Model-based reinforcement learning

Herke van Hoof

Exam info

There are example exams on Canvas

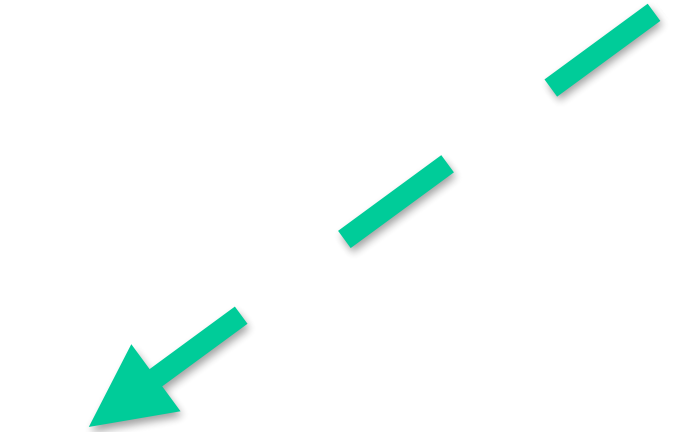
Prioritize “What you should know” or “Conclusions” provided for most lectures!

No tools (calculator). We’ll provide a sheet with the policy and value update equations. (Example in 2019 practice exam)

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

Big picture: How to learn policies

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



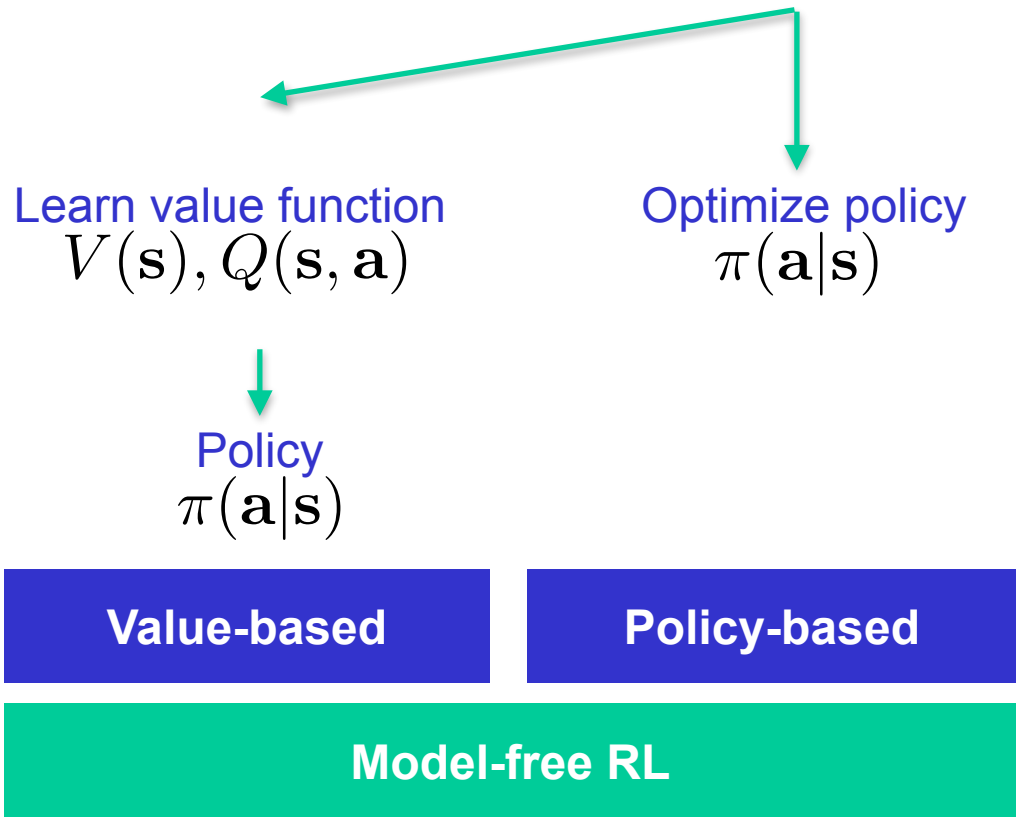
Policy

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

Thanks to Jan Peters

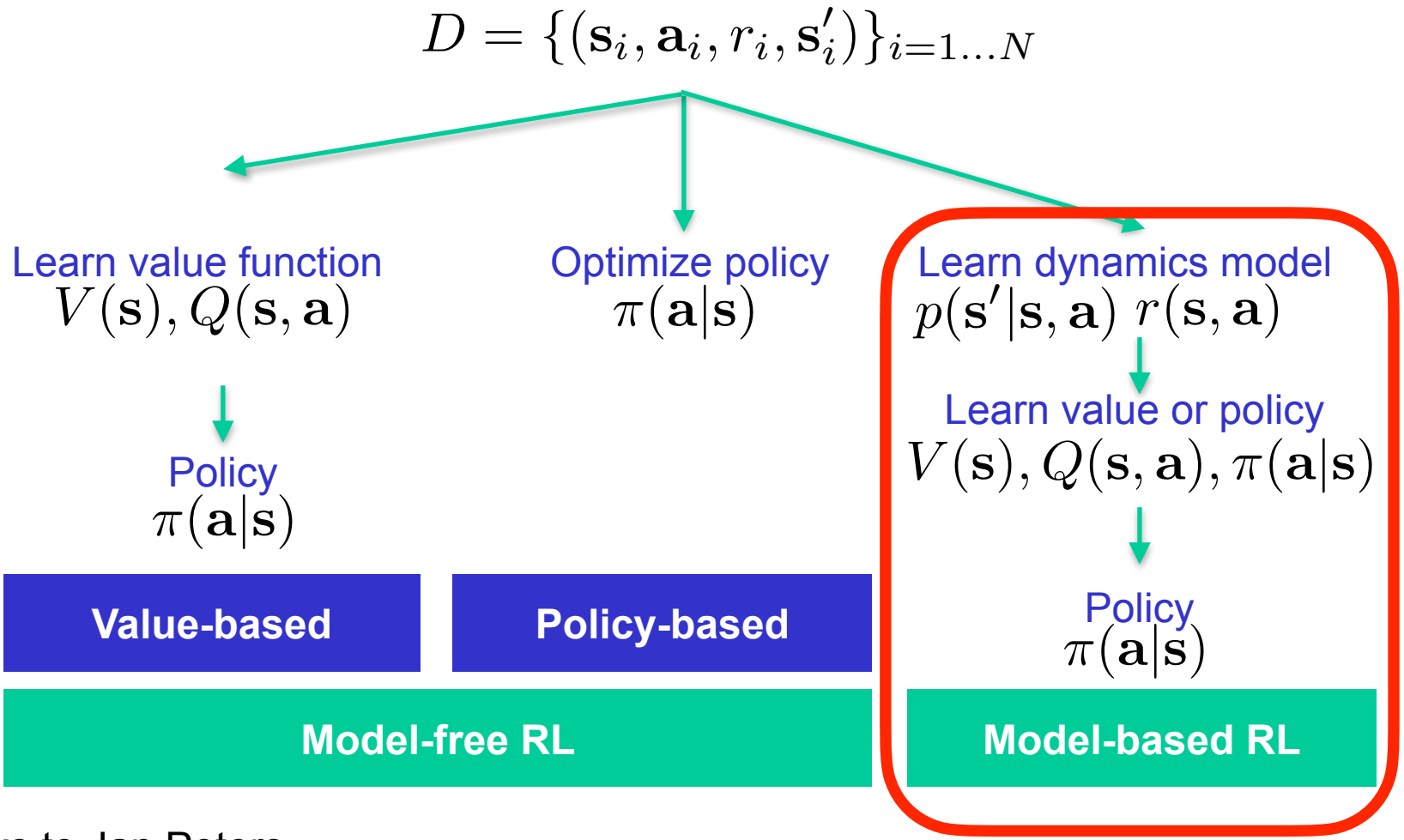
Big picture: How to learn policies

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



Thanks to Jan Peters

Big picture: How to learn policies



