

---

# Model-based reinforcement learning

---

Herke van Hoof

---

---

# Exam info

---

There are example exams on Canvas

Prioritize “What you should know” or “Conclusions” provided for most lectures!

No tools (calculator). We’ll provide a sheet with the policy and value update equations.

I’ll use the last lecture to summarise the main methods for the exam and answer questions about the material if there’s time left

---

---

# Reproducible research lab info

---

Next monday (October 18th): peer feedback

Bring an initial write-up for the lab. Can be incomplete (at least 1 copy for each team member)

Doesn't need to be with group together

Switch write-ups with someone else.

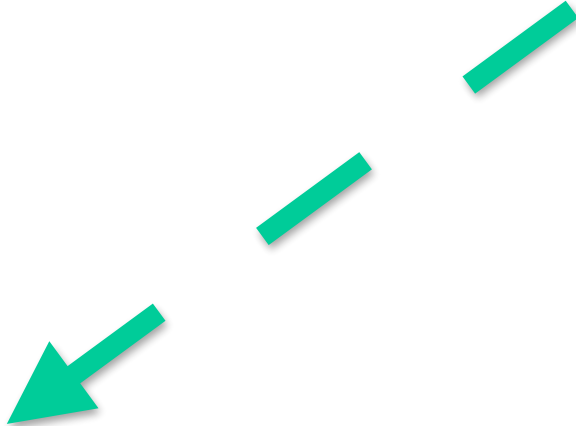
- Hour 1: Carefully read proposal and fill out feedback form
  - Hour 2: finish form write-up, discuss with the person you switched with (15 minutes each). **Take photo of feedback & upload to Canvas.** Hand form to other person.
  - Participation is graded. Feedback won't influence recipient's grade, hopefully they can use it to improve!
-

---

# Big picture: How to learn policies

---

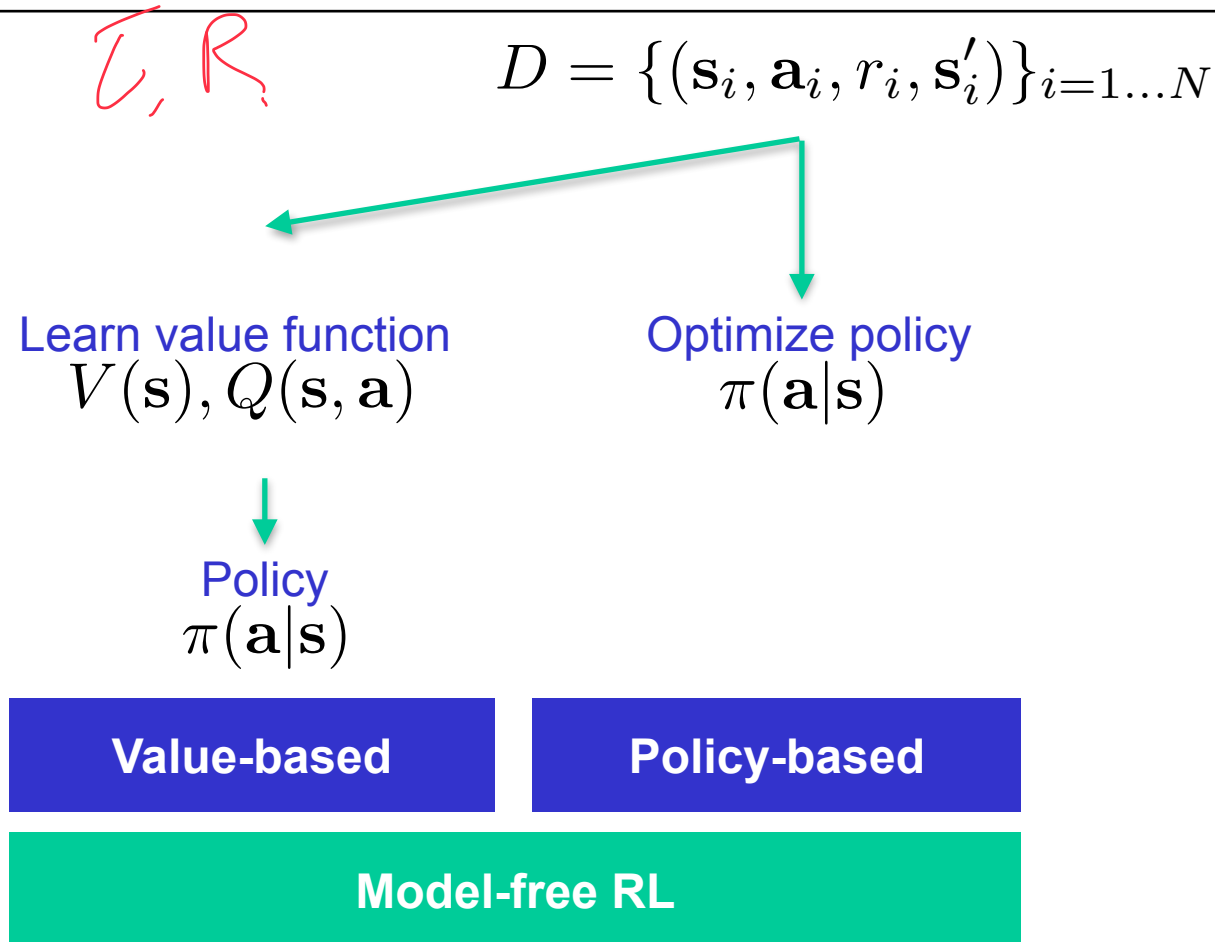
$$D = \{(s_i, \mathbf{a}_i, r_i, s'_i)\}_{i=1\dots N}$$



