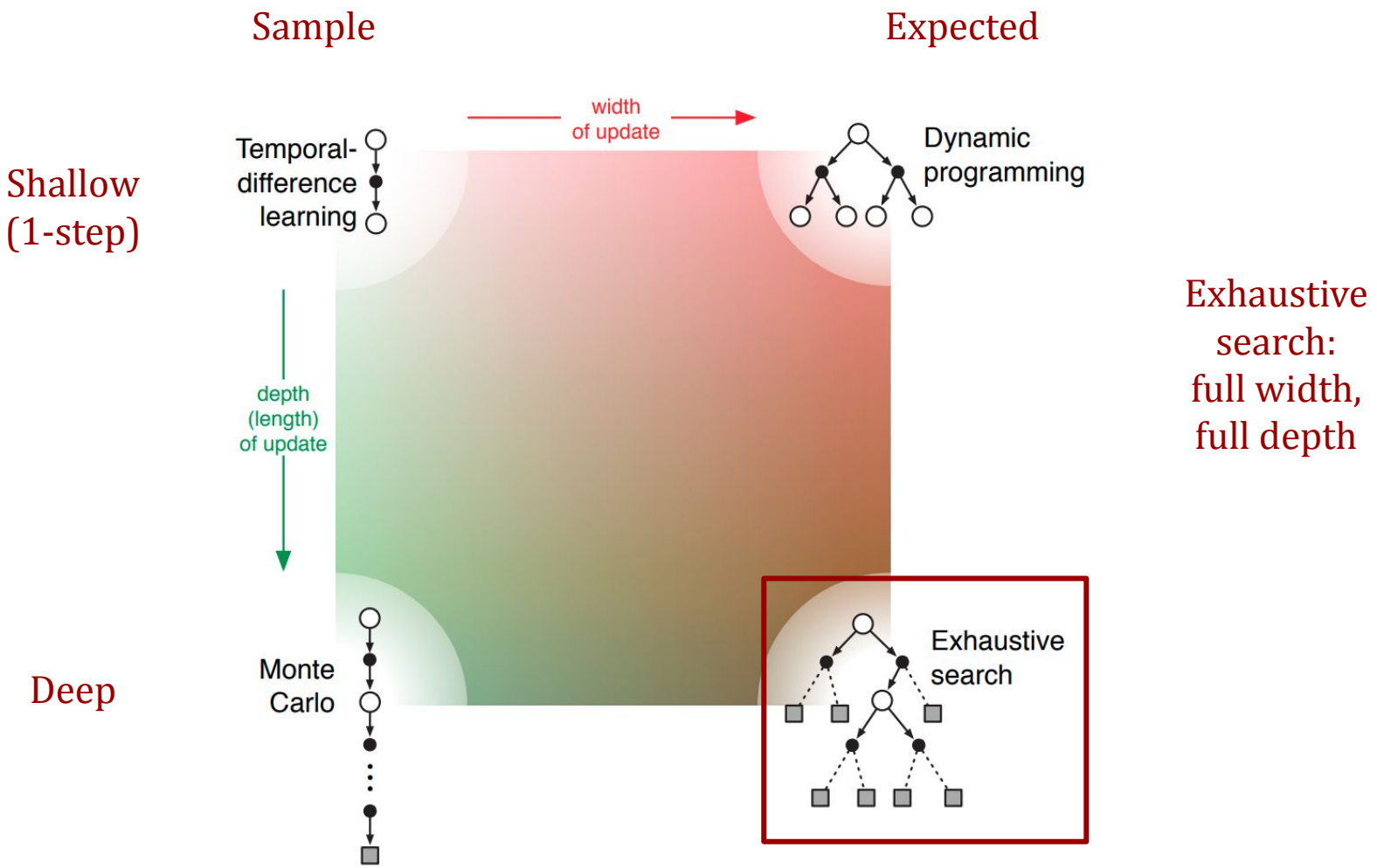


Multi-step back-up diagrams

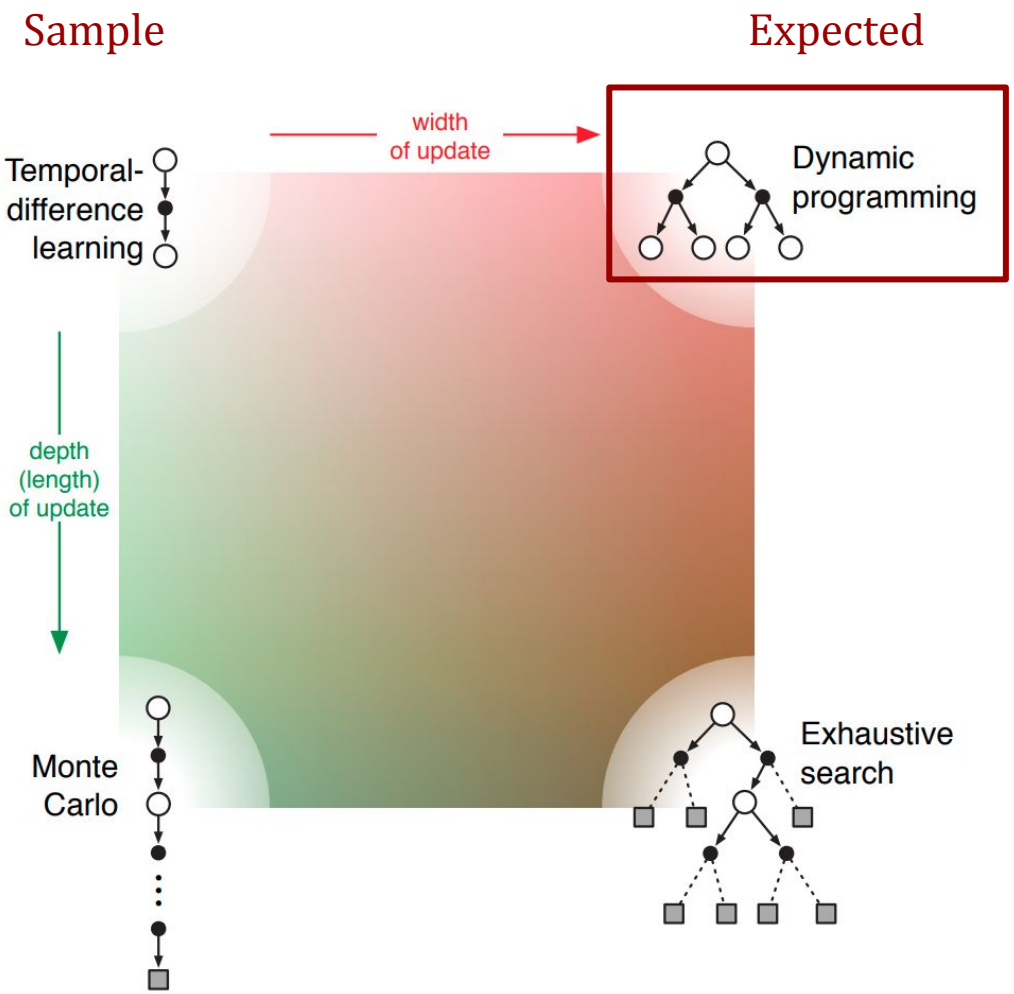


TD learning:
small width,
small depth

Multi-step back-up diagrams



Multi-step back-up diagrams



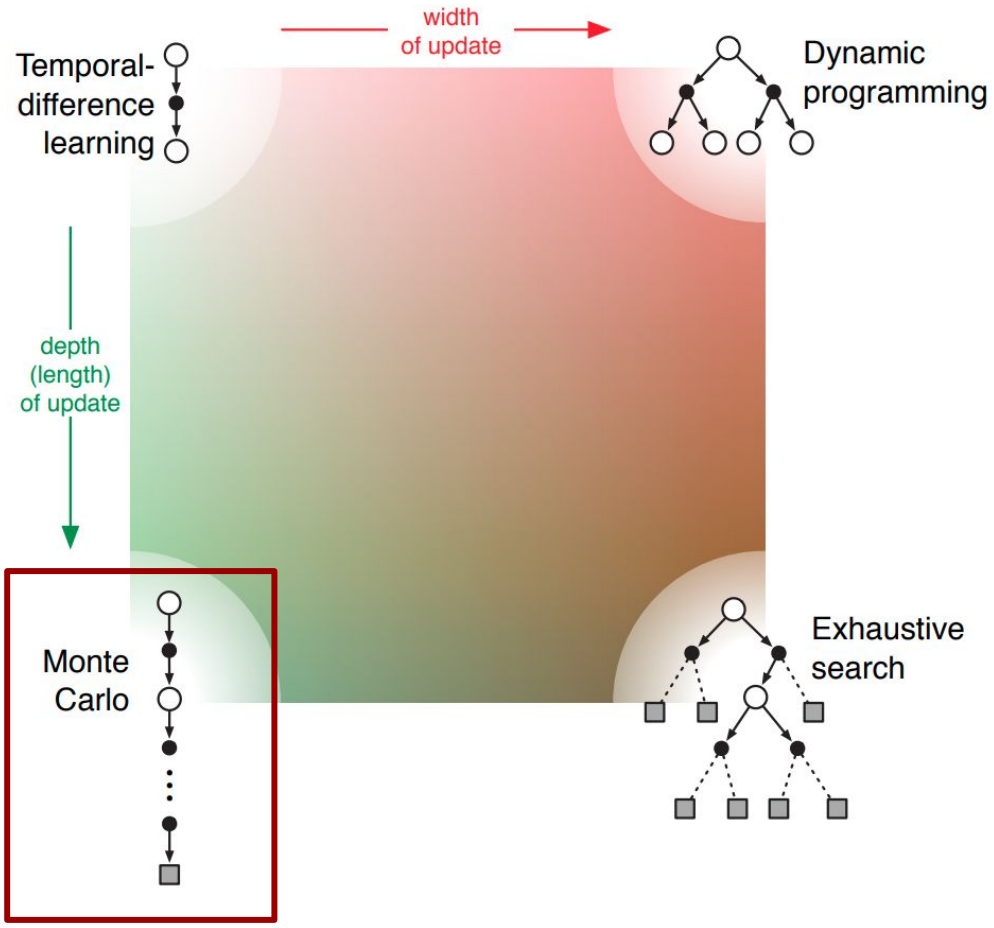
DP:
full width,
small depth

Multi-step back-up diagrams

Sample

Expected

Shallow (1-step)



Deep

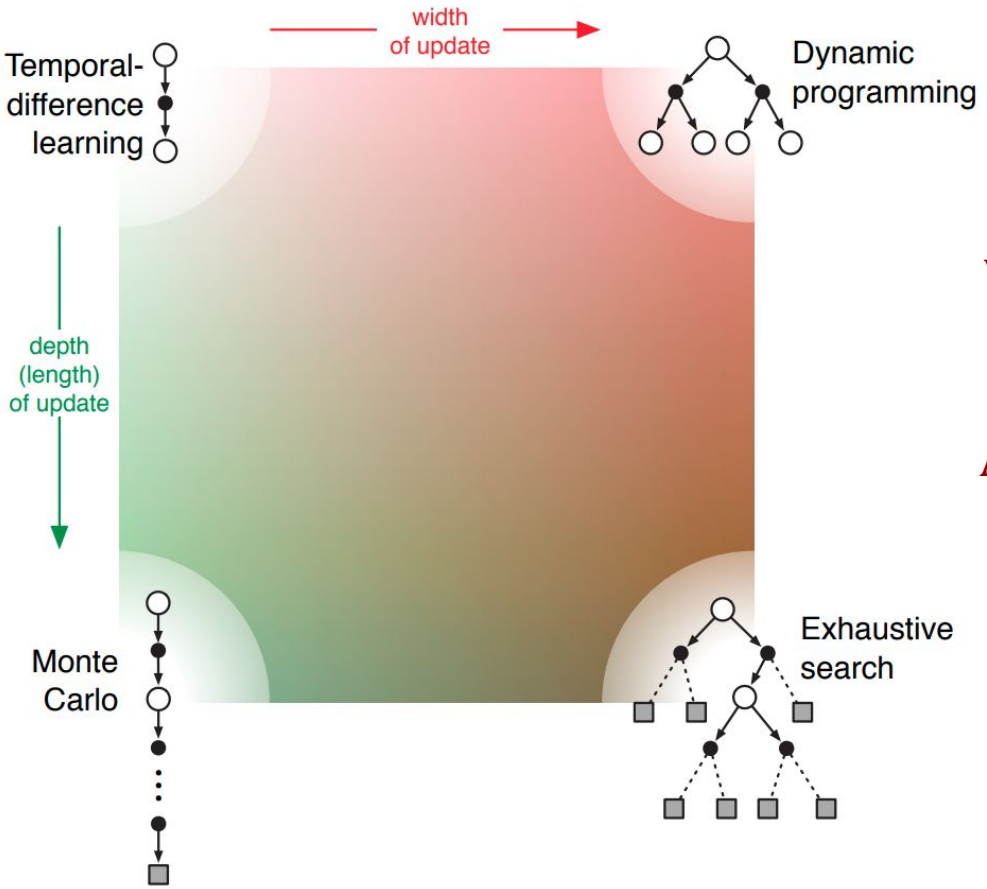
MC RL: small width, full depth

Multi-step back-up diagrams

Sample

Expected

Shallow (1-step)



Whole spectrum of width/depth back-up combinations

Again, carefully study, much insight!