Thanks to Jan Peters

Models

We have seen so far:

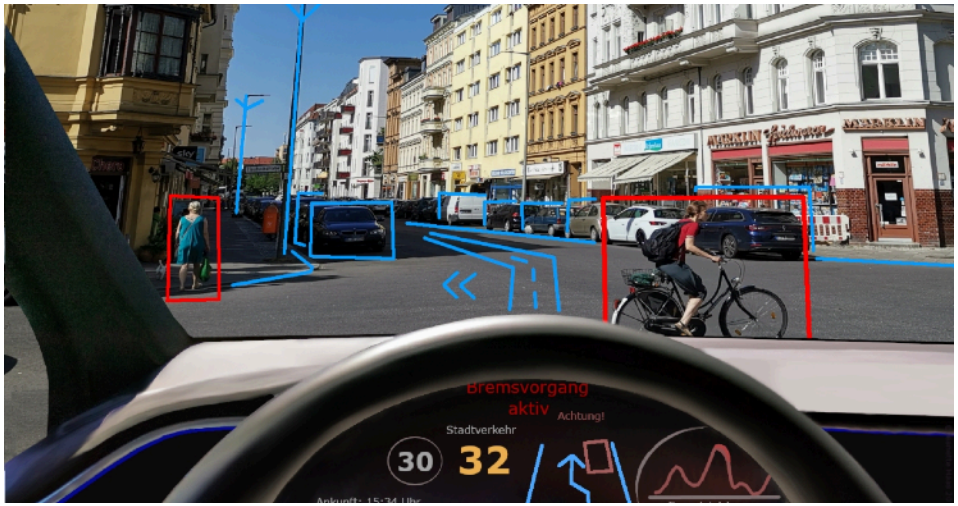
- Planning methods, that require knowledge of the MDP (Dynamic programming: Policy-iteration, value-iteration)
- Learning methods, that directly learn the value function or policy from data (Monte-Carlo, TD methods, policy gradient, etc; Model-free)

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 model based



Wikimedia/Eschenzweig

Steering angle
Gas pedal
Breaks

GPS
Camera images
Radar

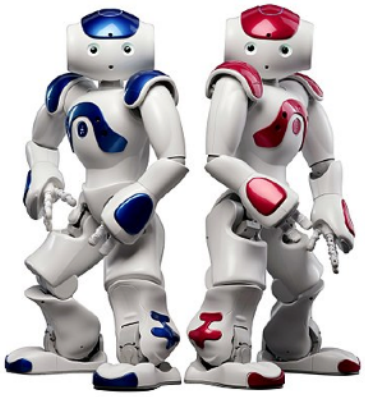
Action1	Obs1
Action2	Obs2
Action3	Obs3
…	…

Limitations of learning on real system:

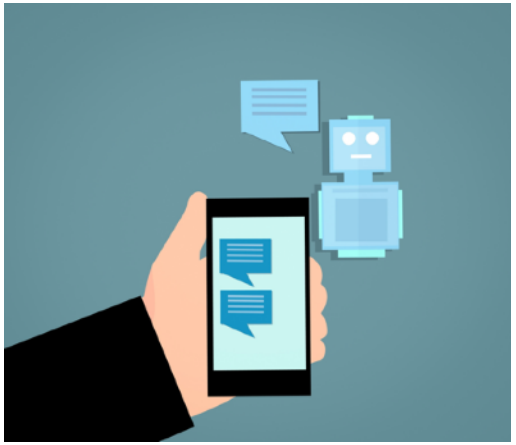
- Data expensive / risky / time-consuming
- No counterfactuals / gradients
- On-policy maybe not possible
- No access to distribution