Policy  
 $\pi(\mathbf{a}|s)$

Thanks to Jan Peters

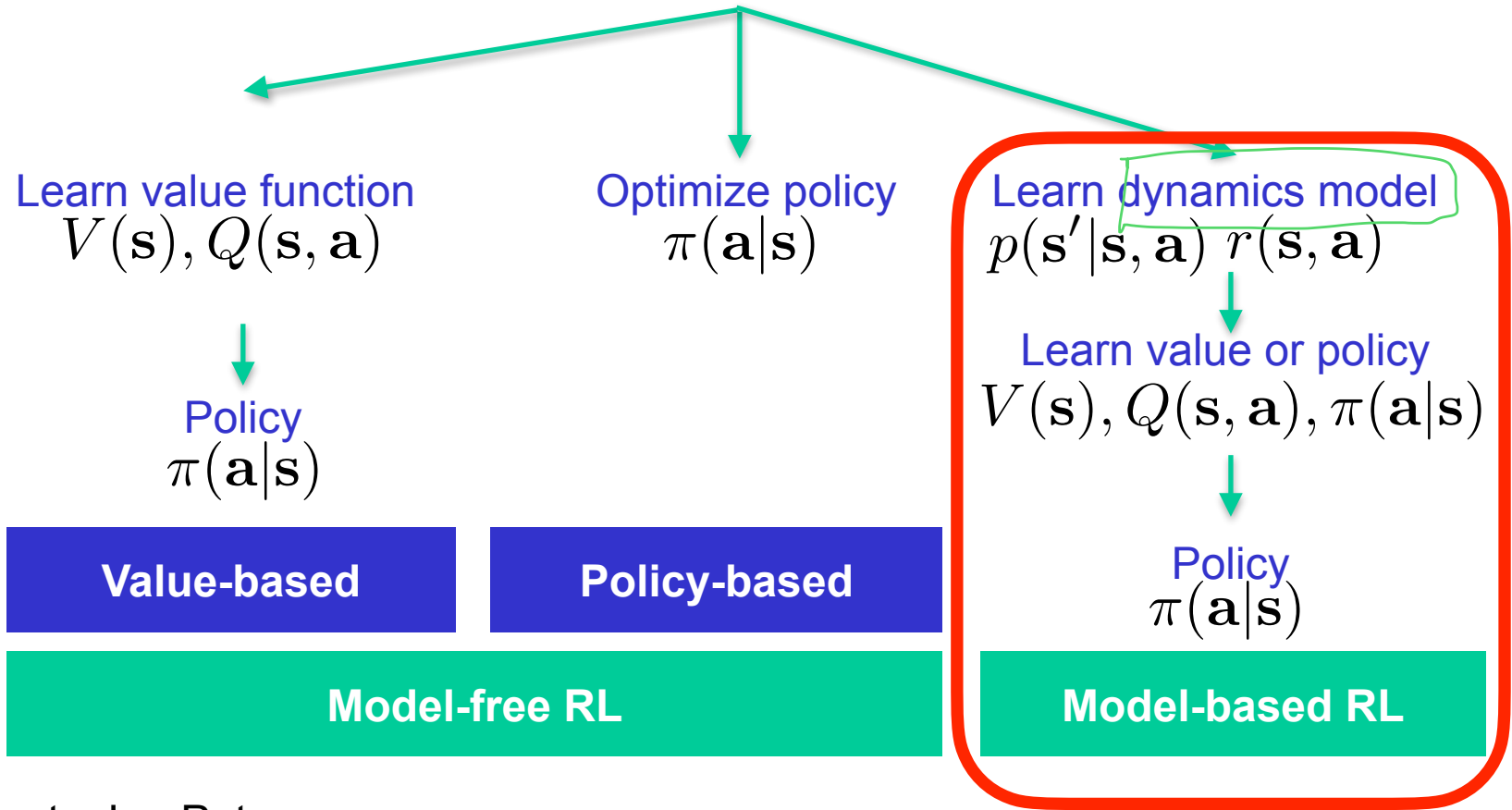
# Big picture: How to learn policies



Thanks to Jan Peters

# Big picture: How to learn policies

$$D = \{(s_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1\dots N}$$



Thanks to Jan Peters

---

# Models

---

We have seen so far:

- Planning methods, that require knowledge of the MDP (Dynamic programming: Policy-iteration, value-iteration)
- Learning methods, that directly learn the value function or policy from data (Monte-Carlo, TD methods, policy gradient, etc; Model-free)

Instead, we can try to learn a dynamics model (prediction of system dynamics) from data and use that for *planning*

In today's lecture, 'planning' is any process that uses a model rather than real data to obtain a policy

Thanks to Shimon Whiteson

# Why use dynamics models?

- lookup table  
- neural net  
(classifier/  
regression.)



Dynamics models have two main benefits:

- Generate proxy data to replace real system data  
Useful if real data is limited, time-consuming or expensive to obtain  
Physical systems (e.g. real robots), interaction with humans, time-intensive physics simulation, ...
- Provide access to things like the probability distribution that even with cheap and plentiful data we do not have access too.
- If these don't matter, better to use model-free techniques



---

# Types of systems and system models

---

A **full or distribution model** is a complete description of the transition probabilities and rewards (*full access simulator*)

A **sample or generative model** can be queried to **produce samples  $r$  and  $s'$**  from any given  $s$  and  $a$  (*e.g. black-box sim*)

A **trajectory or simulation model** can simulate an episode, but not jump to an arbitrary state (*e.g. physical model*)

Which of these is the most general?

How can it be used to generate the other types of data?

<https://webcolleges.uva.nl/Mediasite/Play/bdafdf3c5672446990039ae5aef19aae1d?catalog=ebad9ef95f7d416da0e903846162320721?playFrom=4377000&duration=1246000>

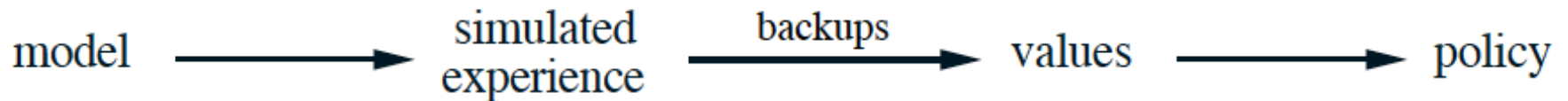
the most general model is full or dist model. because i have full knowledge on what is prob to transit from  $s$  to  $s'$ . i can use this dynamic model to generate episode.

---

# Model-based planning

---

General structure:



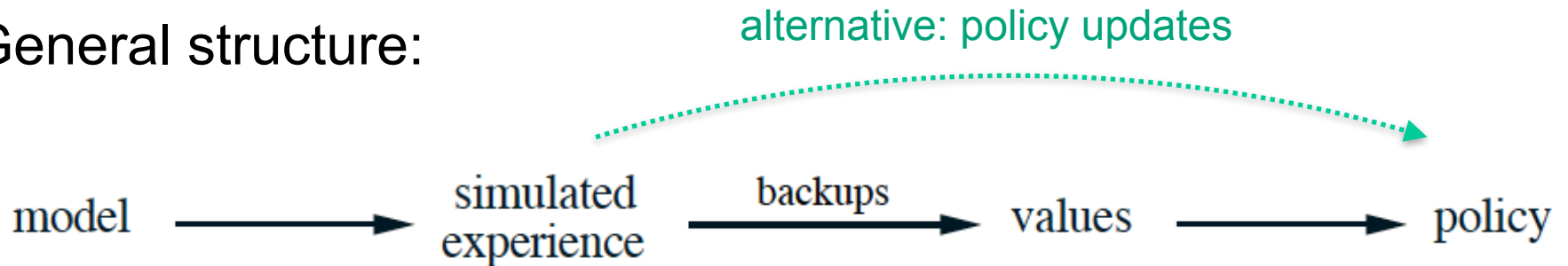
Q-planning:

Do forever:

1. Select a state,  $s \in \mathcal{S}$ , and an action,  $a \in \mathcal{A}(s)$ , at random
2. Send  $s, a$  to a sample model, and obtain  
a sample next state,  $s'$ , and a sample next reward,  $r$
3. Apply one-step tabular Q-learning to  $s, a, s', r$ :  
$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

# Model-based planning

General structure:



Q-planning:

Do forever:

1. Select a state,  $s \in \mathcal{S}$ , and an action,  $a \in \mathcal{A}(s)$ , at random
2. Send  $s, a$  to a sample model, and obtain  
a sample next state,  $s'$ , and a sample next reward,  $r$
3. Apply one-step tabular Q-learning to  $s, a, s', r$ :  
$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

2 diff ways to update value or update policy:

1) model free. direct RL

step1 use policy to generate real data (=experience),

step 2 use model free RL to update value/policy.

note: we don't use transition model at all.

2) model based. indirect RL

step 1 use policy to generate data (=experience)

step 2 use real data to estimate the transition model

step 3 use transition model to generate simulated data, then use simulated data to update value/policy

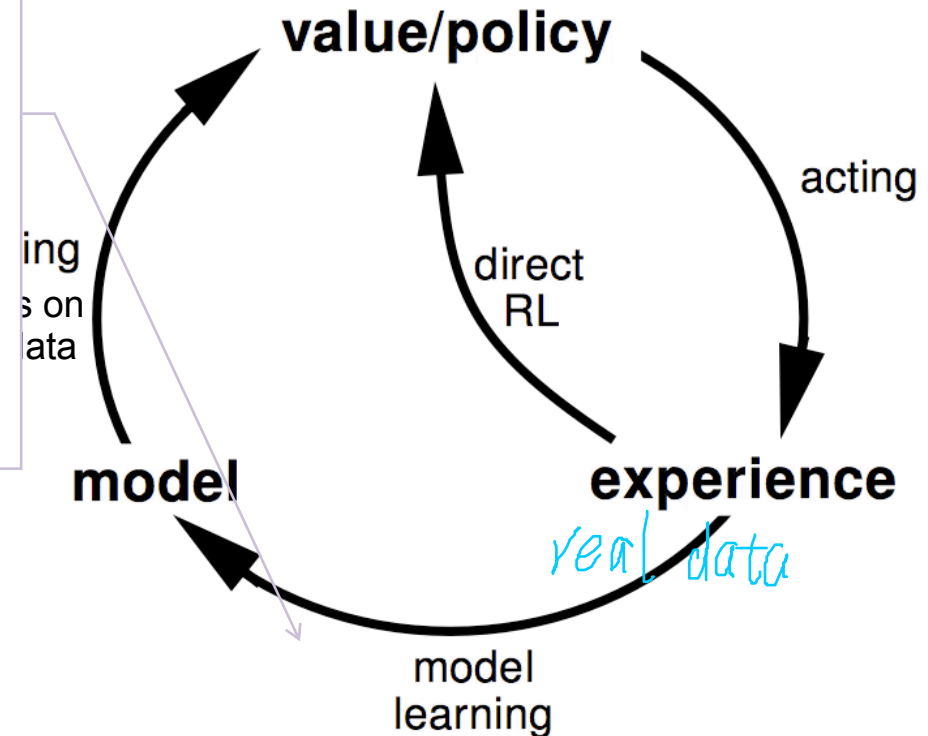
planning = use simulated data from estimated transition model to learn value/policy