Summary 3: Types of back-ups

Back-up diagrams can provide much intuition about the space of possible back-ups

Summary 3: Types of back-ups

Back-up diagrams can provide much intuition about the space of possible back-ups

Essentially four considerations:

1) The width over the actions: sample (shallow) or expected (broad)

2) The back-up policy: on-policy or off-policy

3) The width over the dynamics: sample (shallow) or expected (broad)

4) The depth of the back-up: 1-step (shallow) or full depth (deep)

Summary 3: Types of back-ups

Back-up diagrams can provide much intuition about the space of possible back-ups

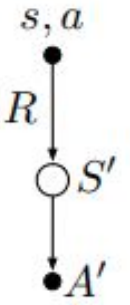
Essentially four considerations:

1) The width over the actions: sample (shallow) or expected (broad)

2) The back-up policy: on-policy or off-policy

3) The width over the dynamics: sample (shallow) or expected (broad)

4) The depth of the back-up: 1-step (shallow) or full depth (deep)



Sarsa

Question: can you indicate each decision for plain SARSA?

Summary 3: Types of back-ups

Back-up diagrams can provide much intuition about the space of possible back-ups

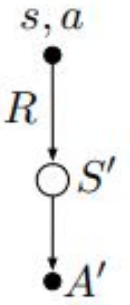
Essentially four considerations:

1) The width over the actions: **sample** (shallow) or expected (broad)

2) The back-up policy: **on-policy** or off-policy

3) The width over the dynamics: **sample** (shallow) or expected (broad)

4) The depth of the back-up: **1-step** (shallow) or full depth (deep)



Sarsa

Question: can you indicate each decision for plain SARSA?

Break

4) Model-based reinforcement learning

Model-based Reinforcement Learning

Interaction with the real world is typically irreversible.

Q: How could we still obtain a model of how the world works?

Model-based Reinforcement Learning

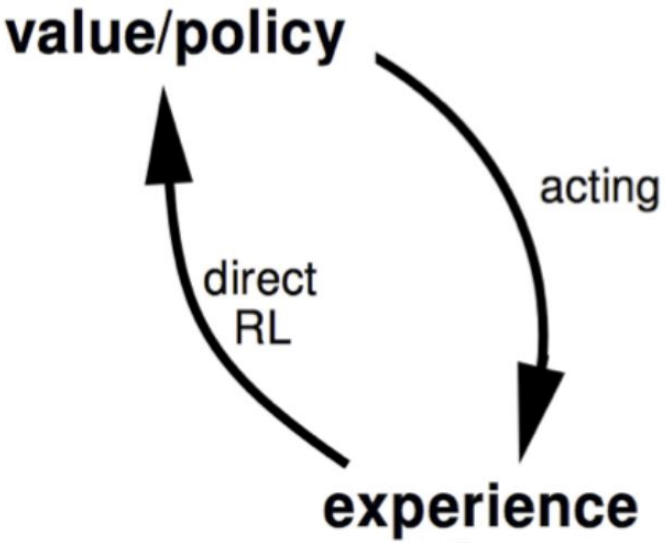
When the environment is irreversible, maybe we can still

learn a reversible model from data

Inspired by the way humans acquire (reversible) dynamics models:

we learn them from real-world (irreversible) experience

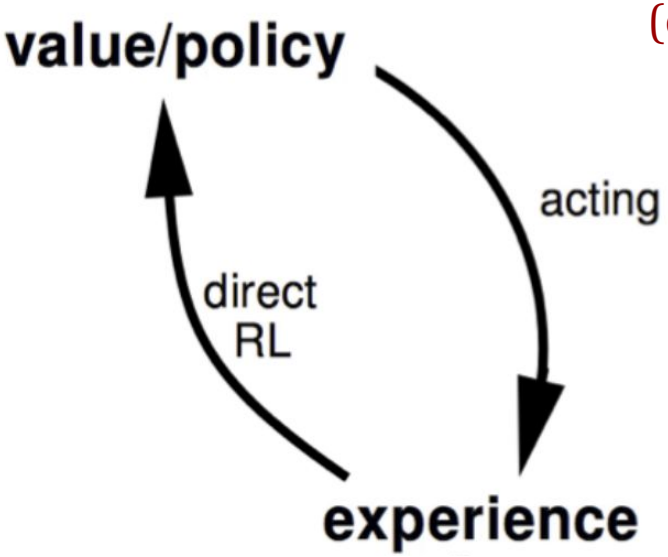
Model-based Reinforcement Learning



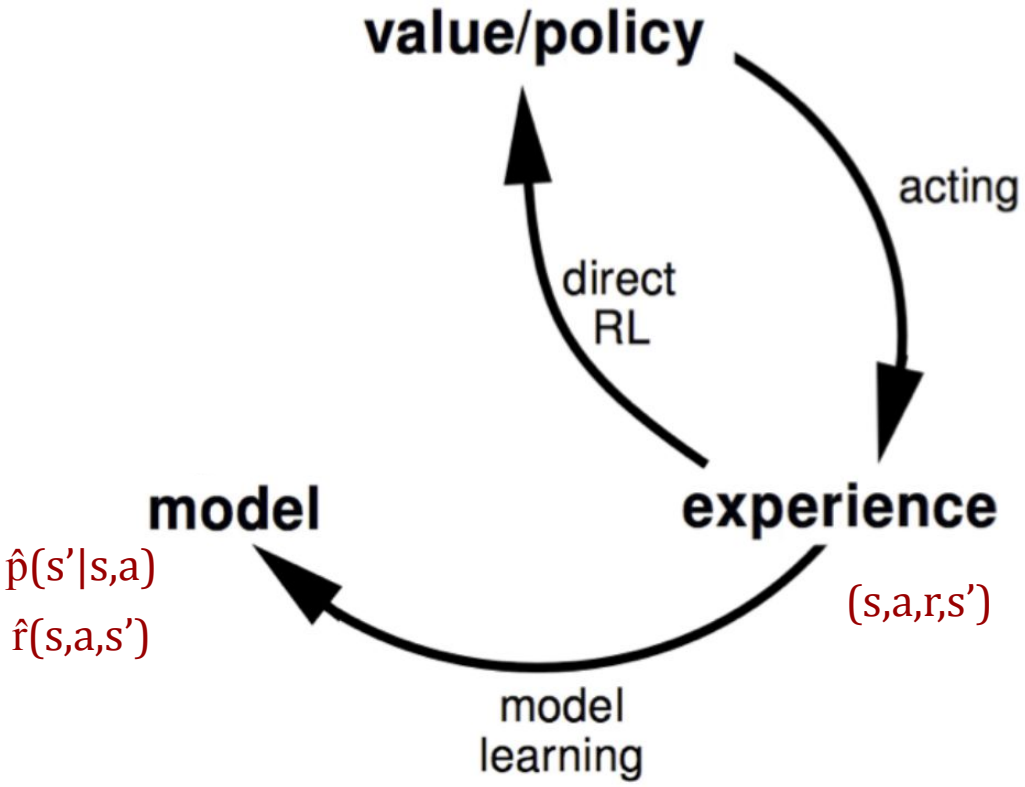
Model-based Reinforcement Learning

Model-free RL

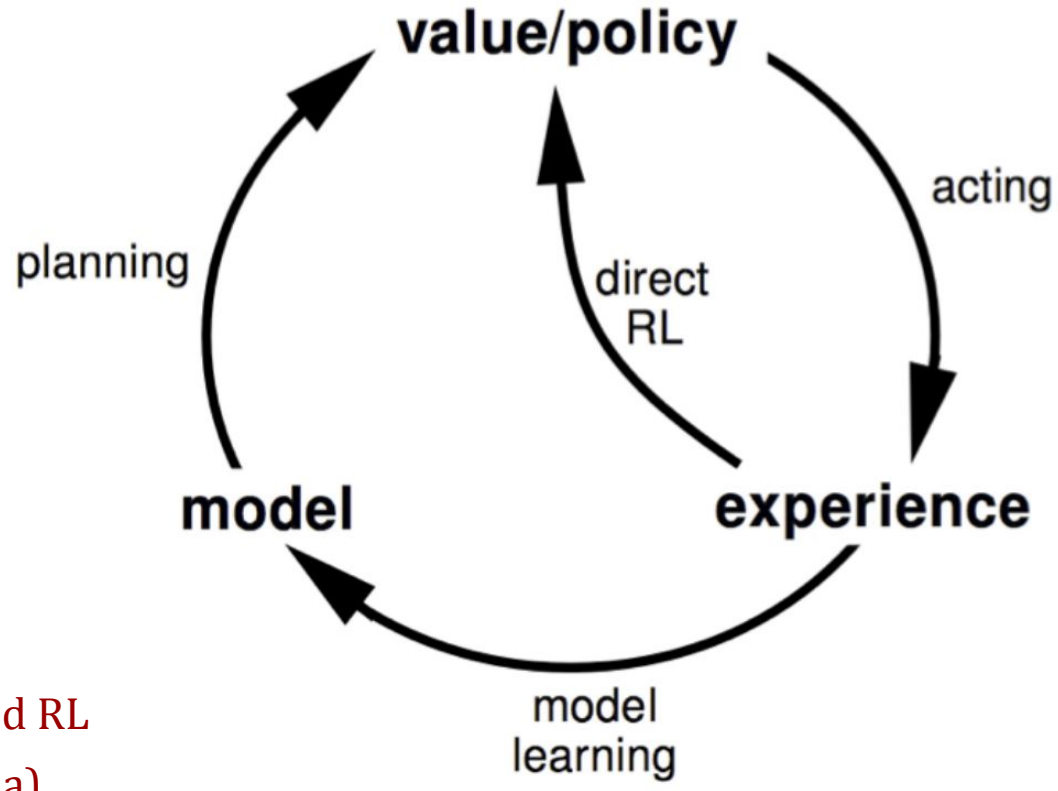
(e.g., Q-learning)



Model-based Reinforcement Learning



Model-based Reinforcement Learning



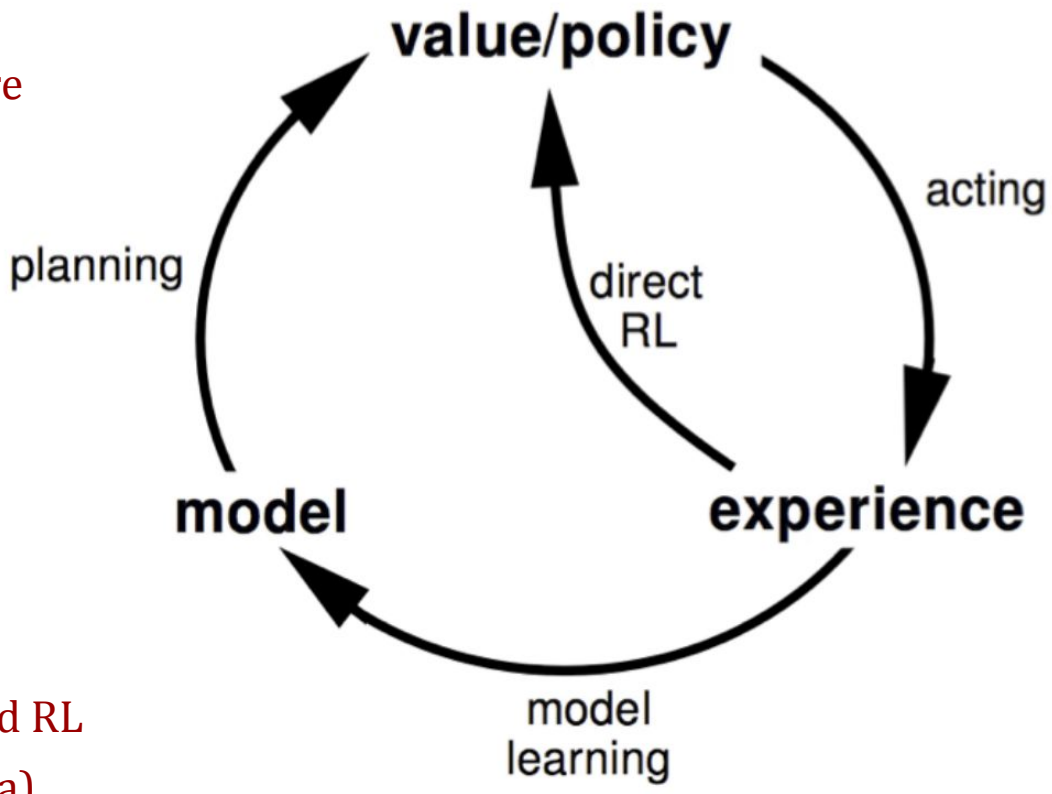
Model-based RL

(e.g., Dyna)

Model-based Reinforcement Learning

Main benefit:

Potentially more data efficient



Model-based RL

(e.g., Dyna)

Model-based Reinforcement Learning

Two important steps:

1. How to learn a model from data? (Sec. 5)

2. How to integrate planning updates and learning updates? (Sec. 6)

5) Learning a model

Learning a model

Main question:

Given a dataset of observed transitions (s,a,r,s'), how can we estimate

1) the dynamics p(s'|s,a) and

2) The reward r(s,a,s') function?