Why model based

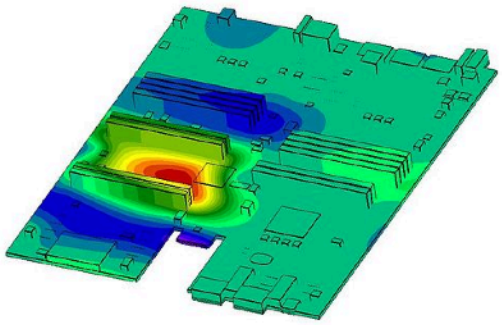
Is data expensive/difficult to obtain?



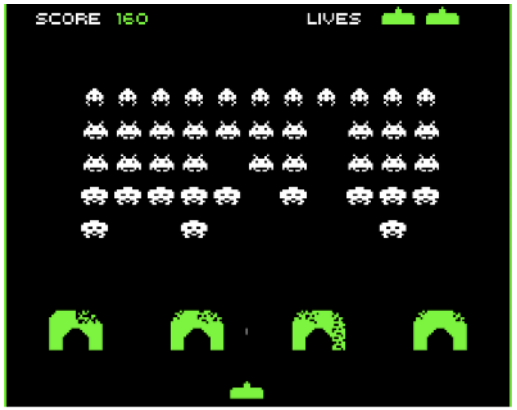
Wikimedia/softbank



Publicdomain pictures.net

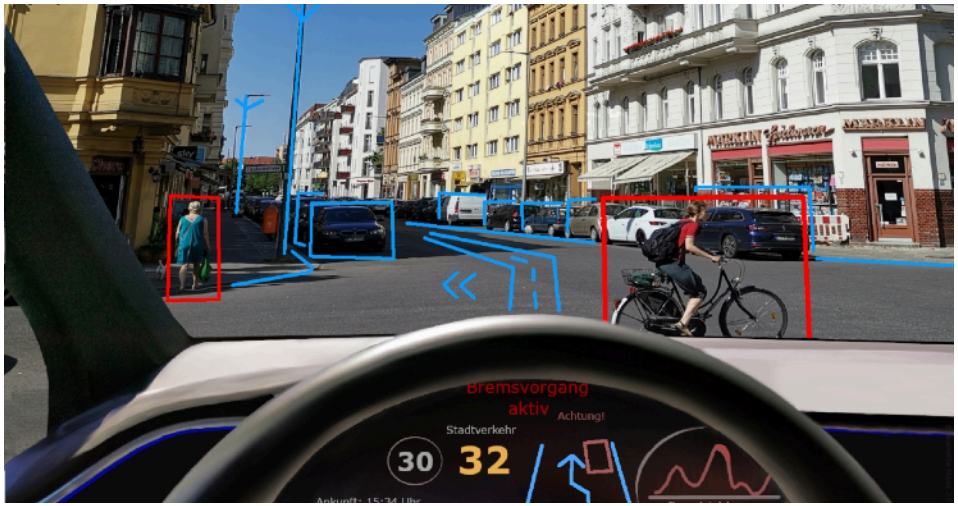


Wikimedia/MTSBoston



Smithsonianmag.com

Why model based



Wikimedia/Eschenzweig

Steering angle
Gas pedal
Breaks

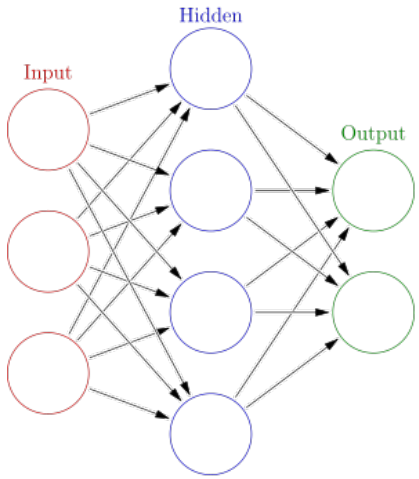
GPS
Camera images
Radar

Use a learned model to simulate the real system

Learned model can generate data

- Cheaply
- In varying circumstances
- Possibly with access to gradients / distribution

Why model based



Wikimedia/Rahem

Steering angle
Gas pedal
Breaks

GPS
Camera images
Radar

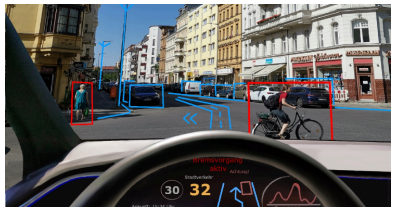
Use a learned model to simulate the real system

Learned model can generate data

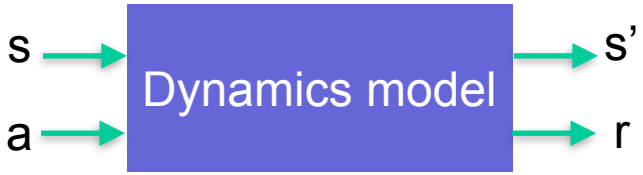
- Cheaply
- In varying circumstances
- Possibly with access to gradients / distribution

Why use dynamics models?