note: model based RL is influenced by error of transition model. if transition model is wrongly

estimated, means it does not represent real data, then we will get wrong value/policy

simpler and not affected by modelling errors

# and Acting



Can also be combined

Thanks to Shimon Whiteson; Figure from Sutton and Barto RL:AI

# Planning, Learning, and Acting

Many ways to implement these steps

Let's start by looking at a simple way to implement each step, in a method called **Dyna**

dyna means this model needs to first generate a dynamic model, then use this estimated dynamic model to optimize value or policy

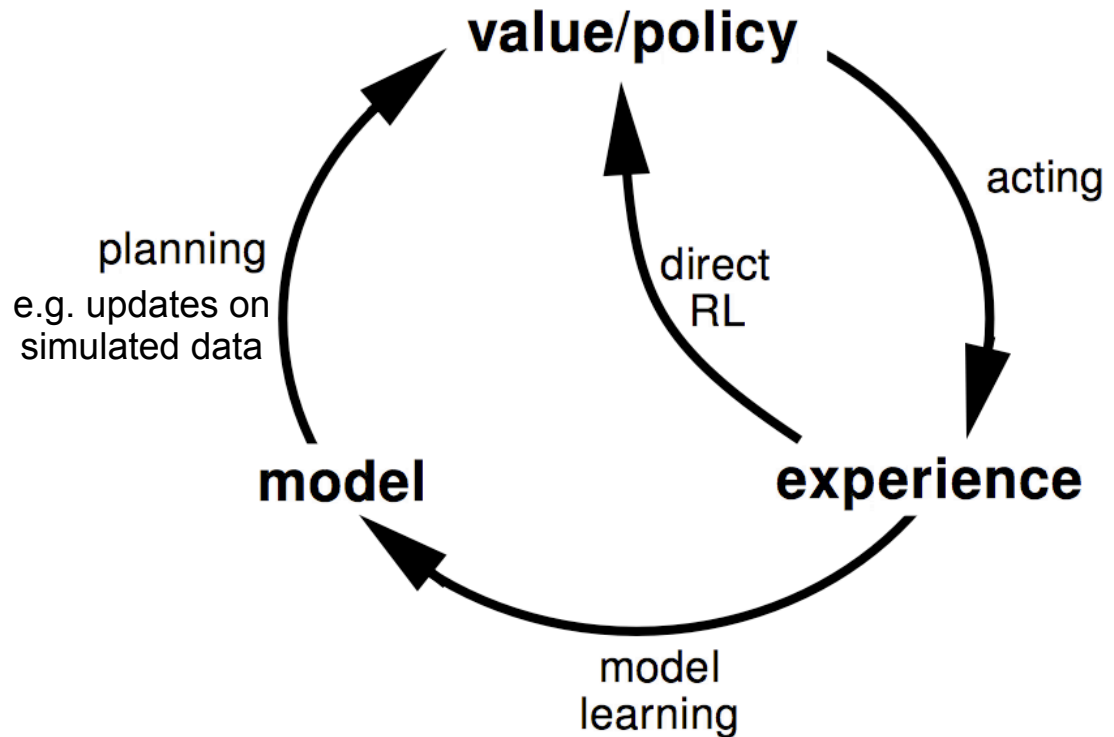


Figure from Sutton and Barto RL:AI

# Dyna-Q

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 reward,  $r$

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

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

(f) Repeat  $N$  times:

$s \leftarrow$  random previously

$a \leftarrow$  random action previously

$s', r \leftarrow Model(s, a)$

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

simulated data :  $s, a, s', r$

we pick a random  $s$  from the states that we have seen before in the real data,

we pick a random  $a$  from the actions that we have seen before in the real data,

we check our transition model table, to find out what is reward and next state then we use simulated  $s, a, r, s'$  to update  $Q$

From Sutton and Barto RL:AI

# Dyna-Q

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  $s$

$s', r \leftarrow Model(s, a)$

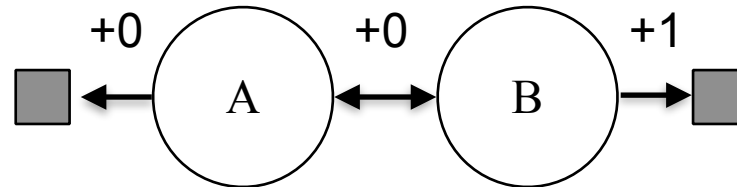
$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

Similar to  
experience  
replay!

From Sutton and Barto RL:AI

# Dyna-Q environment model example

Consider the following simple MDP:



The agent collects experiences during trajectories, e.g.:

$$\tau = [A, \text{right}, 0, B, \text{right}, +1]$$

Experiences can be summarised in environment model, e.g.:

<i><b>action \ state</b></i>	<i><b>A</b></i>	<i><b>B</b></i>
<i><b>right</b></i>	<i><b>B, 0</b></i>	<i><b>G*, +1</b></i>
<i><b>left</b></i>	<i><b>**</b></i>	<i><b>**</b></i>