Learning a model

Main question:

Given a dataset of observed transitions (s,a,r,s'), how can we estimate

1) **the dynamics p(s'|s,a) and**

2) The reward r(s,a,s') function?

Dynamics estimation

1. Track counts **n(s,a,s')**
   a. Number of times we observed s' after taking a in s.

Dynamics estimation

1. Track counts **n(s,a,s')**
   a. Number of times we observed s' after taking a in s.
   b. Can estimate from transition data (s,a,r,s')

Dynamics estimation

1. Track counts **n(s,a,s')**
   a. Number of times we observed s' after taking a in s.
   b. Can estimate from transition data (s,a,r,s')
   c. Array of size |S| x |A| x |S|

Dynamics estimation

1. Track counts **n(s,a,s')**
   a. Number of times we observed s' after taking a in s.
   b. Can estimate from transition data (s,a,r,s')
   c. Array of size |S| x |A| x |S|

2. Estimate p(s'|s,a) by normalizing the observed counts:

$$\hat{p}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$$

Dynamics estimation

1. Track counts **n(s,a,s')**
   a. Number of times we observed s' after taking a in s.
   b. Can estimate from transition data (s,a,r,s')
   c. Array of size |S| x |A| x |S|

2. Estimate p(s'|s,a) by normalizing the observed counts:

Number of times we observed s' after s,a

$$\hat{p}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$$

Total number of trials at s,a

Dynamics estimation

1. Track counts **n(s,a,s')**
   a. Number of times we observed s' after taking a in s.
   b. Can estimate from transition data (s,a,r,s')
   c. Array of size |S| x |A| x |S|

2. Estimate p(s'|s,a) by normalizing the observed counts:

Number of times we observed s' after s,a

$$\hat{p}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)} = \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')}$$

Total number of trials at s,a

Dynamics estimation

Example: After taking a=1 in s=1, we have stored the following counts:

- n(s=1,a=1,s'=1) = 4
- n(s=1,a=1,s'=2) = 2
- n(s=1,a=1,s'=3) = 6

Dynamics estimation

Example: After taking a=1 in s=1, we have stored the following counts:

- n(s=1,a=1,s'=1) = 4
- n(s=1,a=1,s'=2) = 2
- n(s=1,a=1,s'=3) = 6

Q: Compute p(s'|s=1,a=1)

$$\hat{p}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$$

Dynamics estimation

Example: After taking a=1 in s=1, we have stored the following counts:

- n(s=1,a=1,s'=1) = 4
- n(s=1,a=1,s'=2) = 2
- n(s=1,a=1,s'=3) = 6

Q: Compute p(s'|s=1,a=1)

$$\hat{p}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$$

A:

- p(s'=1|s=1,a=1) = 4/12 = ⅓
- p(s'=2|s=1,a=1) = 2/12 = ⅙
- p(s'=3|s=1,a=1) = 6/12 = ½

Learning a model

Main question:

Given a dataset of observed transitions (s,a,r,s'), how can we estimate

1) the dynamics p(s'|s,a) and

2) the reward r(s,a,s') function?

Reward function estimation

1. Also track total transition rewards $\mathbf{R_{sum}(s,a,s')}$
   a. Sum of all observed rewards when reaching s' after taking a in s.

Reward function estimation

1. Also track total transition rewards $\mathbf{R_{sum}(s,a,s')}$
   a. Sum of all observed rewards when reaching s' after taking a in s.
   b. Can estimate from transition data (s,a,r,s')
   c. Array of size |S| x |A| x |S|

Reward function estimation

1. Also track total transition rewards $\mathbf{R_{sum}(s,a,s')}$
   a. Sum of all observed rewards when reaching s' after taking a in s.
   b. Can estimate from transition data (s,a,r,s')
   c. Array of size |S| x |A| x |S|

2. Estimate r(s,a,s') by computing the average transition reward:

$$\hat{r}(s,a,s') = \frac{R_{\text{sum}}(s,a,s')}{n(s,a,s')}$$

Reward function estimation

1. Also track total transition rewards $\mathbf{R_{sum}(s,a,s')}$
   a. Sum of all observed rewards when reaching s' after taking a in s.
   b. Can estimate from transition data (s,a,r,s')
   c. Array of size |S| x |A| x |S|

2. Estimate r(s,a,s') by computing the average transition reward:

Total observed reward

$$\hat{r}(s,a,s') = \frac{R_{\text{sum}}(s,a,s')}{n(s,a,s')}$$

Number of transitions

Model estimation pseudocode

Full pseudocode in the lecture notes

Model estimation pseudocode

Full pseudocode in the lecture notes

Algorithm 1: Tabular model update pseudo-code. PS = prioritized sweeping.

Input: Maximum number of timesteps T.

Initialization: Initialize $n(s, a, s') = 0$ and $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

repeat T **times**

Observe $\langle s, a, r, s' \rangle$	/* Observe new transition */
$n(s, a, s') \leftarrow n(s, a, s') + 1$	/* Update transition counts */
$R_{sum}(s, a, s') \leftarrow R_{sum}(s, a, s') + r$	/* Update reward sums */
$\hat{p}(s'\|s, a) = \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')}$	/* Estimate transition function */
$\hat{r}(s, a, s') = \frac{R_{sum}(s,a,s')}{n(s,a,s')}$	/* Estimate reward function */
$\hat{p}(s, a\|s') = \frac{n(s,a,s')}{\sum_{s,a} n(s,a,s')}$	/* Reverse model (only for PS) */

end

Summary 5: Learning a model

Estimate tabular p(s'|s,a) and r(s,a,s') from observed data tuples <s,a,r,s'>

Summary 5: Learning a model

Estimate tabular p(s'|s,a) and r(s,a,s') from observed data tuples <s,a,r,s'>

1. Maintain counts n(s,a,s') and total rewards $R_{sum}(s,a,s')$.

Summary 5: Learning a model

Estimate tabular p(s'|s,a) and r(s,a,s') from observed data tuples <s,a,r,s'>

1. Maintain counts n(s,a,s') and total rewards $R_{sum}(s,a,s')$.

2. Compute transition model p(s'|s,a) from normalizing counts.

Summary 5: Learning a model

Estimate tabular p(s'|s,a) and r(s,a,s') from observed data tuples <s,a,r,s'>

1. Maintain counts n(s,a,s') and total rewards $R_{sum}(s,a,s')$.
2. Compute transition model p(s'|s,a) from normalizing counts.
3. Compute reward model r(s,a,s') from averaging total rewards.

Summary 5: Learning a model

Estimate tabular p(s'|s,a) and r(s,a,s') from observed data tuples <s,a,r,s'>

1. Maintain counts n(s,a,s') and total rewards $R_{sum}(s,a,s')$.
2. Compute transition model p(s'|s,a) from normalizing counts.
3. Compute reward model r(s,a,s') from averaging total rewards.

Next section: how may this model be useful?

6) Model-based RL algorithms

Model-based RL algorithms

We want to make use of the learned model!

(combine planning and learning)

Model-based RL algorithms

Discuss three algorithms:

a) Real-time Dynamic Programming (RTDP)

b) Dyna

c) Prioritized sweeping

Model-based RL algorithms

Discuss three algorithms:

a) **Real-time Dynamic Programming (RTDP)**

b) Dyna

c) Prioritized sweeping

a) Real-time Dynamic Programming

We already discussed Dynamic Programming (DP) in an earlier lecture

a) Real-time Dynamic Programming

We already discussed Dynamic Programming (DP) in an earlier lecture

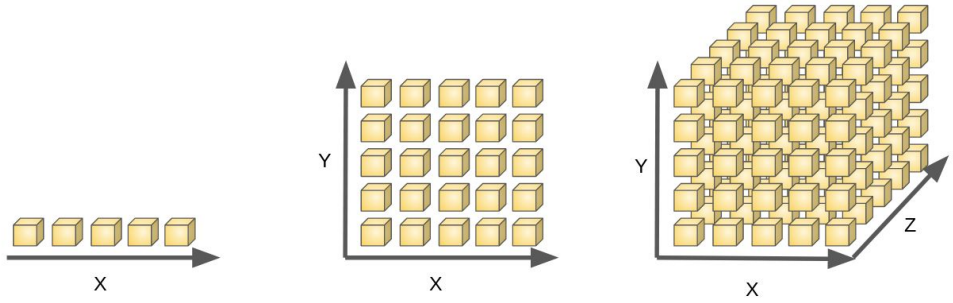
- Classic bridging algorithm between planning and learning
- Sweep through the entire state space
- At each state update with Bellman optimality equation
- Guaranteed convergence

a) Real-time Dynamic Programming

We already discussed Dynamic Programming (DP) in an earlier lecture

- Classic bridging algorithm between planning and learning
- Sweep through the entire state space
- At each state update with Bellman optimality equation
- Guaranteed convergence