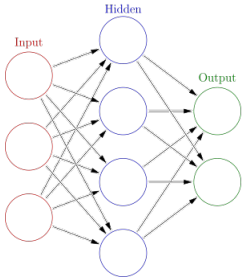
Real system



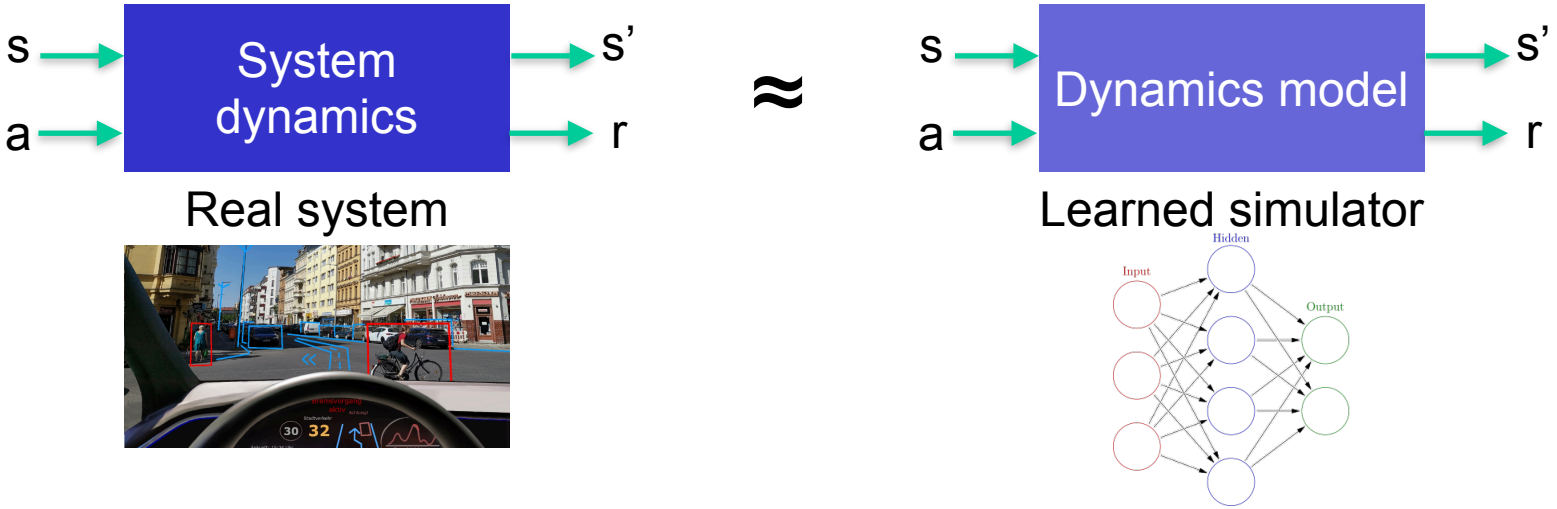
≈



Learned simulator



Why use dynamics models?



We refer to learned simulators as *dynamics models*

Such models are useful if:

- Real data is limited, time-consuming or expensive to obtain
- Need to access internal gradients, probability distribution, etc

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)*

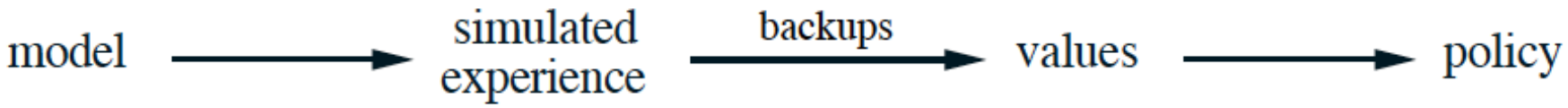
A **trajectory** or **simulation** model can simulate an episode, but not jump to an arbitrary state *(e.g. physical model)*

Which of these is the most general?
How can it be used to generate the other types of data?

Thanks to Shimon Whiteson

Model-based planning

General structure:



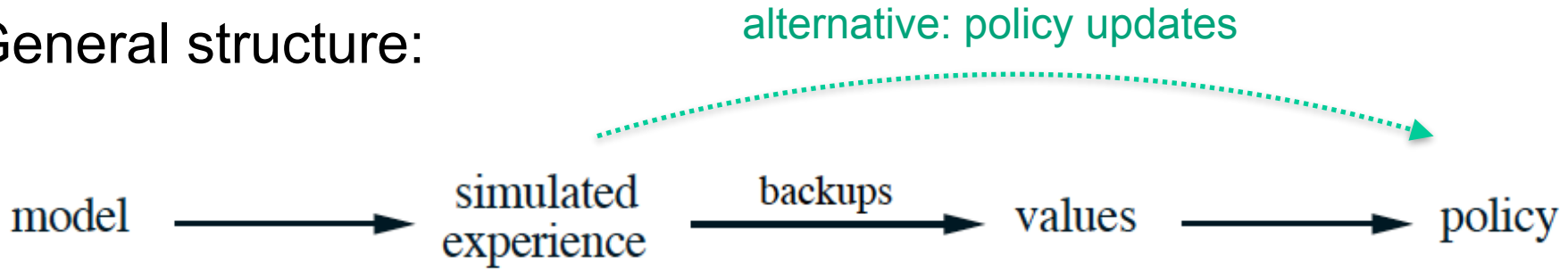
Q-planning:

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:

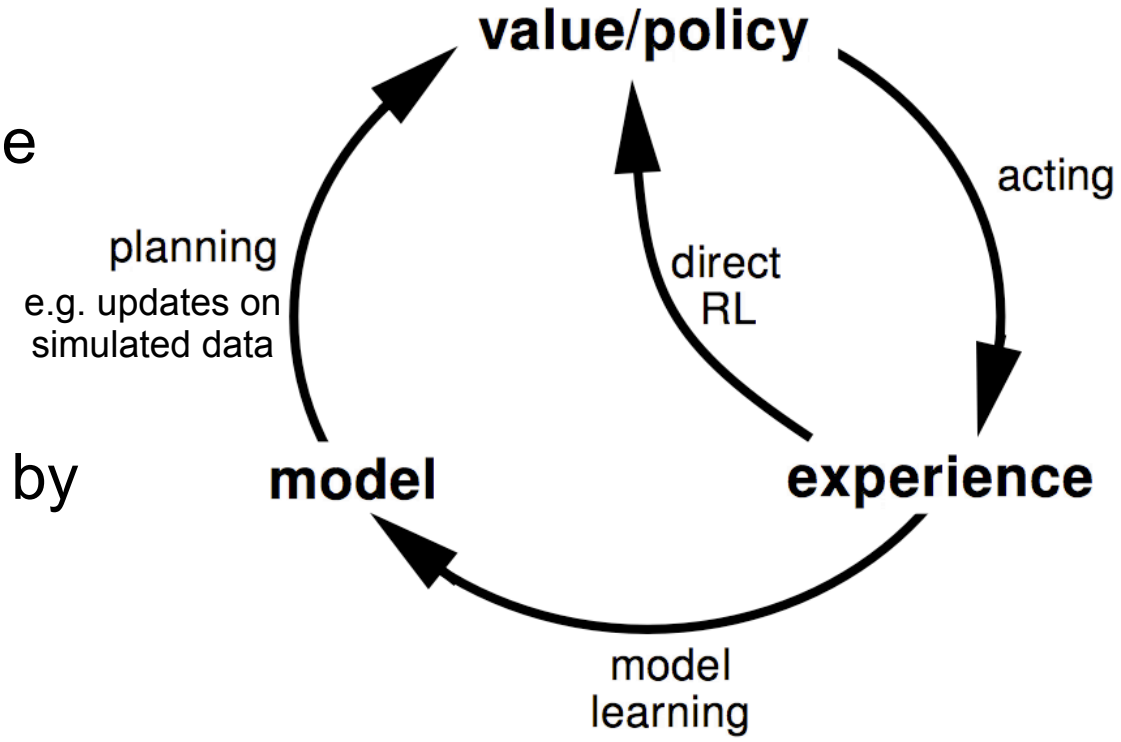
1. Select a state, $s \in \mathcal{S}$, and an action, $a \in \mathcal{A}(s)$, at random
2. Send s, a to a sample model, and obtain
   a sample next state, s', and a sample next reward, r
3. Apply one-step tabular Q-learning to s, a, s', r:
   $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')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

Planning, Learning, and Acting

Model-based methods make fuller use of experience: lower sample complexity

Model-free methods are simpler and not affected by modelling errors

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**

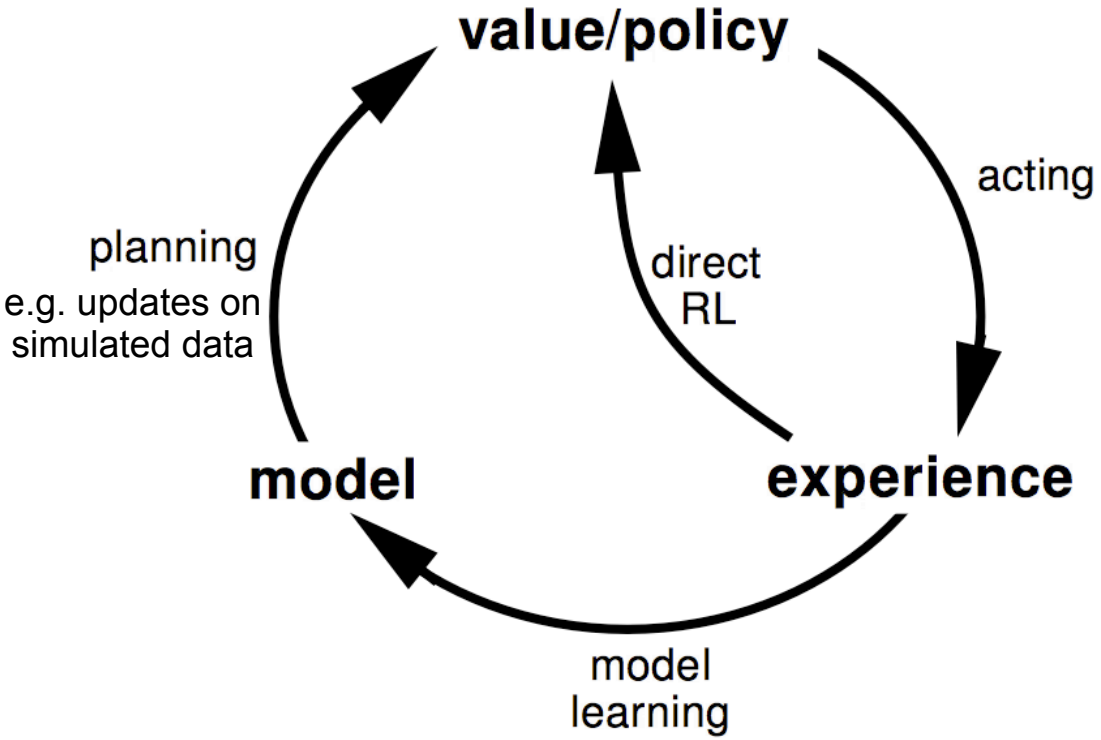


Figure from Sutton and Barto RL:AI

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 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

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

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

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

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

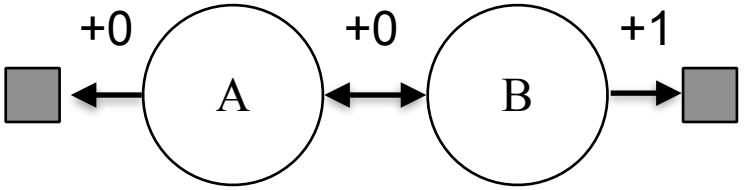
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