\*G indicates goal / terminal state

\*\* Unvisited states are undefined



# Dyna-Q

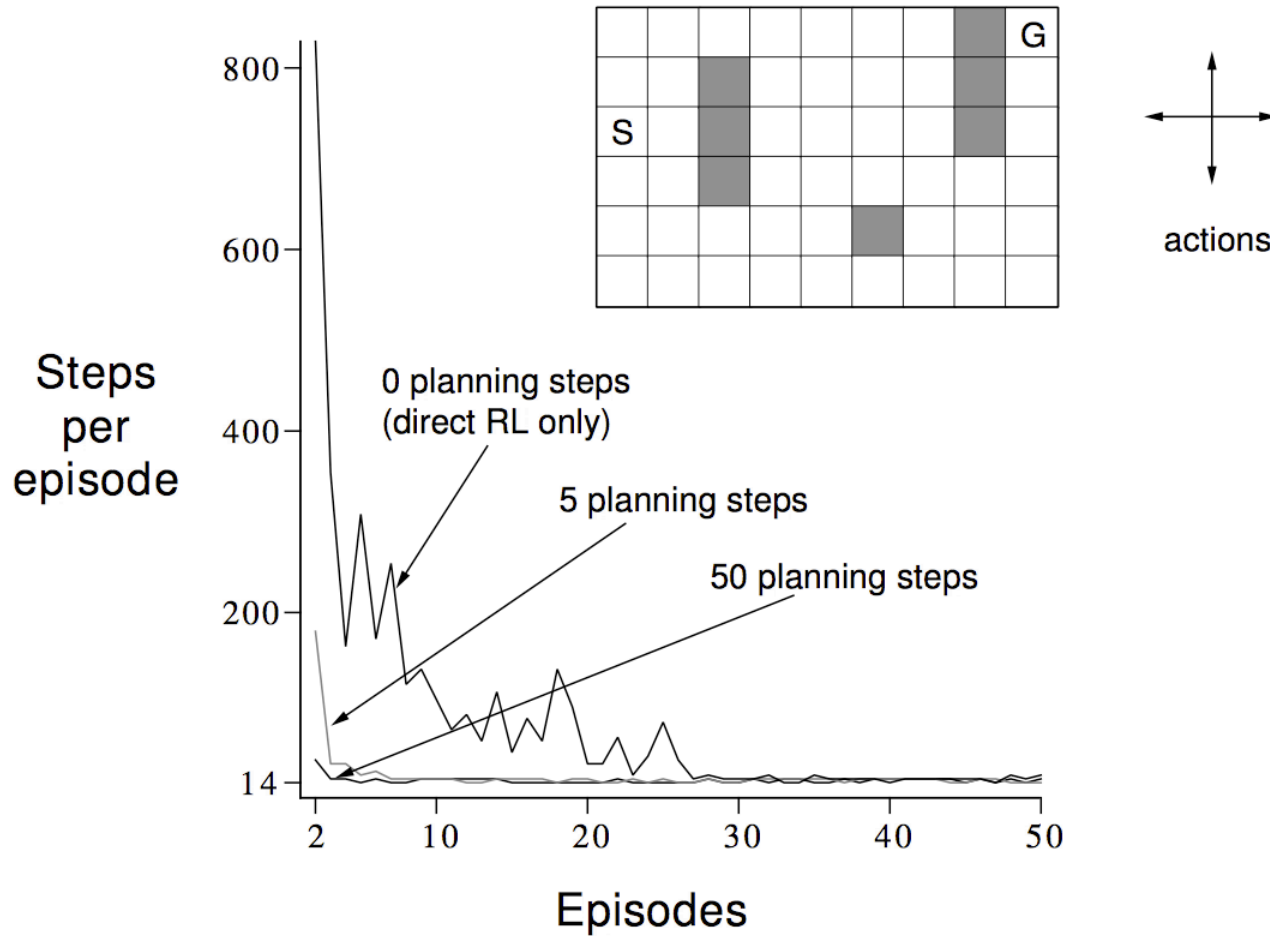
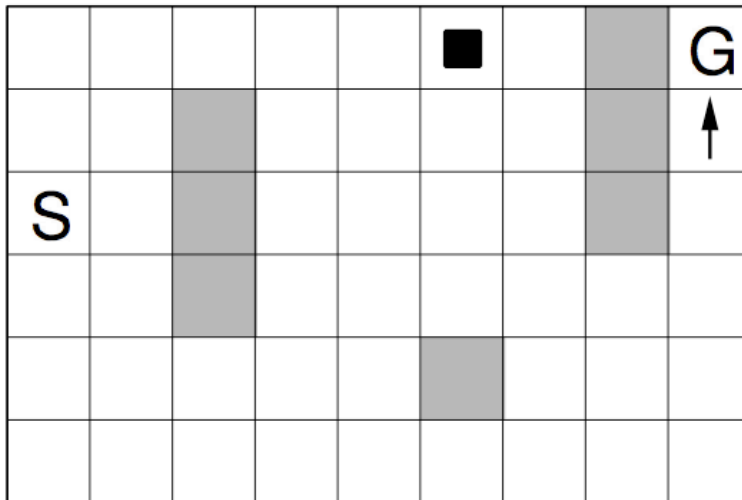


Figure from Sutton and Barto RL:AI

# Dyna-Q

During the 2nd real episode

WITHOUT PLANNING ( $N=0$ )



WITH PLANNING ( $N=50$ )

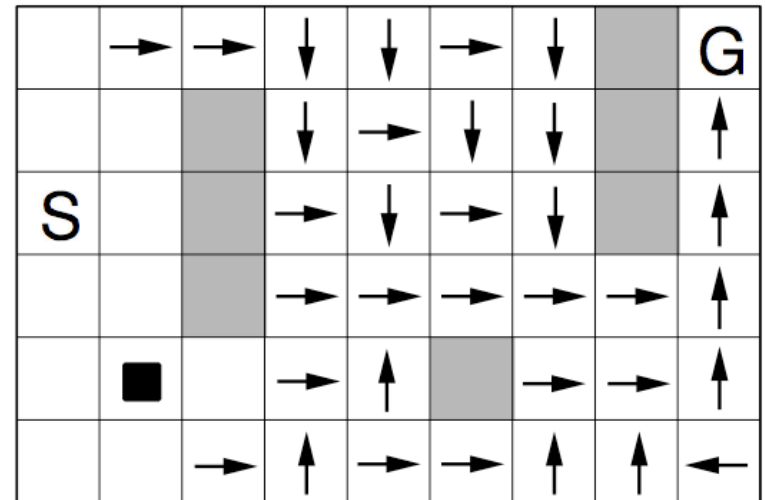


Figure from Sutton and Barto RL:AI

---

# Dyna-Q

---

Note that the comparisons look at Q-learning and Dyna-Q for equal number of **real** experience

Model-based RL typically takes more compute time for the same amount of real experience

- Any time used to learn the model
- Any time used to update the policy using simulated samples

If real samples are expensive, model-based usually better

In terms of #updates, there is no consistent winner

# Dyna-Q

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( $s, Q$ ) **How to learn model?**
- (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: **When to plan?**

$s \leftarrow$  random previously observed state

**What to update?**

$a \leftarrow$  random action previously taken in  $s$

$s', r \leftarrow Model(s, a)$

**How to update?**

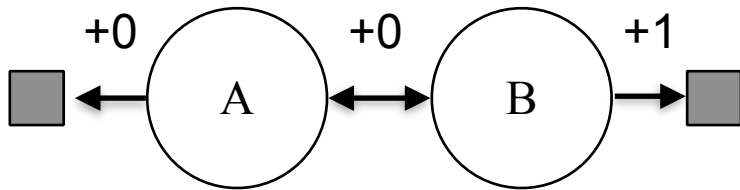
$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

From Sutton and Barto RL:AI

# How to learn the model

Dyna just stores the observed resulting state  $s'$  in a big table

Consider the following simple MDP:



The agent collects experiences during trajectories, e.g.:

$$\tau = [A, \text{right}, 0, B, \text{right}, +1]$$

Experiences can be summarised in environment model, e.g.:

<i>action \ state</i>	<i>A</i>	<i>B</i>
<i>right</i>	<i>B, 0</i>	<i>G*, +1</i>
<i>left</i>	<i>**</i>	<i>**</i>

\*G indicates goal / terminal state


\*\* Unvisited states are undefined

Assumes: transitions are deterministic

# How to learn model

Dealing with stochastic transitions:  
count how often each transition occurs

	$s' = 1$	$s' = 2$
state 1, action 1	0	0
state 2, action 1	0	0
state 1, action 2	0	0
state 2, action 2	0	0



$$D = \{(s_i, \mathbf{a}_i, r_i, s'_i)\}_{i=1\dots N} = \{(1, 2, 1, 2)\}$$

---

# How to learn model

---

Dealing with stochastic transitions:  
count how often each transition occurs

	$s' = 1$	$s' = 2$
state 1, action 1	1	1
state 2, action 1	3	1
state 1, action 2	2	3
state 2, action 2	1	0

$$D = \{(s_i, \mathbf{a}_i, r_i, s'_i)\}_{i=1\dots N} = \{(1, 2, 1, 2), \dots\}$$

Calculate the maximum likelihood model

$$\hat{p}_{ss'}^a = \frac{n_{ss'}^a}{n_s^a}$$

---

# How to learn model

---

Dealing with stochastic transitions:  
count how often each transition occurs

	<b>s' = 1</b>	<b>s' = 2</b>
<b>state 1, action 1</b>	<b>1 (50%)</b>	<b>1 (50%)</b>
<b>state 2, action 1</b>	<b>3 (75%)</b>	<b>1 (25%)</b>
<b>state 1, action 2</b>	<b>2 (40%)</b>	<b>3 (60%)</b>
<b>state 2, action 2</b>	<b>1 (100%)</b>	<b>0 (0%)</b>

$$D = \{(s_i, \mathbf{a}_i, r_i, s'_i)\}_{i=1\dots N} = \{(1, 2, 1, 2), \dots\}$$

Calculate the maximum likelihood model

$$\hat{p}_{ss'}^a = \frac{n_{ss'}^a}{n_s^a}$$



---

# What to update

---

Dyna-Q picks random previously observed states and actions

Simple and practical: can always make a prediction

But we might spend a lot of updates on  $(s,a)$  pairs that

- Don't require an update
- Are not relevant for the optimal policy

---

# What to update

---

Dyna-Q picks random previously observed states and actions

Simple and practical: can always make a prediction

But we might spend a lot of updates on  $(s,a)$  pairs that

- Don't require an update
- Are not relevant for the optimal policy

Also, we can only do this for certain models...