Main problem = curse of dimensionality

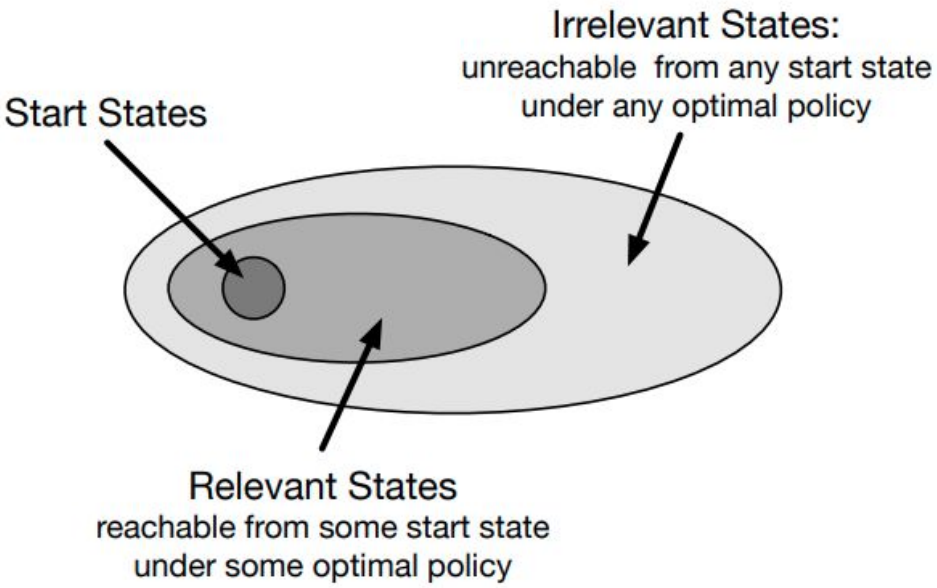


Real-time Dynamic Programming

Insight: many states are often not even reachable from the start

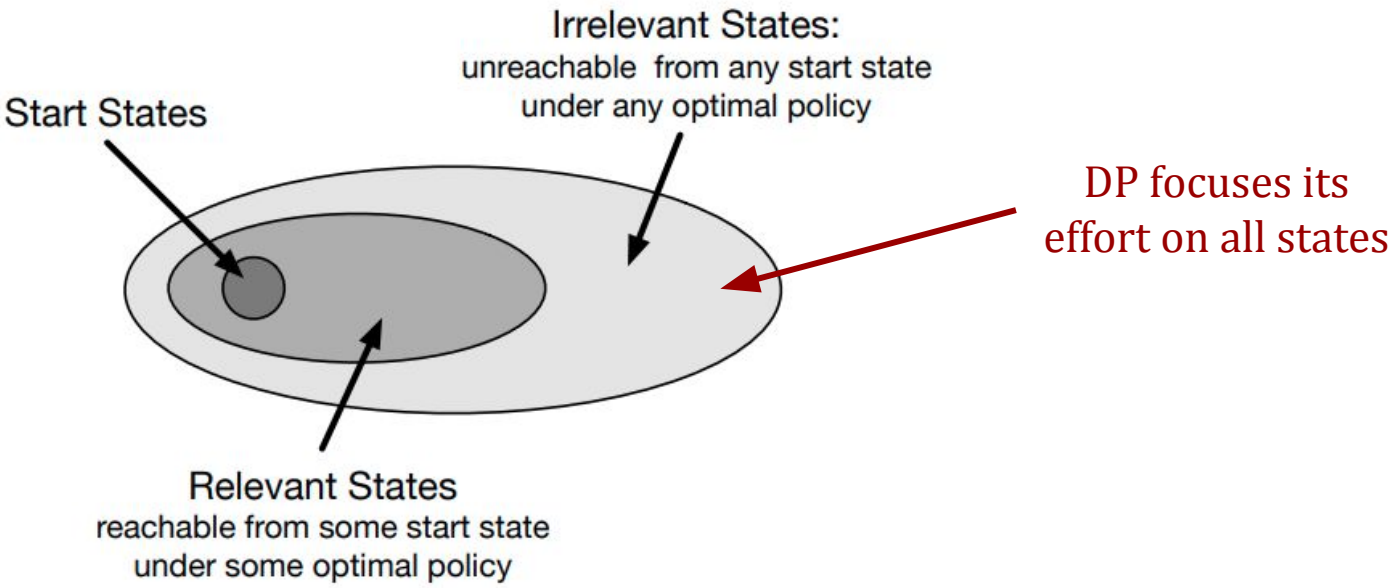
Real-time Dynamic Programming

Insight: many states are often not even reachable from the start



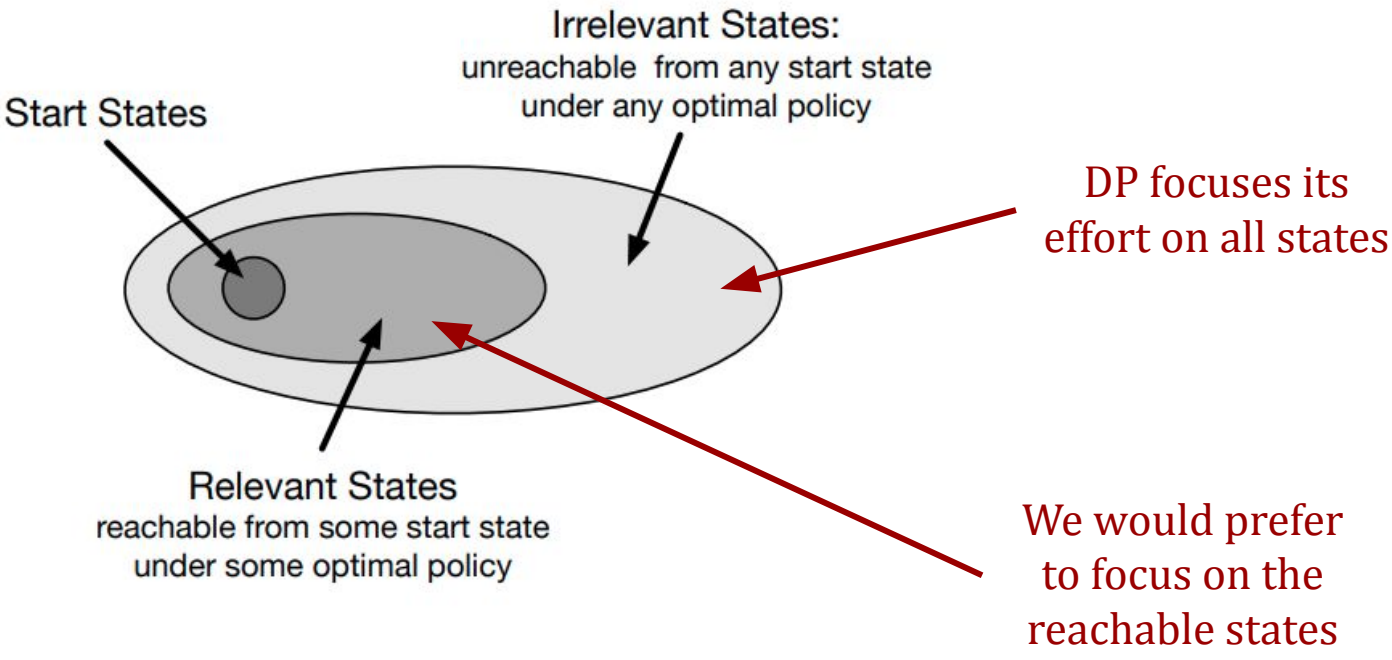
Real-time Dynamic Programming

Insight: many states are often not even reachable from the start



Real-time Dynamic Programming

Insight: many states are often not even reachable from the start



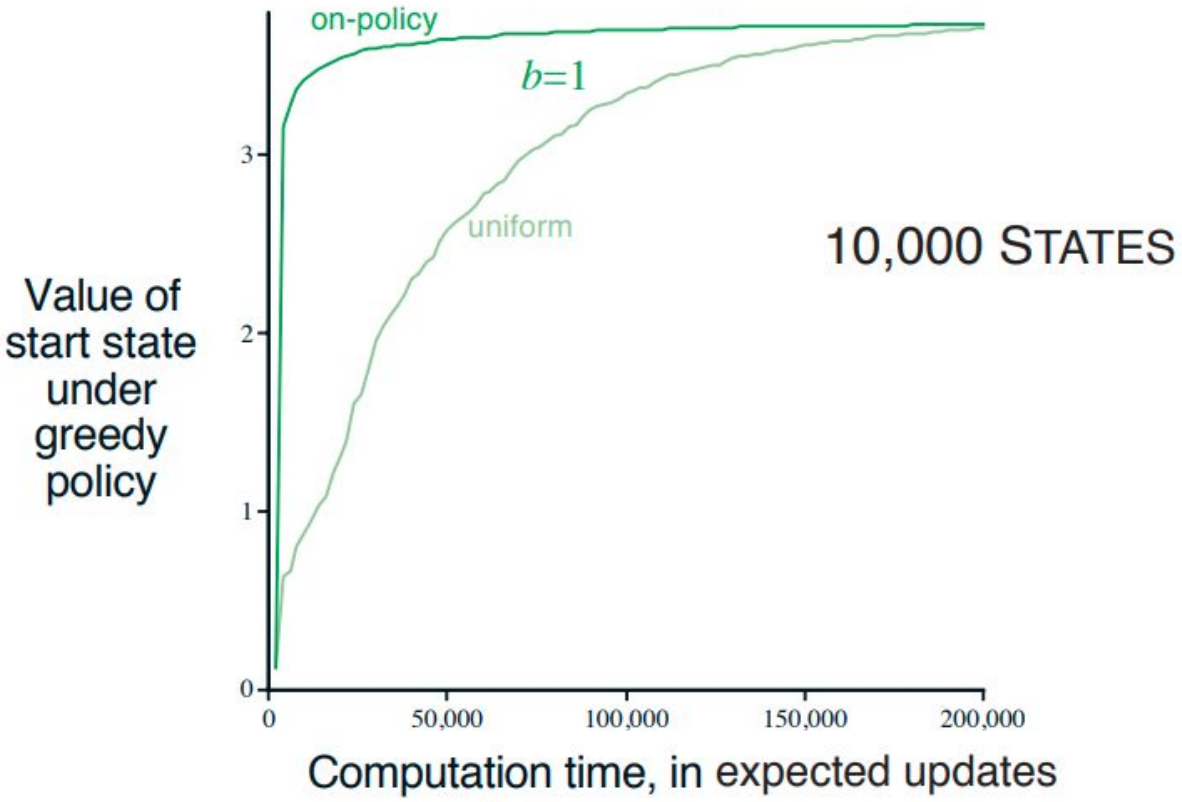
Real-time Dynamic Programming

Solution of **real-time DP:**

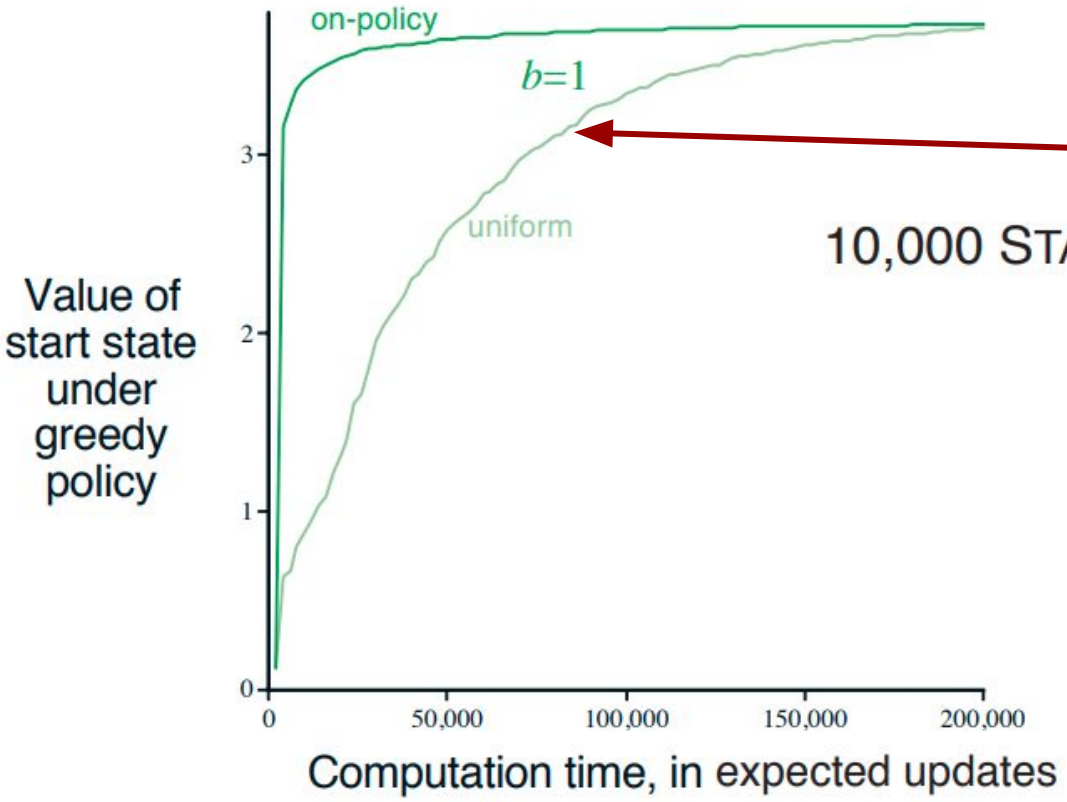
apply DP updates on traces sampled from the start

(All model-free RL approaches do this by definition,
since the access to the MDP is then irreversible)