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.:

action \ state	*A*	*B*
right	*B, 0*	*G*, +1*
left	**	**

*G indicates goal / terminal state

** Unvisited states are undefined

Dyna-Q

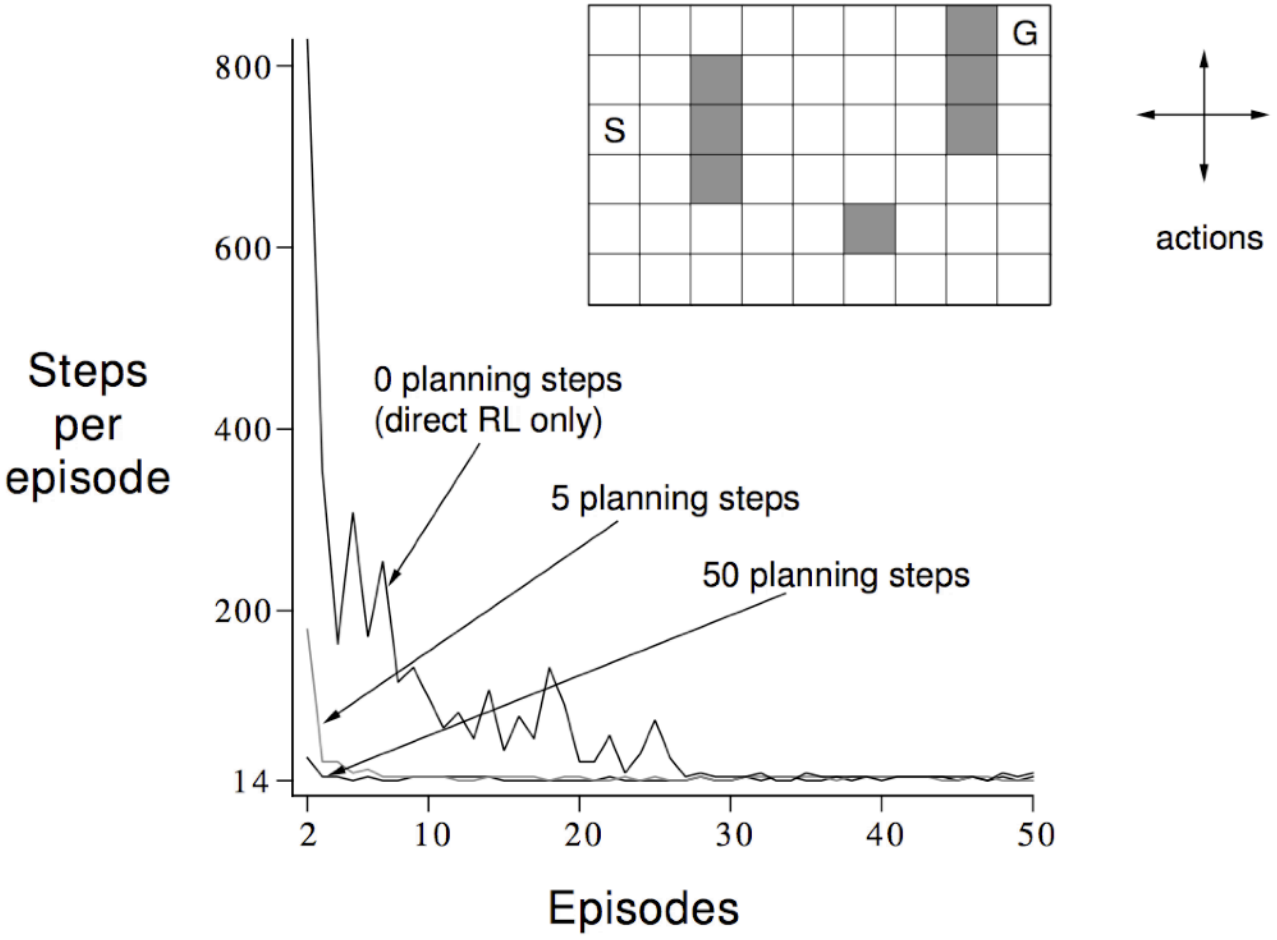


Figure from Sutton and Barto RL:AI

Dyna-Q

During the 2nd real episode

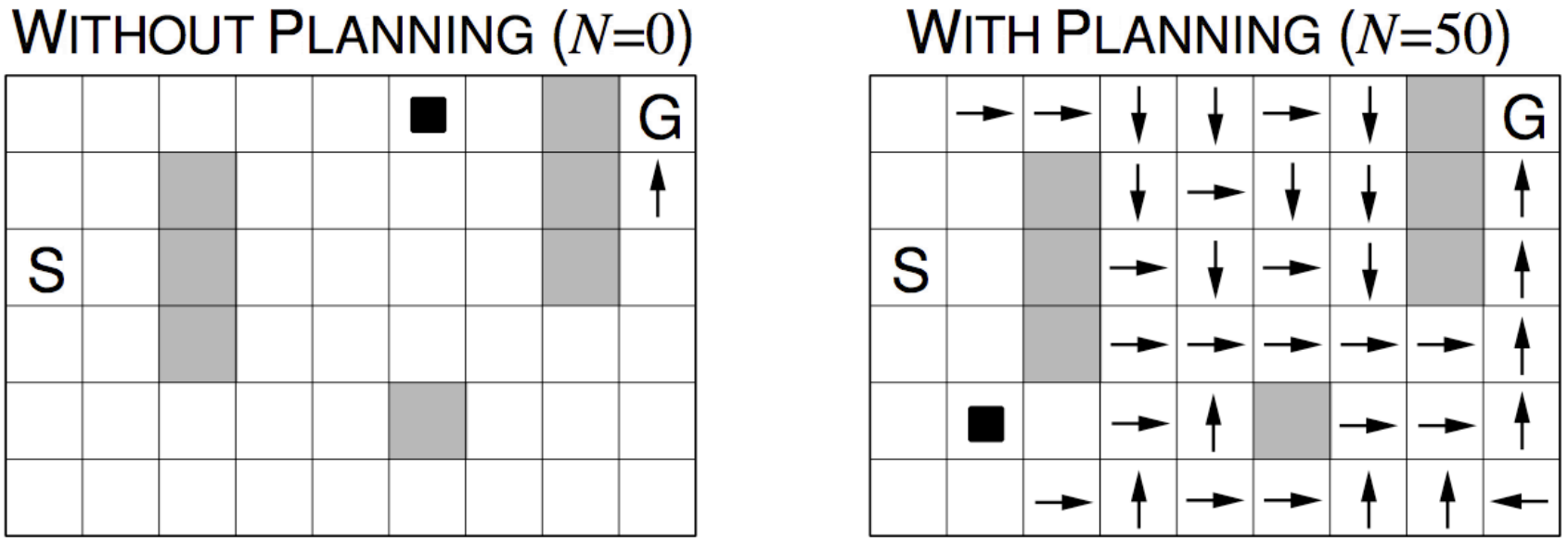


Figure from Sutton and Barto RL:AI

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?**

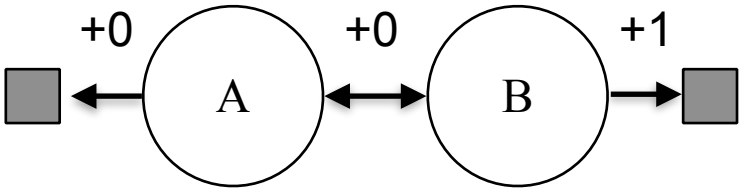
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

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.:

action \ state	*A*	*B*
right	*B, 0*	*G*, +1*
left	**	**

*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 → 1
state 2, action 2	0	0

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

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

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

Calculate the maximum likelihood model

$$\hat{P}^a_{ss'} = \frac{n^a_{ss'}}{n^a_s}$$

How to learn model

Dealing with stochastic transitions:
count how often each transition occurs

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

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

Calculate the maximum likelihood model

$$\hat{P}^a_{ss'} = \frac{n^a_{ss'}}{n^a_s}$$

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

Prioritised sweeping

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Do forever:
- (a) $s \leftarrow$ current (nonterminal) state
- (b) $a \leftarrow policy(s,Q)$
- (c) Execute action a; observe resultant state, s', and reward, r
- (d) $Model(s,a) \leftarrow s', r$
- (e) $p \leftarrow |r + \gamma \max_{a'} Q(s',a') - Q(s,a)|$.
- (f) if $p > \theta$, then insert s,a into $PQueue$ with priority p
- (g) Repeat N times, while $PQueue$ is not empty:
  - $s,a \leftarrow first(PQueue)$
  - $s', r \leftarrow Model(s,a)$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  - Repeat, for all $\bar{s}, \bar{a}$ predicted to lead to s:
    - $\bar{r} \leftarrow$ predicted reward
    - $p \leftarrow |\bar{r} + \gamma \max_a Q(s,a) - Q(\bar{s},\bar{a})|$.
    - if $p > \theta$ then insert $\bar{s}, \bar{a}$ into $PQueue$ with priority p

What to update?