---

# Reminder: types of models

---

A **full** or **distribution** model is a complete description of the transition probabilities and rewards (*full access simulator*)

A **sample** or **generative** model can be queried to produce samples  $r$  and  $s'$  from any given  $s$  and  $a$  (*e.g. black-box sim*)

A **trajectory** or **simulation** model can simulate an episode, but not jump to an arbitrary state (*e.g. physical model*)

Which of these do we need for Dyna-Q?

Thanks to Shimon Whiteson

---

---

# What to update

---

Only update  $(s,a)$  that require an update?

---

# What to update

---

Only update  $(s,a)$  that require an update?

Prioritised sweeping

- Work backward from states whose values changed
- Maintain a queue of states that should be updated as a result
- Sort queue by priority: amount of change of target  $Q(s')$
- Update states in order of the queue

# What to update?

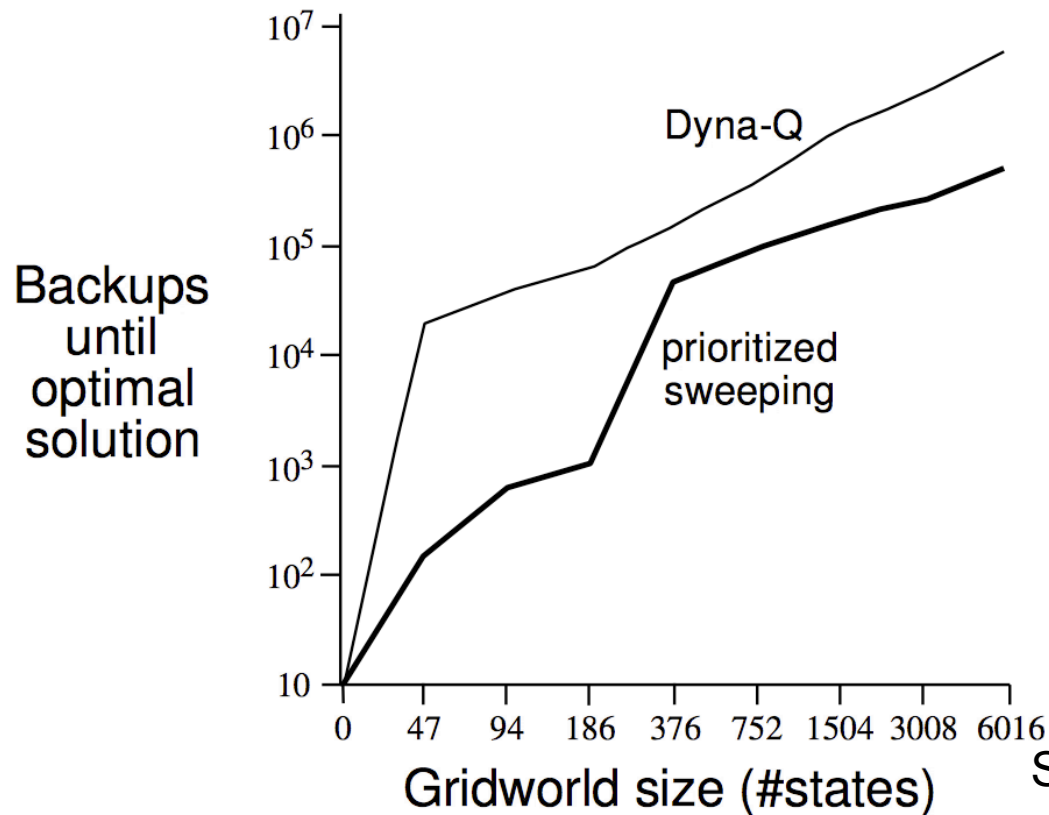


Figure from  
Sutton and Barto RL:AI

---

# What to update

---

Alternative: what is relevant for policy?

---

# What to update

---

Alternative: what is relevant for policy?

Want accurate values for states often visited by current policy

We might not care for states that are rarely visited

So: update  $(s,a)$  from on-policy distribution. Easiest to get such  $(s,a)$  by just simulating episodes



---

# What to update

---

Alternative: what is relevant for policy?

Want accurate values for states often visited by current policy

We might not care for states that are rarely visited

So: update  $(s,a)$  from on-policy distribution. Easiest to get such  $(s,a)$  by just simulating episodes

**full /  
distributional?**

**sample /  
generative?**

**simulation /  
episodic?**

# What to update

$b$  = branching factor

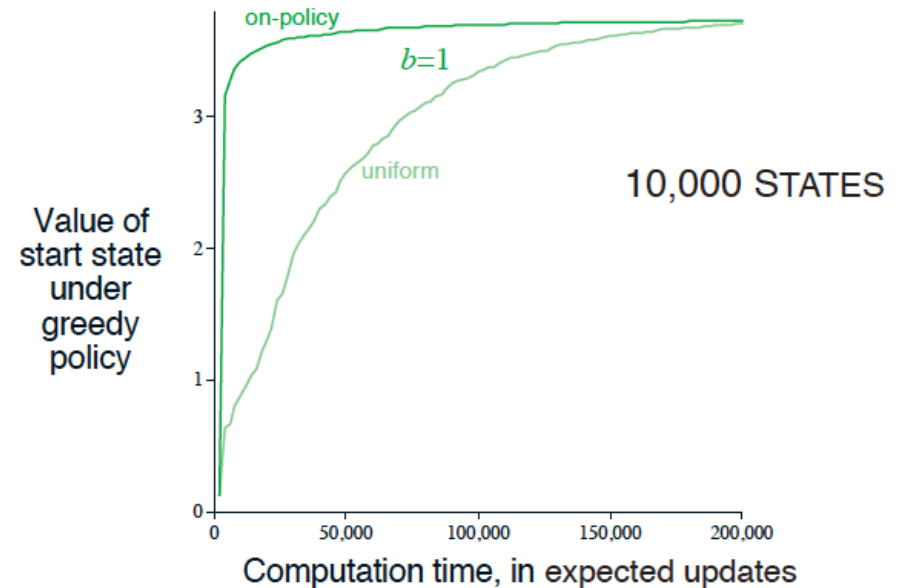
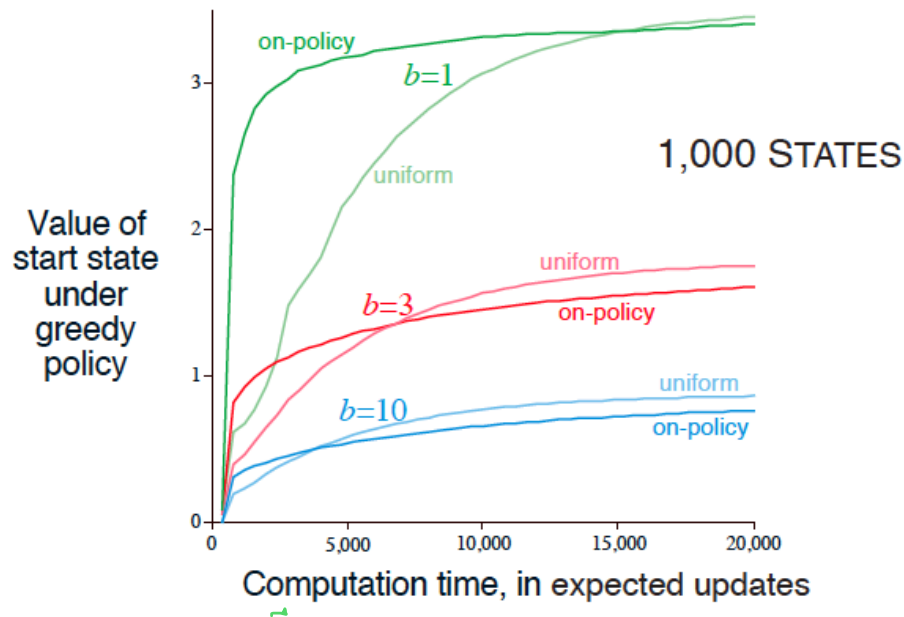