Real-time Dynamic Programming

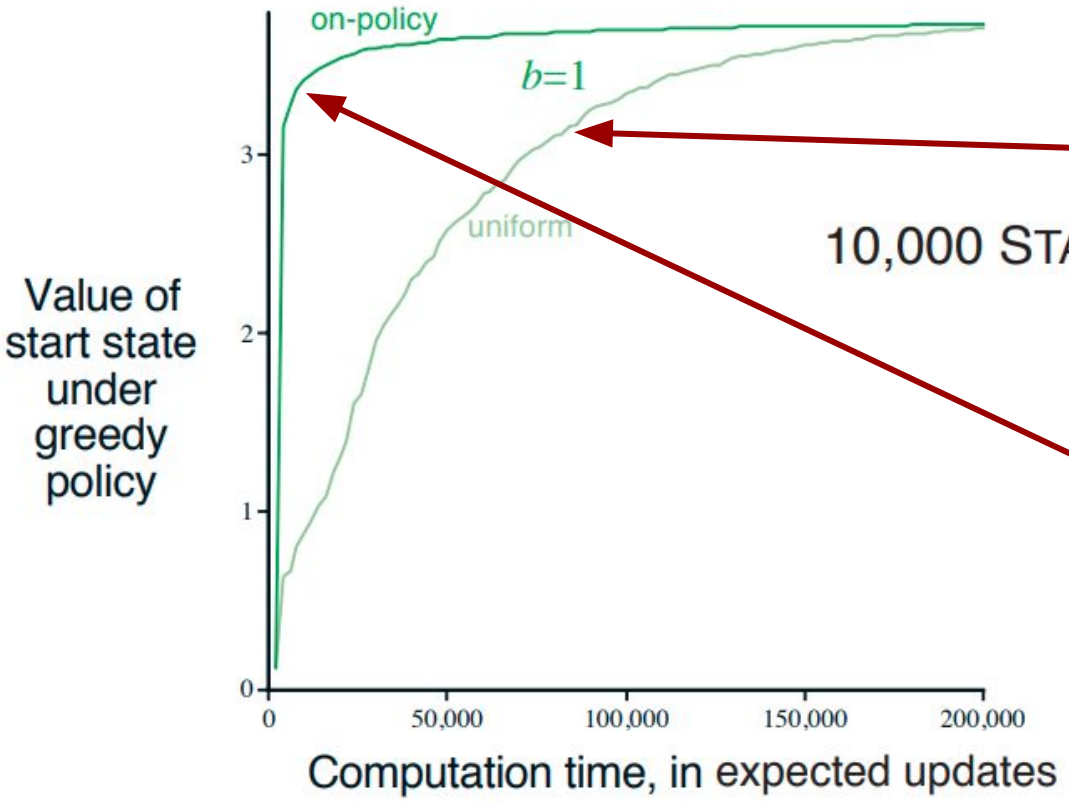


Real-time Dynamic Programming



DP updates are uniform across state space

Real-time Dynamic Programming



DP updates are uniform across state space

In larger problems updating on trajectories from the start is often preferable

Model-based RL algorithms

Discuss three algorithms:

a) Real-time Dynamic Programming (RTDP)

b) Dyna

c) Prioritized sweeping

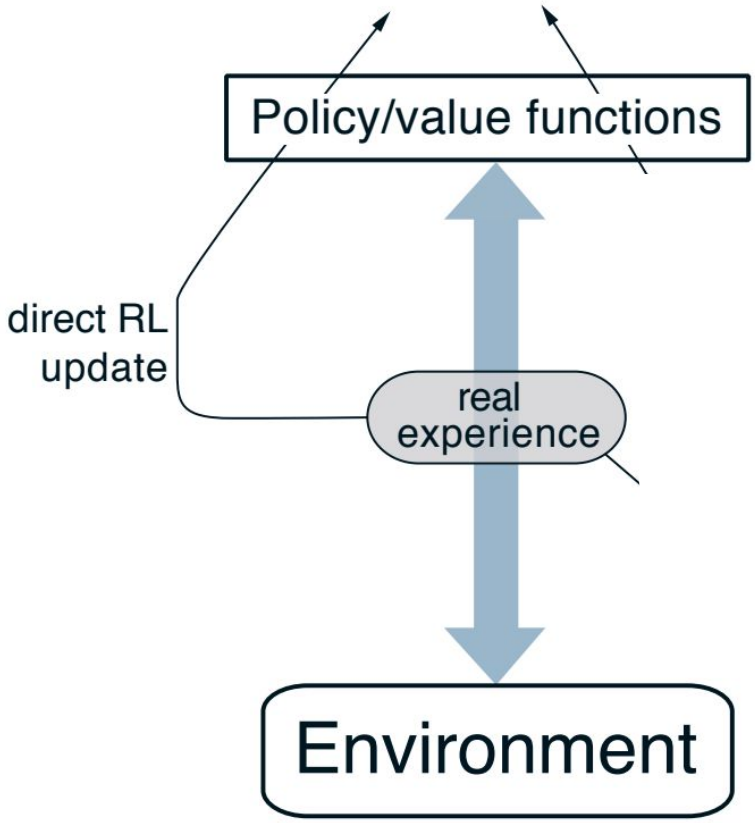
Dyna

Main idea:

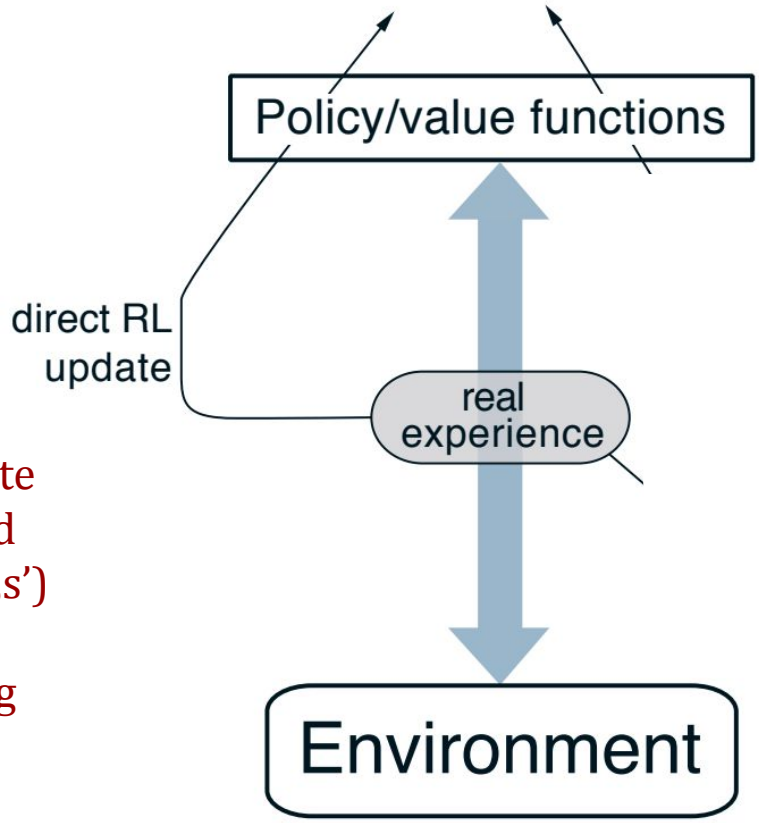
Learn a model to generate additional transition data,

apply standard update to these simulated transitions.

Dyna


Policy/value functions
direct RL
update
real
experience
Environment

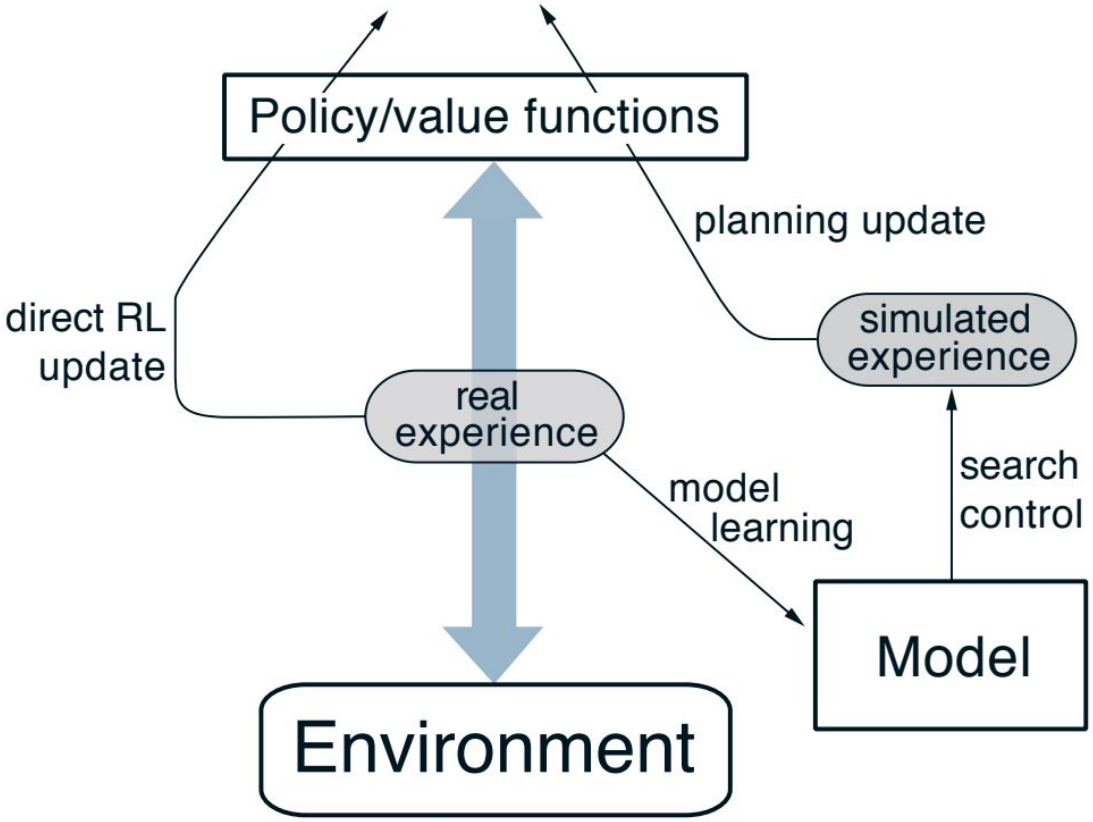
Dyna



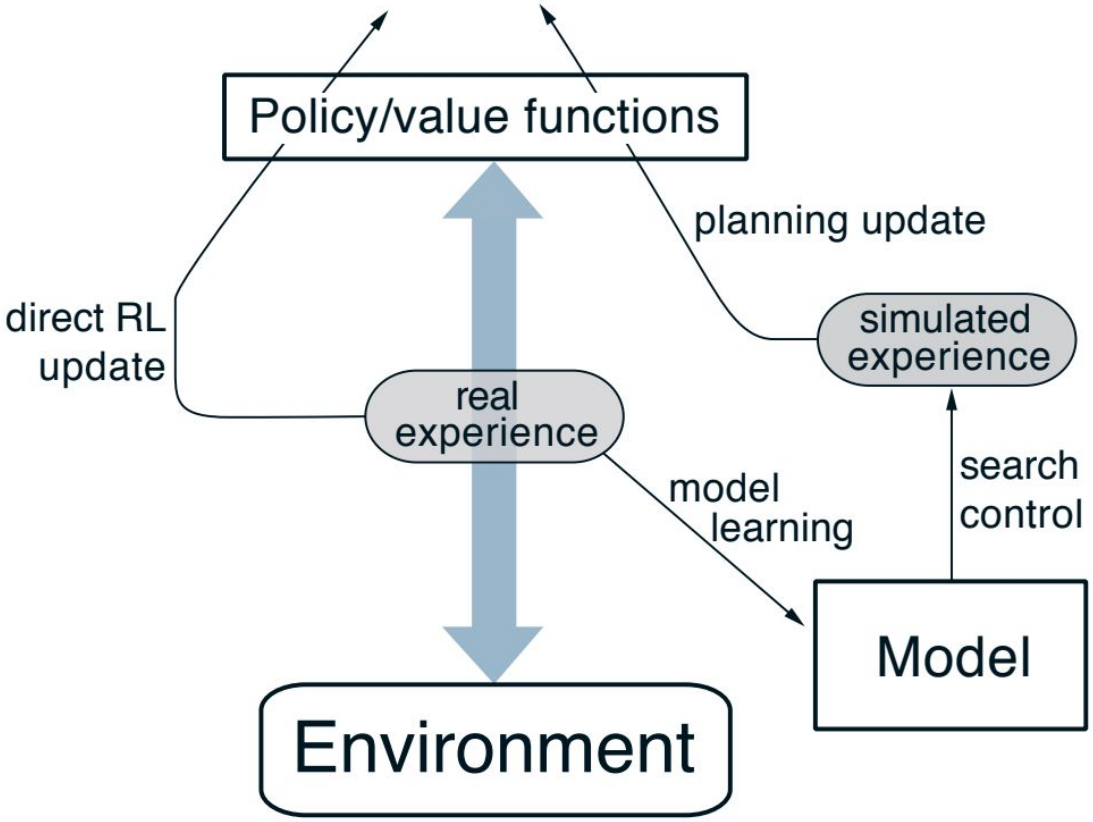
Standard update from observed transition (s,a,r,s')

E.g. Q-learning

Dyna

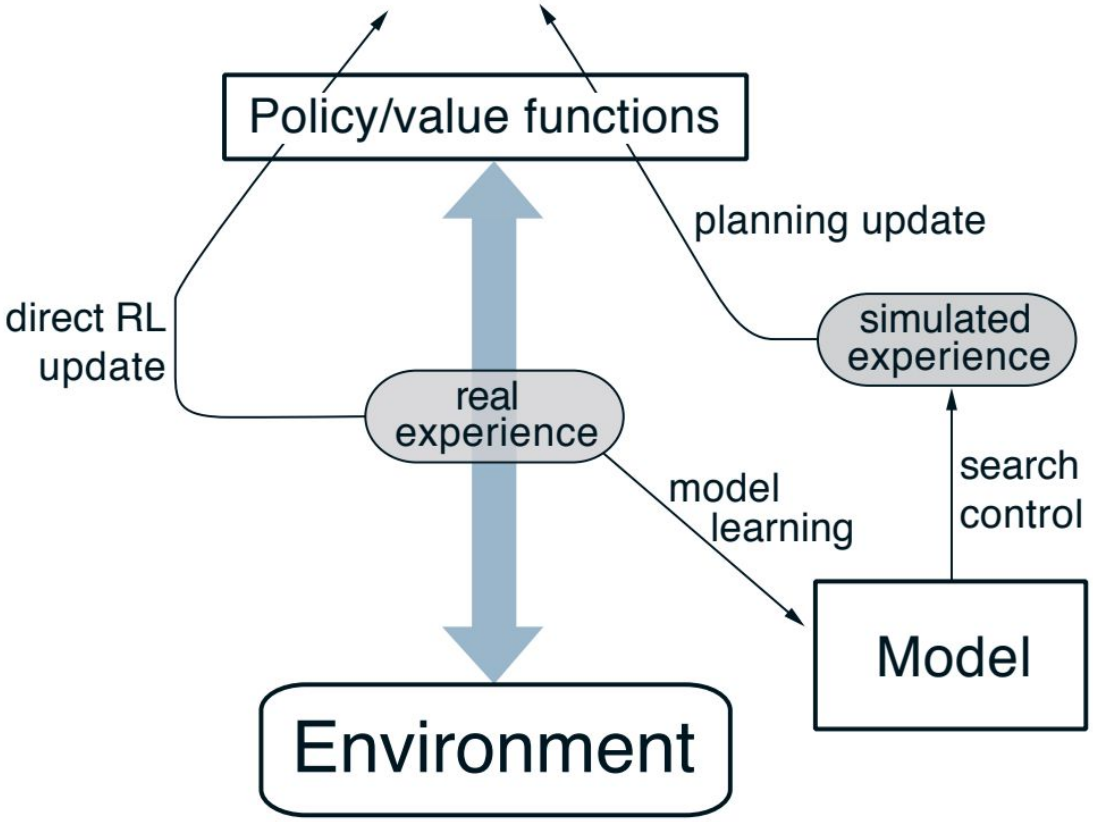

Policy/value functions
planning update
direct RL
update
simulated
experience
real
experience
search
control
model
learning
Model
Environment

Dyna



learn model

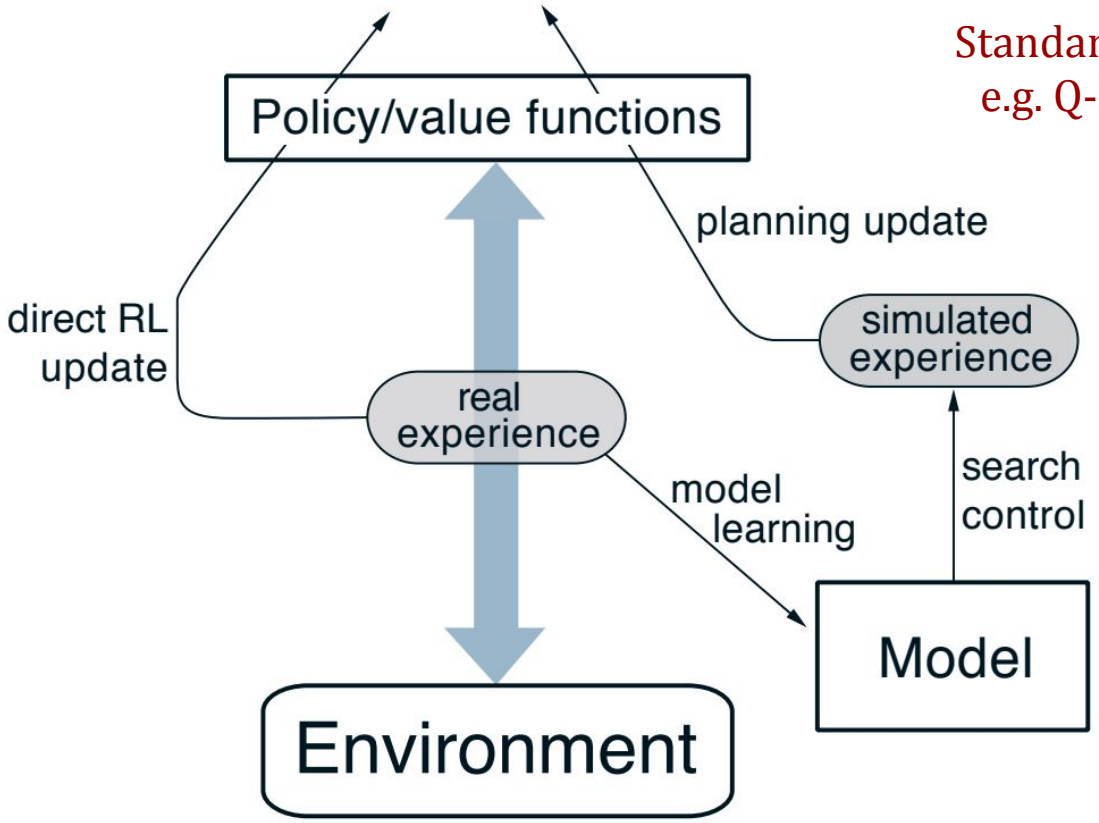
Dyna



Use model to sample new (s,a,r,s')

learn model

Dyna



Standard update, e.g. Q-learning

Use model to sample new (s,a,r,s')

learn model

Dyna

Algorithm 2: *Dyna* Q-learning with ϵ-greedy exploration.

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`

end

(Q-learning)

Dyna

Algorithm 2: *Dyna* Q-learning with ϵ-greedy exploration.

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — /* Reset when environment terminates */
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — /* Sample action */
- $r, s' \sim p(r, s'|s, a)$ — /* Simulate environment */
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — /* Update model (Alg.1) */
- $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — /* Update Q-table */
- **repeat** K **times**
  - $s \leftarrow$ random previously observed state — /* State to plan on */
  - $a \leftarrow$ previously taken action in state s — /* Planning action */
  - $s', r \sim \hat{p}(s', r|s, a)$ — /* Simulate model */
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — /* Update Q-table */
- **end**

end

Learn model

Make k planning updates in between every real step

Dyna

Algorithm 2: *Dyna* Q-learning with ϵ-greedy exploration.

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$.

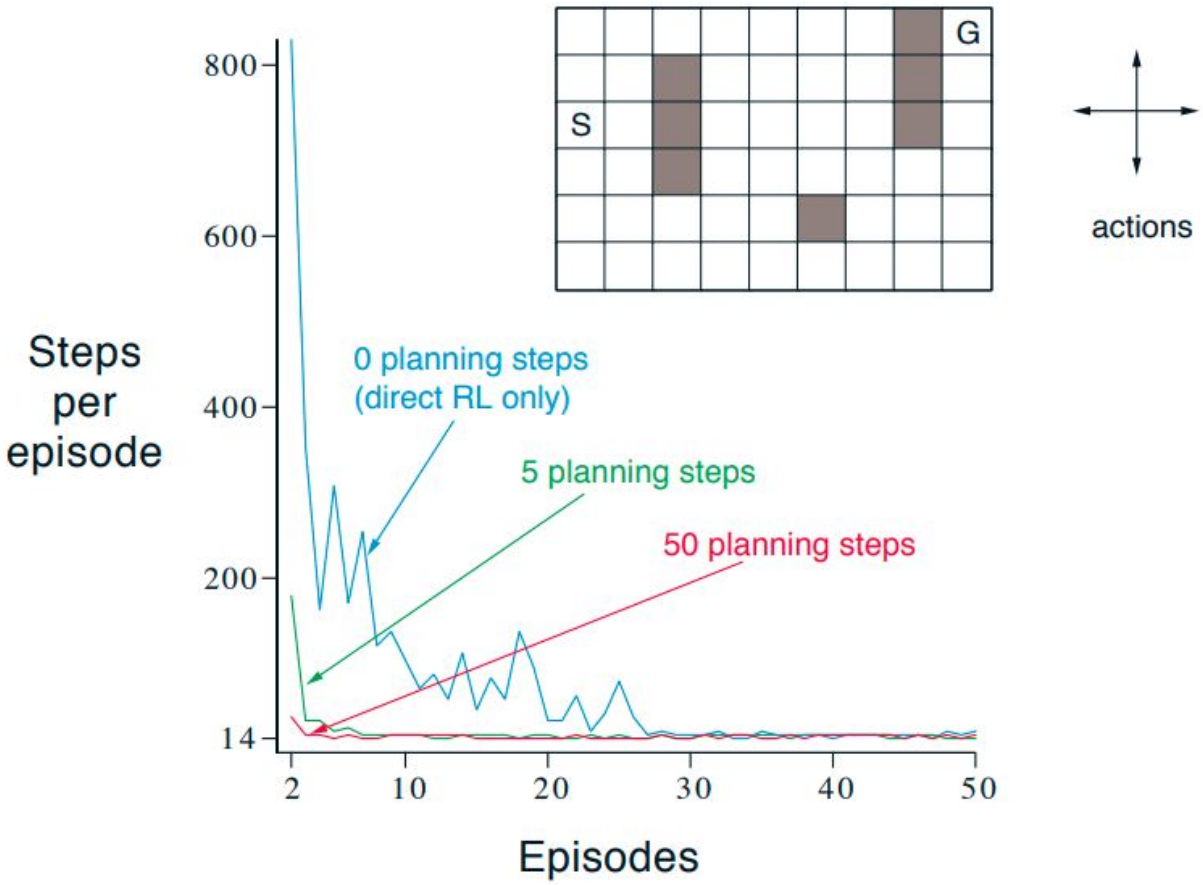
for $t = 1...T$ **do**

- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
- **repeat** K **times**
  - $s \leftarrow$ random previously observed state — `/* State to plan on */`
  - $a \leftarrow$ previously taken action in state s — `/* Planning action */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
- **end**

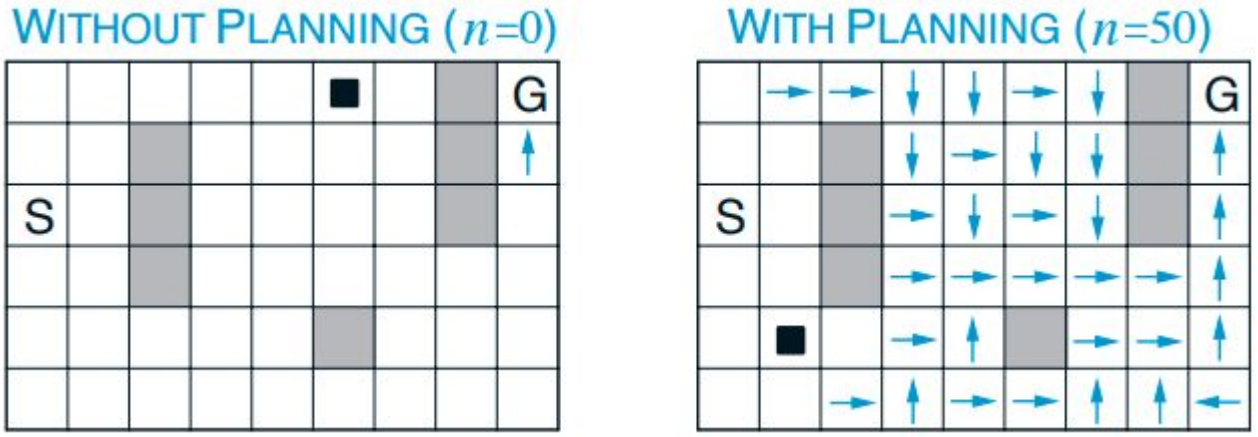
end

Algorithm you will implement in the assignment!

Dyna


G
S
actions
800
600
400
200
14
Steps per episode
0 planning steps (direct RL only)
5 planning steps
50 planning steps
2
10
20
30
40
50
Episodes

Dyna



Model-based RL algorithms

Discuss three algorithms:

a) Real-time Dynamic Programming (RTDP)

b) Dyna

c) Prioritized sweeping

Prioritized sweeping

If the value estimate of a state changes a lot,

then the states that precede it should probably also be updated.