Figure from Sutton and Barto RL:AI

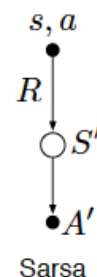
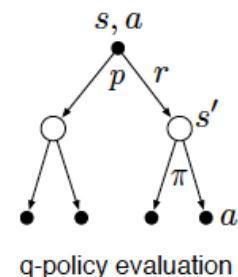
# How to update

We could consider *expected updates* (like DP) where we average over states rather than sample.

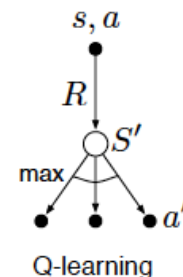
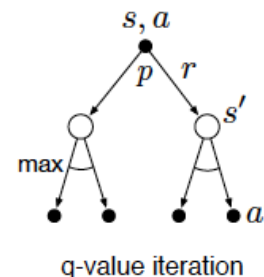
Knowledge of MDP needed, but model can be substituted

Expected update is exact, sample update depends on random process. Is expected update always better?

$$q_{\pi}(s, a)$$

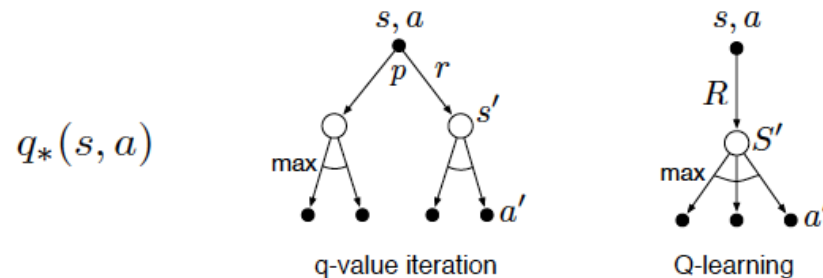


$$q_{*}(s, a)$$



# How to update

Expected update always need value from all possible next states! With samples we get close with fewer compute!



Compare number of 'max' operations!

Figure from Sutton and Barto RL:AI

# How to update

Expected update always need value from all possible next states! With samples we get close with fewer compute!

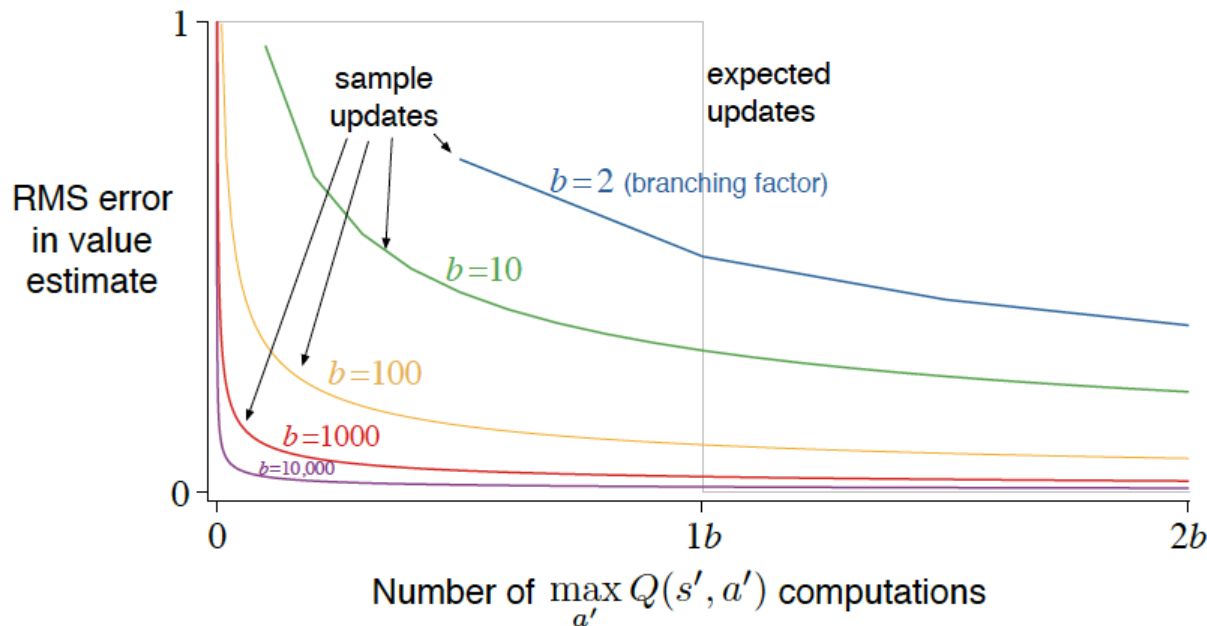


Figure from Sutton and Barto RL:AI

# How to update

Expected update always need value from all possible next states! With samples we get close with fewer compute!