Prioritized sweeping

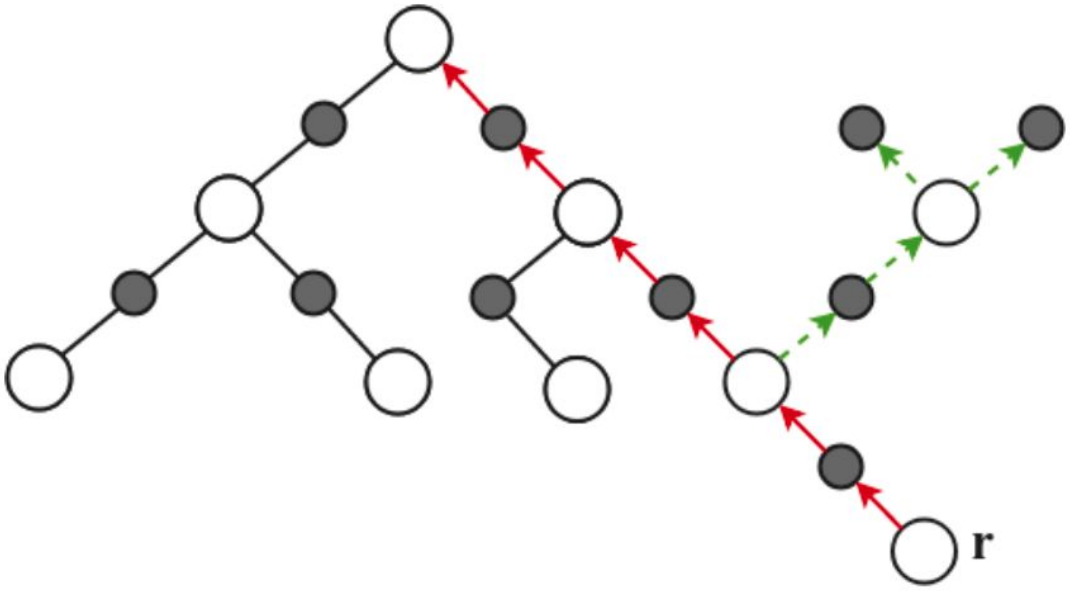
If the value estimate of a state changes a lot,

then the states that precede it should probably also be updated.

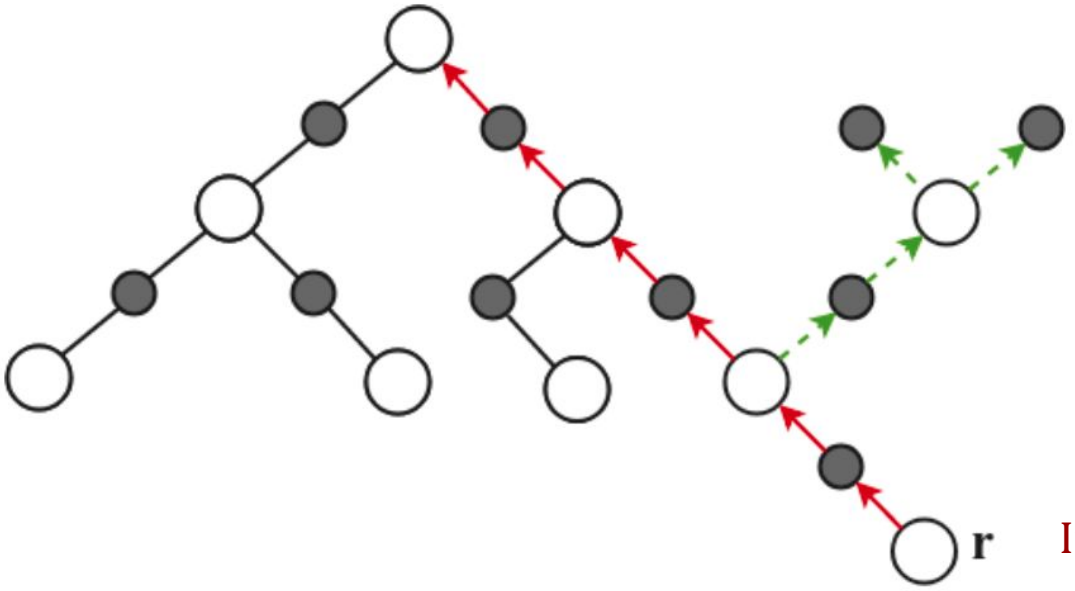
Main idea:

Use a backward/reverse model to identify states that likely need updating (backwards search to spread information faster)

Prioritized sweeping

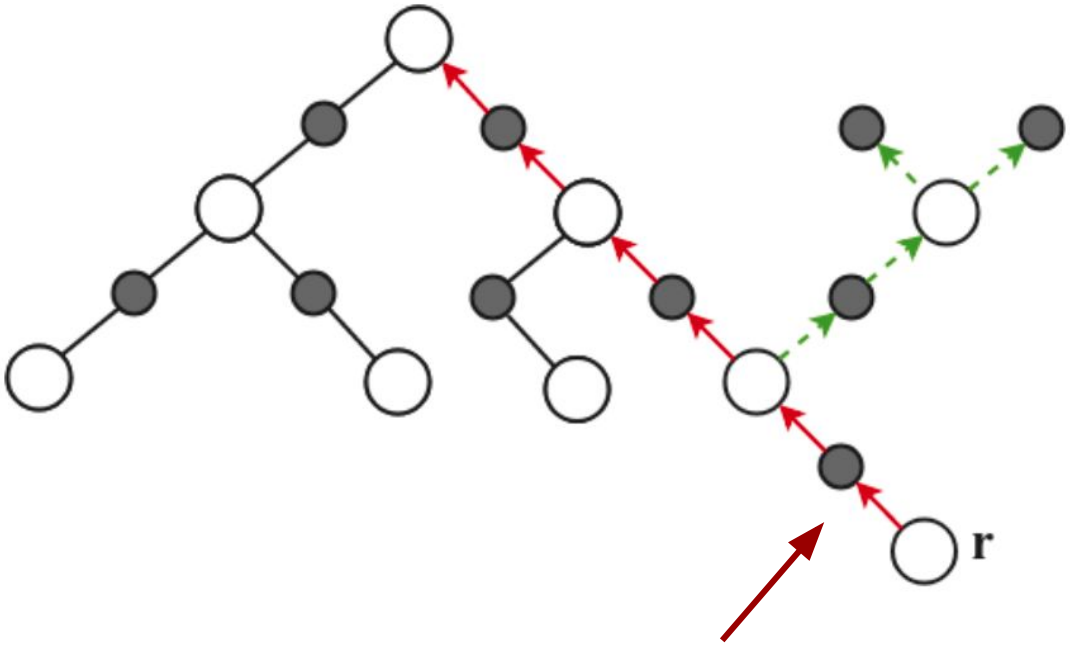


Prioritized sweeping



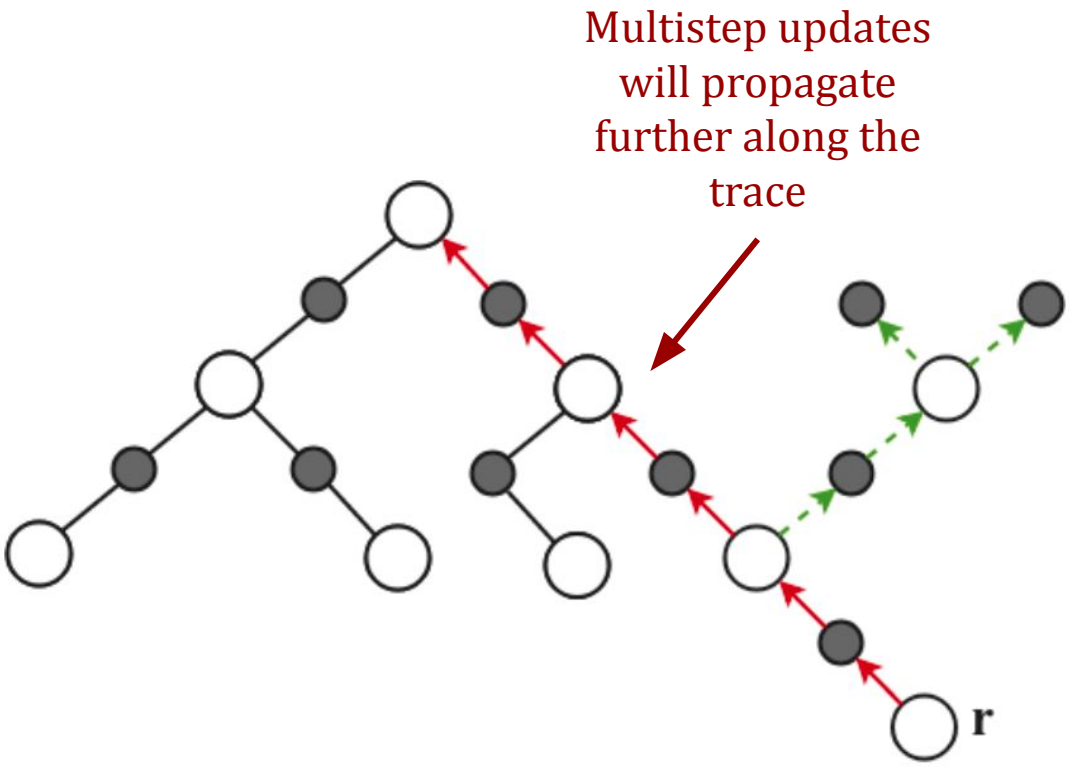
Imagine we find a new big reward

Prioritized sweeping

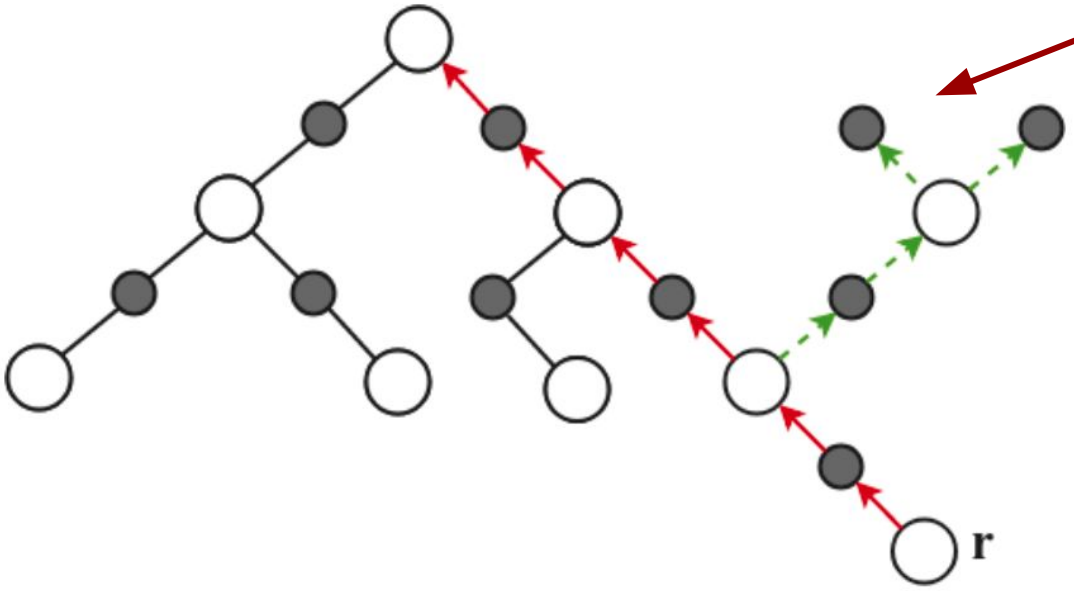


Standard one-step update will only back-up one step

Prioritized sweeping

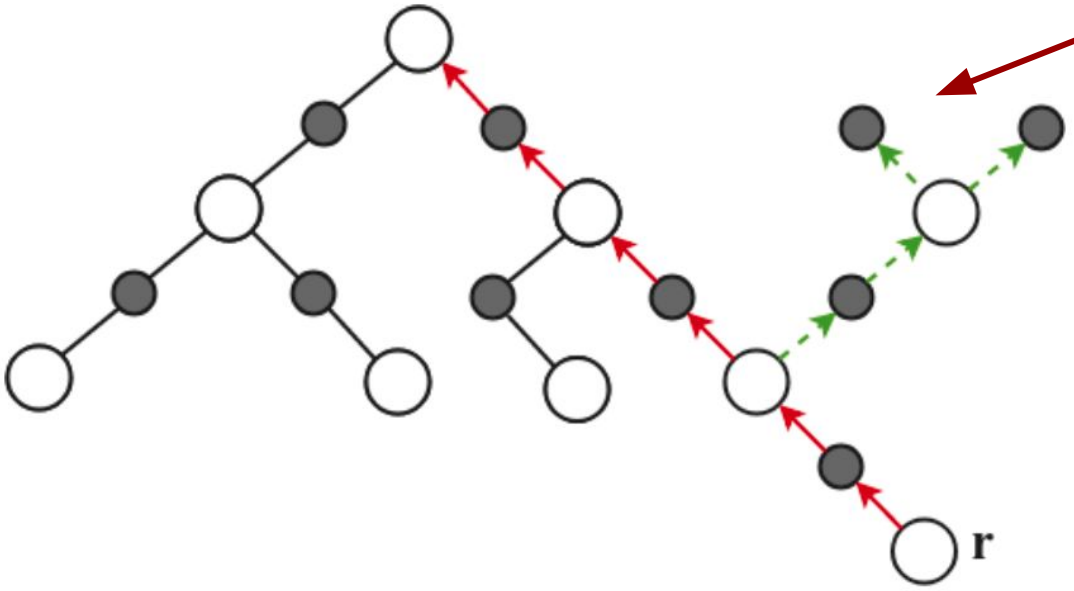


Prioritized sweeping



Prioritized sweeping identifies states that may also lead to the new information, and *prioritizes* these for updating