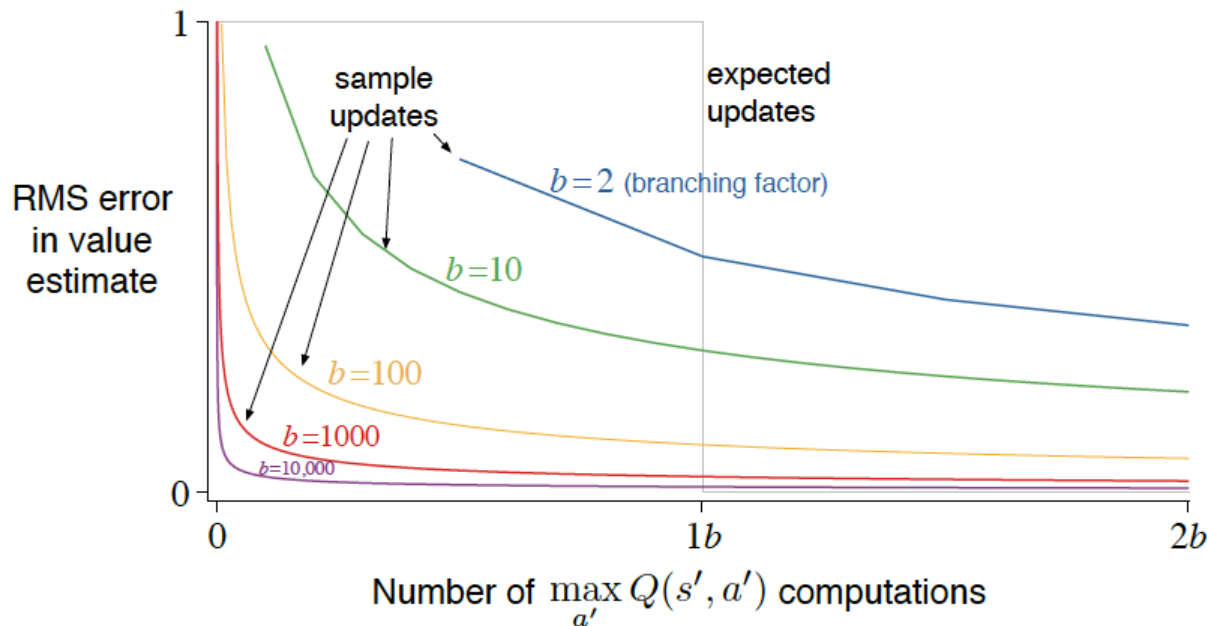
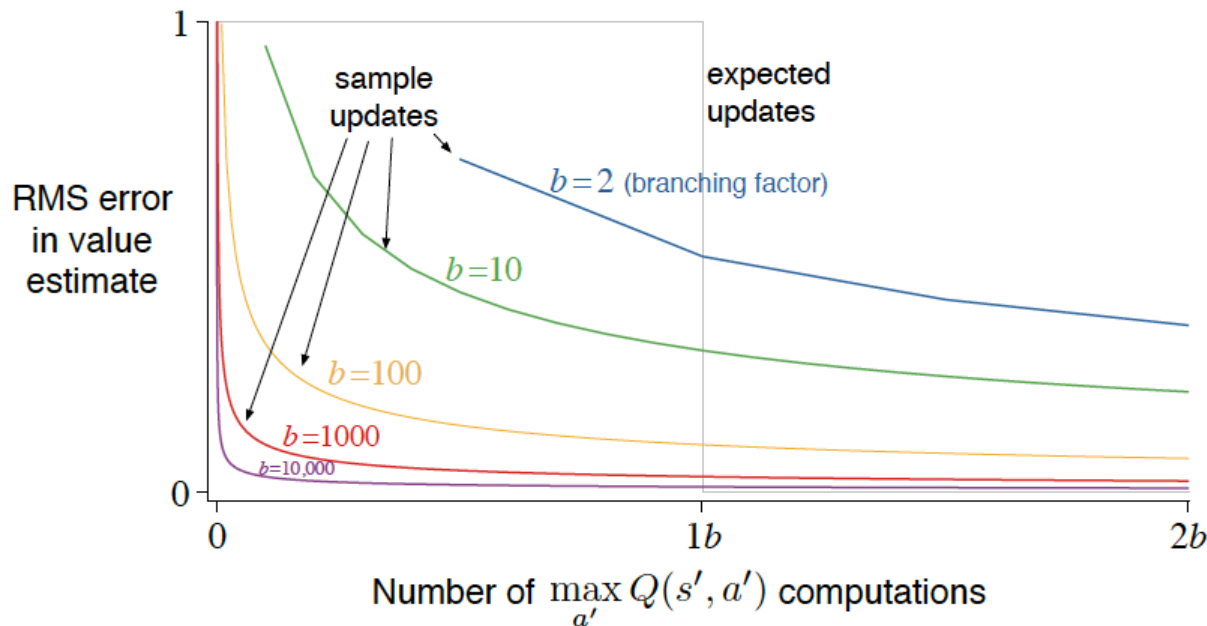


Figure from Sutton and Barto RL:AI

# How to update

Expected update always need value from all possible next states! With samples we get close with fewer compute!



**full /  
distributional?**

**sample /  
generative?**

**simulation /  
episodic?**

---

# When to plan

---

Dyna-Q uses the model to learn a good policy for any state. We can think of doing this ‘ahead of’ acting in the world

We can also plan ahead ‘while’ acting in the world.

- Think of playing chess. You’re thinking ahead from your current position, not to solve from any position
- Look only at part of state space relevant to current state



---

# When to plan

---

---

# When to plan

---

## Planning at decision time

- Simulate (all possible) sequences for the next  $k$  time steps
- Pick the action that gives the best return (using e.g. normal q-value back-ups)
- If enough compute: simulate until end of episode or when  $\gamma^k$  small
- If long enough simulations are not possible, combine with learned Q/V fc to indicate return after  $k$  steps

---

# When to plan

---

## Planning at decision time

- Simulate (all possible) sequences for the next  $k$  time steps
- Pick the action that gives the best return (~~using e.g. normal q-value back-ups~~)
- If enough compute: simulate until end of episode or when  $\gamma^k$  small
- If long enough simulations are not possible, combine with learned Q/V fc to indicate return after  $k$  steps

Can combine decision-time planning with learned policies

- This happens in Alpha-Go!

---

# Model-based policy search

---

Instead of using the learned model to learn a value function, we can try to directly learn a **policy**

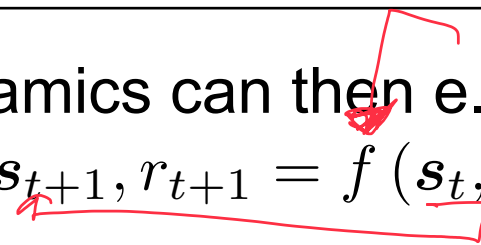
This is especially useful when actions are **continuous** (As using pure value-based methods in this case is tricky)

---

# Model-based policy search

---

Dynamics can then e.g. be written as

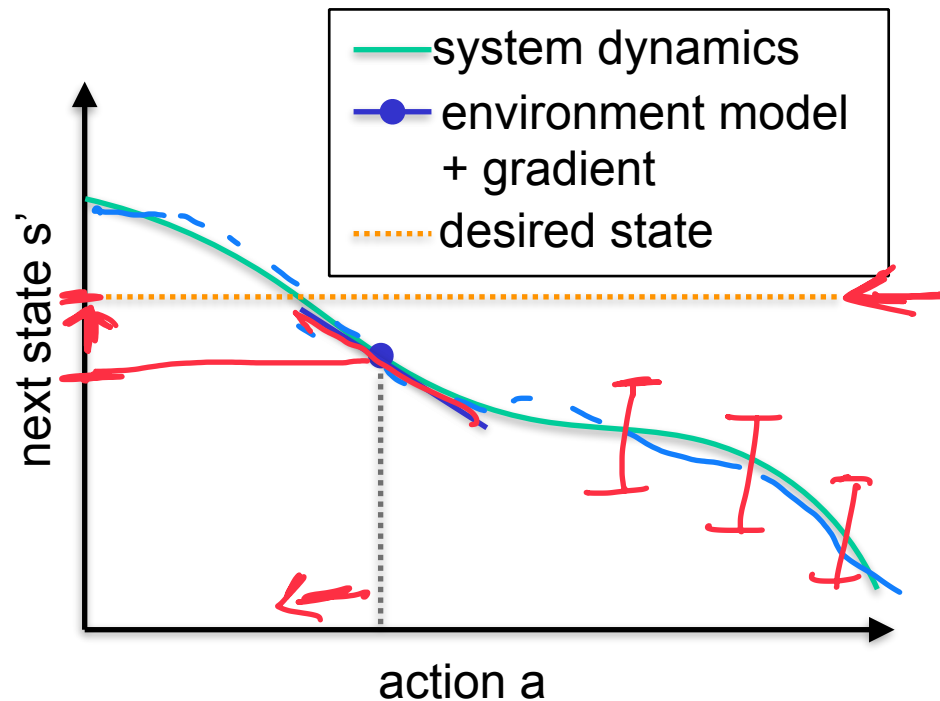

$$s_{t+1}, r_{t+1} = f(s_t, a_t), \quad s_0 \sim p(s_0)$$

Goal is to learn transition function  $f$   
(using any function approximator, e.g. a neural net)

Use  $f$  to:

1. Generate data to use with any ‘regular’ policy search method
2. Use extra information from the model, e.g. gradients
  - Only possible in model-based settings
  - Typically more efficient

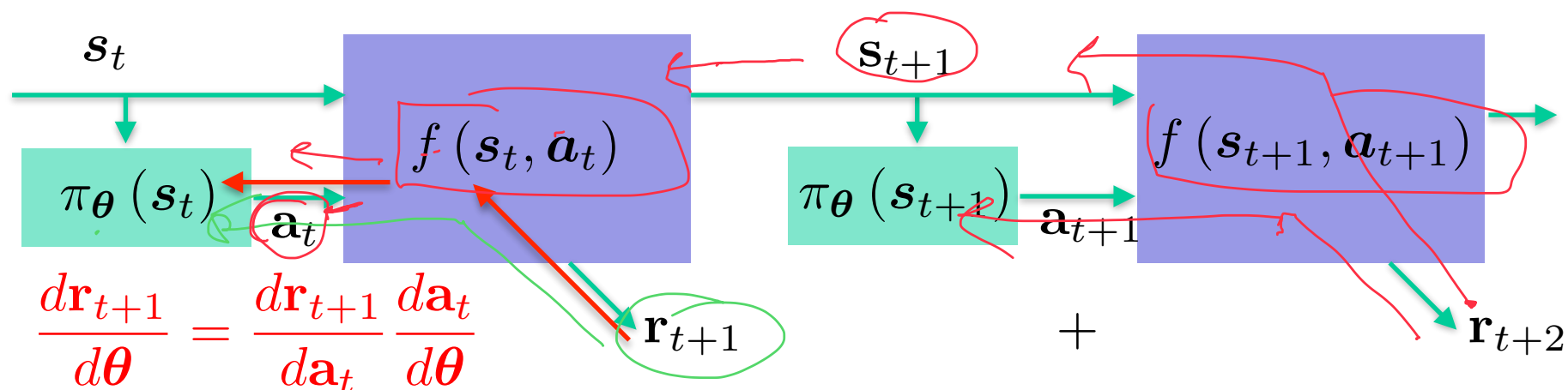
# Environment model with gradient



# Policy update strategies

Backpropagation with deterministic system and policy

$$V(s_t) = r_{t+1} + \gamma r_{t+2} + \dots$$
$$\nabla_{\theta} V(s_t) = \nabla_{\theta} r_{t+1} + \gamma \nabla_{\theta} r_{t+2} + \dots$$



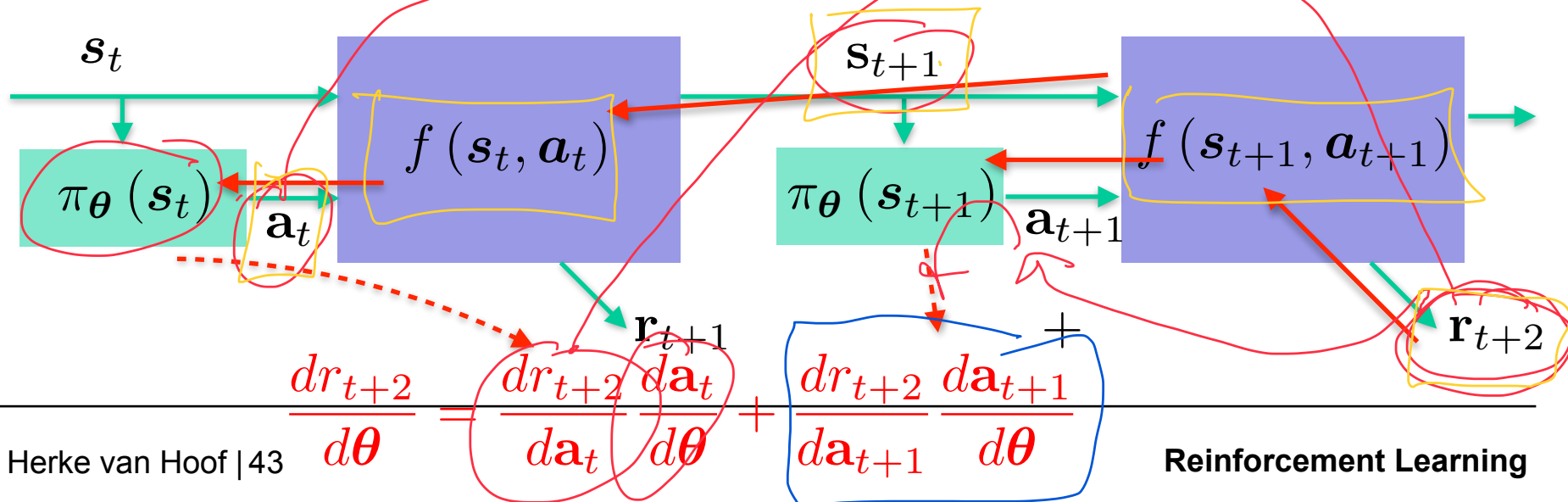
# Policy update strategies

Backpropagation with deterministic system and policy

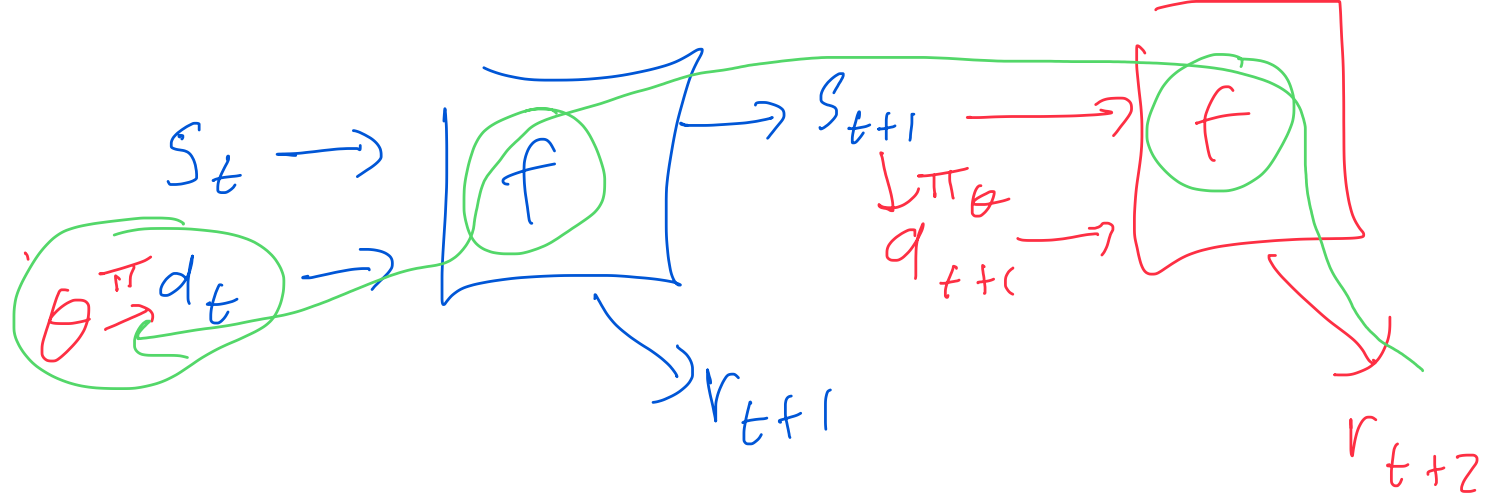
$$V(s_t) = r_{t+1} + \gamma r_{t+2} + \dots$$

$$\nabla_{\theta} V(s_t) = \nabla_{\theta} r_{t+1} + \gamma \nabla_{\theta} r_{t+2} + \dots$$

$$\frac{dr_{t+2}}{ds_{t+1}} \frac{ds_{t+1}}{da_t} \frac{da_t}{d\theta}$$







$$\frac{dr_{t+1}}{da_t}$$

$$\frac{dr_{t+2}}{ds_{t+1}} \quad \frac{ds_{t+1}}{da_t} \quad \frac{da_t}{d\theta}$$

---

# Points to remember

---

Why do model-based reinforcement learning?

Model based **value learning** vs model based **policy search**

What is the general structure of model-based learning

What are some answers to the questions:

- How to learn model
- When to update
- What to update
- How to update

---

# Thanks for your attention!

---

Feedback?

[h.c.vanhoof@uva.nl](mailto:h.c.vanhoof@uva.nl)