Prioritized sweeping



Prioritized sweeping identifies states that may also lead to the new information, and *prioritizes* these for updating

Prioritized sweeping

Main idea: *prioritize* states that deserve an update & additional backward search

Prioritized sweeping

Main idea: *prioritize* states that deserve an update & additional backward search

- Maintain priority queue, with priority as the absolute TD error

$$\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$$

New Q-learning estimate

Current estimate

Prioritized sweeping

Pseudocode in assignment/lecture notes

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T, priority threshold θ.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — `/* Compute priority p */`
- **if** $\mathbf{p} > \theta$ **then**
  - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
- **end**
- **/// Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — `/* Sample PQ, break when empty */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
  - **/// Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — `/* Get reward from model */`
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_{a} \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — `/* Compute priority p */`
    - **if** $\mathbf{p} > \theta$ **then**
      - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
    - **end**
  - **end**
- **end**

end

Prioritized sweeping

Pseudocode in assignment/lecture notes

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T, priority threshold θ.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

Maintain a queue

for $t = 1...T$ **do**
- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — `/* Compute priority p */`
- **if** $\mathbf{p} > \theta$ **then**
  - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
- **end**
- **/// Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — `/* Sample PQ, break when empty */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
  - **/// Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — `/* Get reward from model */`
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_{a} \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — `/* Compute priority p */`
    - **if** $\mathbf{p} > \theta$ **then**
      - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
    - **end**
  - **end**
- **end**

end

Prioritized sweeping

Pseudocode in assignment/lecture notes

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T, priority threshold θ.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — `/* Compute priority p */`
- **if** $\mathbf{p} > \theta$ **then**
  - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
- **end**
- **/// Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — `/* Sample PQ, break when empty */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
  - **/// Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — `/* Get reward from model */`
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_{a} \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — `/* Compute priority p */`
    - **if** $\mathbf{p} > \theta$ **then**
      - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
    - **end**
  - **end**
- **end**

end

Maintain a queue

Learn a forward and backward model

Prioritized sweeping

Pseudocode in assignment/lecture notes

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T, priority threshold θ.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — `/* Compute priority p */`
- **if** $\mathbf{p} > \theta$ **then**
  - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
- **end**
- **/// Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — `/* Sample PQ, break when empty */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
  - **/// Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — `/* Get reward from model */`
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_{a} \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — `/* Compute priority p */`
    - **if** $\mathbf{p} > \theta$ **then**
      - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
    - **end**
  - **end**
- **end**

end

Maintain a queue

Learn a forward and backward model

Add to queue based on absolute error

Prioritized sweeping

Pseudocode in assignment/lecture notes

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T, priority threshold θ.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — `/* Compute priority p */`
- **if** $\mathbf{p} > \theta$ **then**
  - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
- **end**
- /// **Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — `/* Sample PQ, break when empty */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
  - /// **Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — `/* Get reward from model */`
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_a \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — `/* Compute priority p */`
    - **if** $\mathbf{p} > \theta$ **then**
      - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
    - **end**
  - **end**
- **end**

end

Maintain a queue

Learn a forward and backward model

Add to queue based on absolute error

Pop from queue

Prioritized sweeping

Pseudocode in assignment/lecture notes

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T, priority threshold θ.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — /* Reset when environment terminates */
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — /* Sample action */
- $r, s' \sim p(r, s'|s, a)$ — /* Simulate environment */
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — /* Update model (Alg. 1) */
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — /* Compute priority p */
- **if** $\mathbf{p} > \theta$ **then**
  - Insert (s, a) into PQ with priority $\mathbf{p}$ — /* State-action needs update */
- **end**
- /// **Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — /* Sample PQ, break when empty */
  - $s', r \sim \hat{p}(s', r|s, a)$ — /* Simulate model */
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — /* Update Q-table */
  - /// **Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — /* Get reward from model */
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_{a} \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — /* Compute priority p */
    - **if** $\mathbf{p} > \theta$ **then**
      - Insert (s, a) into PQ with priority $\mathbf{p}$ — /* State-action needs update */
    - **end**
  - **end**
- **end**

end

Maintain a queue

Learn a forward and backward model

Add to queue based on absolute error

Pop from queue

Update Q table

Prioritized sweeping

Pseudocode in assignment/lecture notes

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T, priority threshold θ.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — /* Reset when environment terminates */
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — /* Sample action */
- $r, s' \sim p(r, s'|s, a)$ — /* Simulate environment */
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — /* Update model (Alg.1) */
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — /* Compute priority p */
- **if** $\mathbf{p} > \theta$ **then**
  - Insert (s, a) into PQ with priority $\mathbf{p}$ — /* State-action needs update */
- **end**
- /// **Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — /* Sample PQ, break when empty */
  - $s', r \sim \hat{p}(s', r|s, a)$ — /* Simulate model */
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — /* Update Q-table */
  - /// **Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — /* Get reward from model */
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_{a} \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — /* Compute priority p */
    - **if** $\mathbf{p} > \theta$ **then**
      - Insert (s, a) into PQ with priority $\mathbf{p}$ — /* State-action needs update */
    - **end**
  - **end**
- **end**

end

Maintain a queue

Learn a forward and backward model

Add to queue based on absolute error

Pop from queue

Update Q table

Loop all precursors, potentially add to queue

Prioritized sweeping

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T, priority threshold θ.

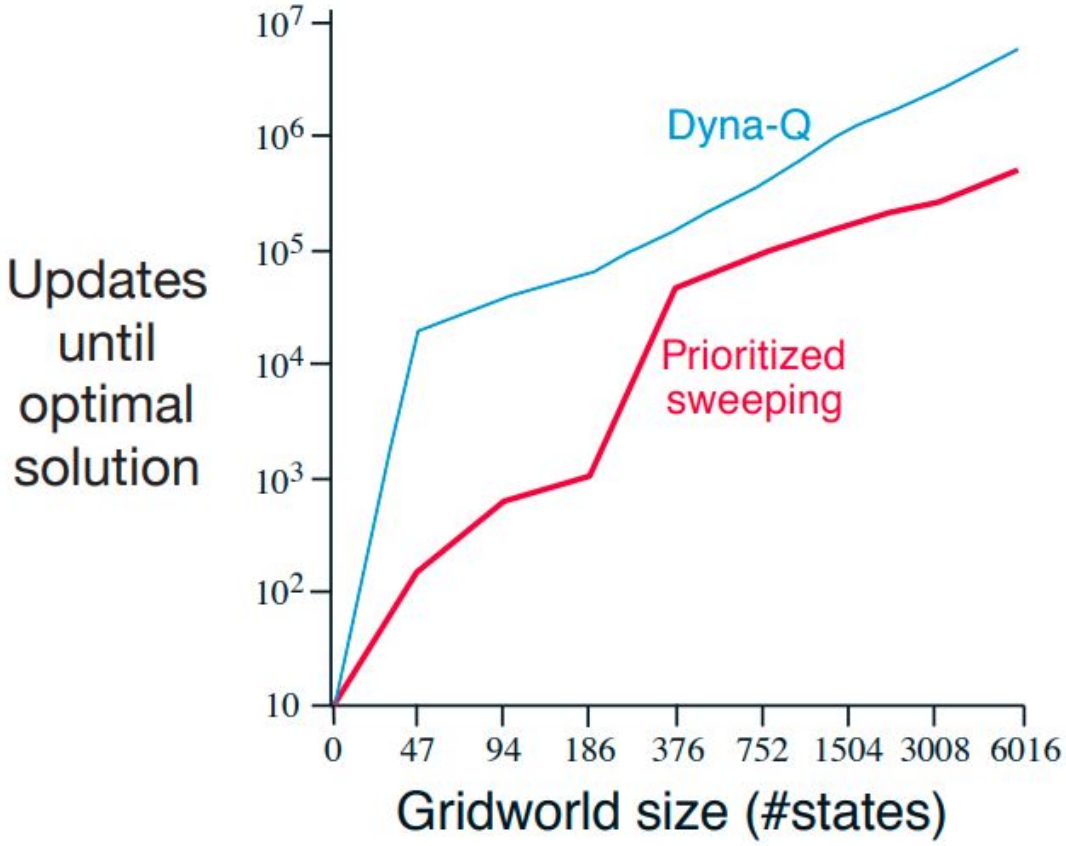
Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

for $t = 1...T$ **do**
- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — `/* Compute priority p */`
- **if** $\mathbf{p} > \theta$ **then**
  - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
- **end**
- **/// Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — `/* Sample PQ, break when empty */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
  - **/// Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — `/* Get reward from model */`
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_a \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — `/* Compute priority p */`
    - **if** $\mathbf{p} > \theta$ **then**
      - Insert (s, a) into PQ with priority $\mathbf{p}$ — `/* State-action needs update */`
    - **end**
  - **end**
- **end**

end

You will also implement this algorithm for the assignment!

Prioritized sweeping



Summary 6: Model-based RL algorithms

Planning and learning can be combined in a variety of ways

Summary 6: Model-based RL algorithms

Planning and learning can be combined in a variety of ways

- E.g., Dyna
  - Learn forward model
  - Use model to sample additional transition data
  - Apply standard model-free RL update to simulated experience as well.

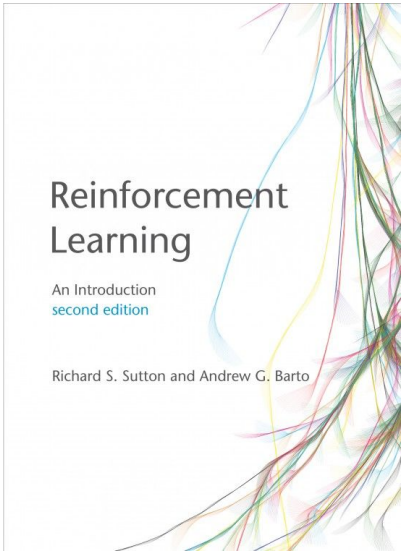
Summary 6: Model-based RL algorithms

Planning and learning can be combined in a variety of ways

- E.g., Dyna
  - Learn forward model
  - Use model to sample additional transition data
  - Apply standard model-free RL update to simulated experience as well.

- E.g., Prioritized sweeping
  - Learn a backward model
  - Use backward model to identify states that will likely change on the next update
  - Prioritize these states for updating in a separate queue

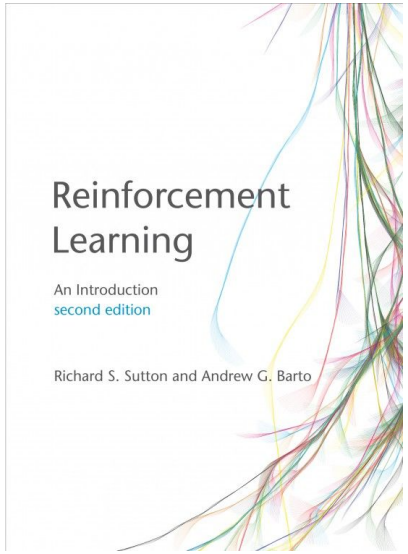
To Do



Read

- Sutton and Barto, Chapter 8
- Lecture notes
- Take care to study the back-up diagrams and associated equations!

To Do

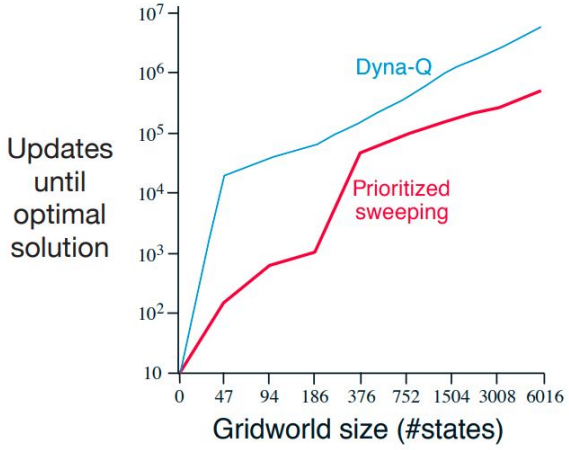


Read

- Sutton and Barto, Chapter 8
- Lecture notes
- Take care to study the back-up diagrams and associated equations!

Assignment:

1. Implement two MBRL algorithms:
   a. Dyna
   b. Prioritized sweeping
2. Investigate their performance
3. Write a report



Questions?