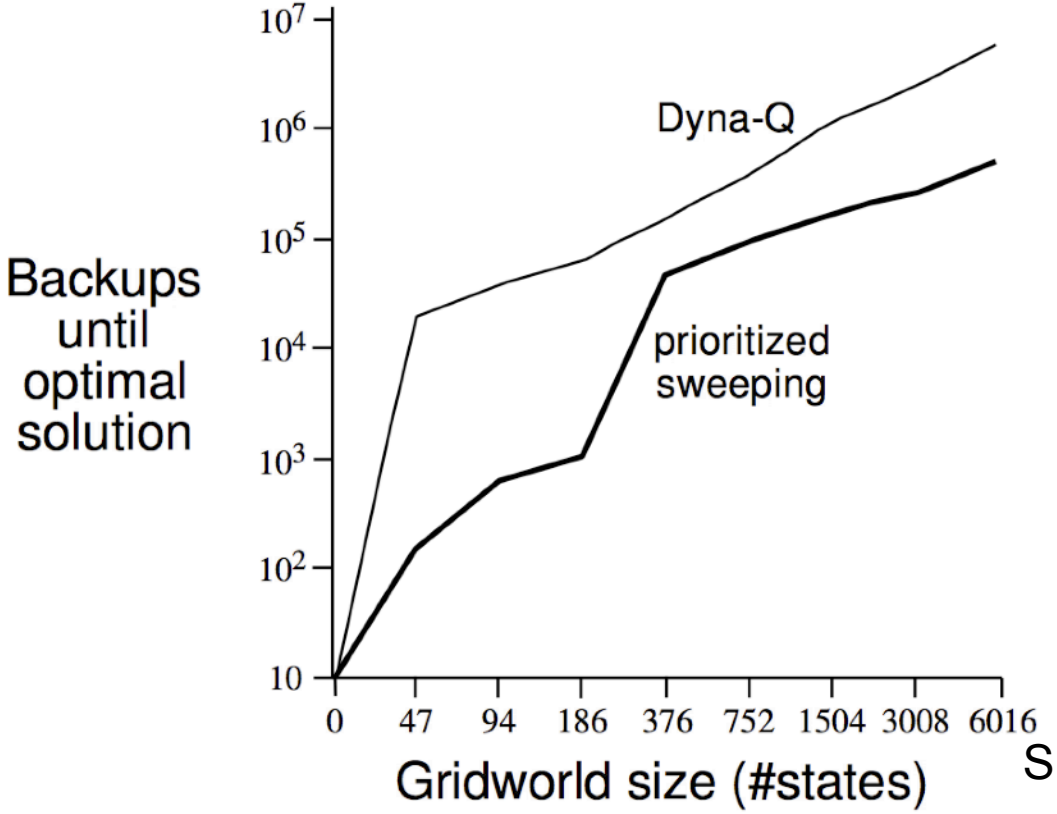


Figure from Sutton and Barto RL:AI

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

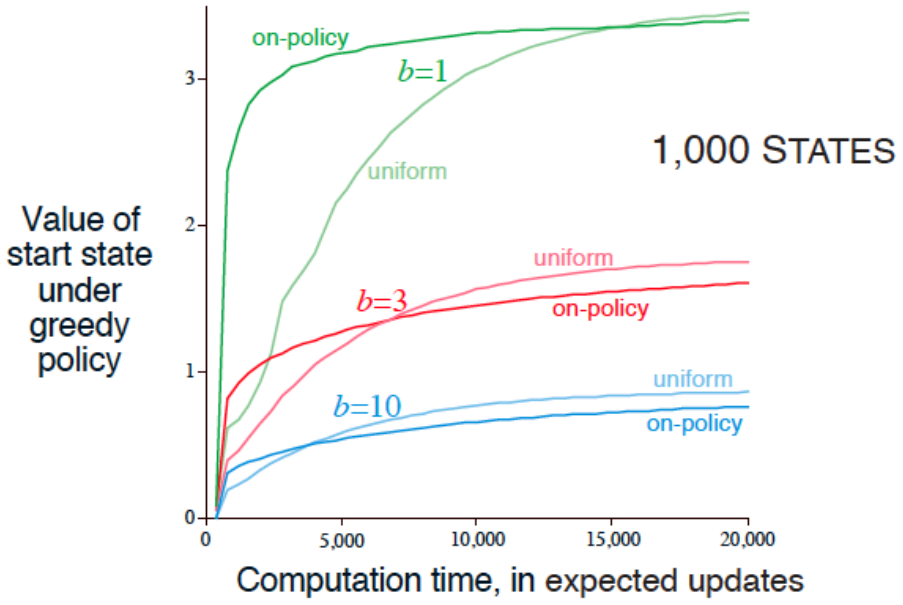
full / distributional?

sample / generative?

simulation / episodic?

What to update

b = branching factor



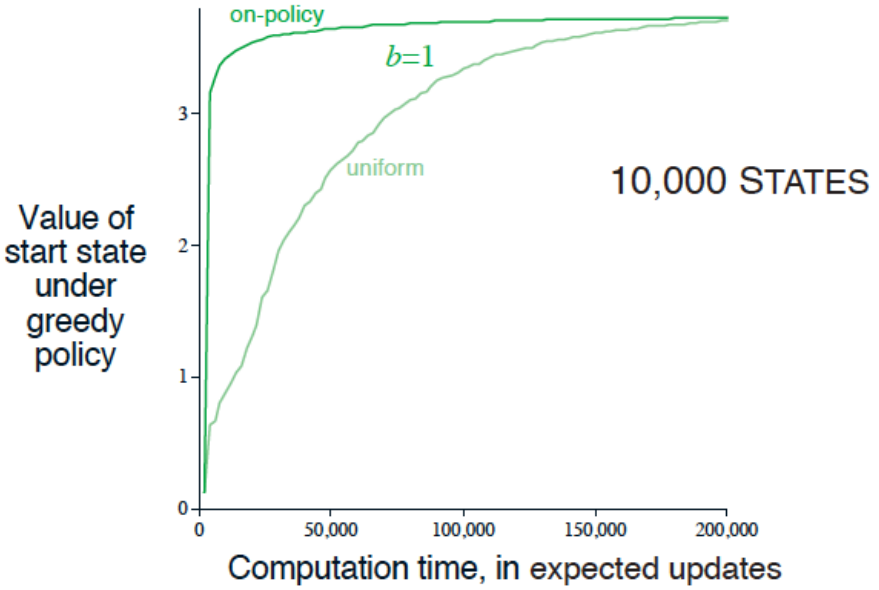


Figure from Sutton and Barto RL:AI

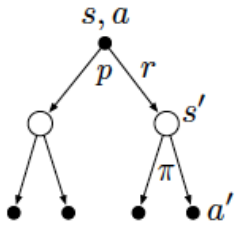
How to update

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

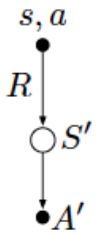
Knowledge of MDP needed, but model can be substituted

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

$q_\pi(s,a)$

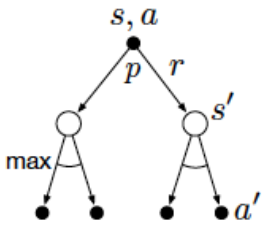


q-policy evaluation

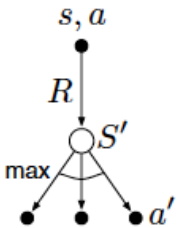


Sarsa

$q_*(s,a)$



q-value iteration

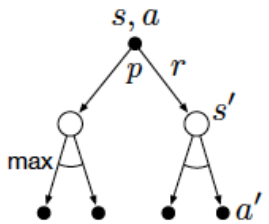


Q-learning

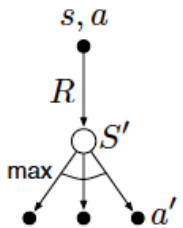
How to update

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

$q_*(s, a)$



q-value iteration



Q-learning

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!

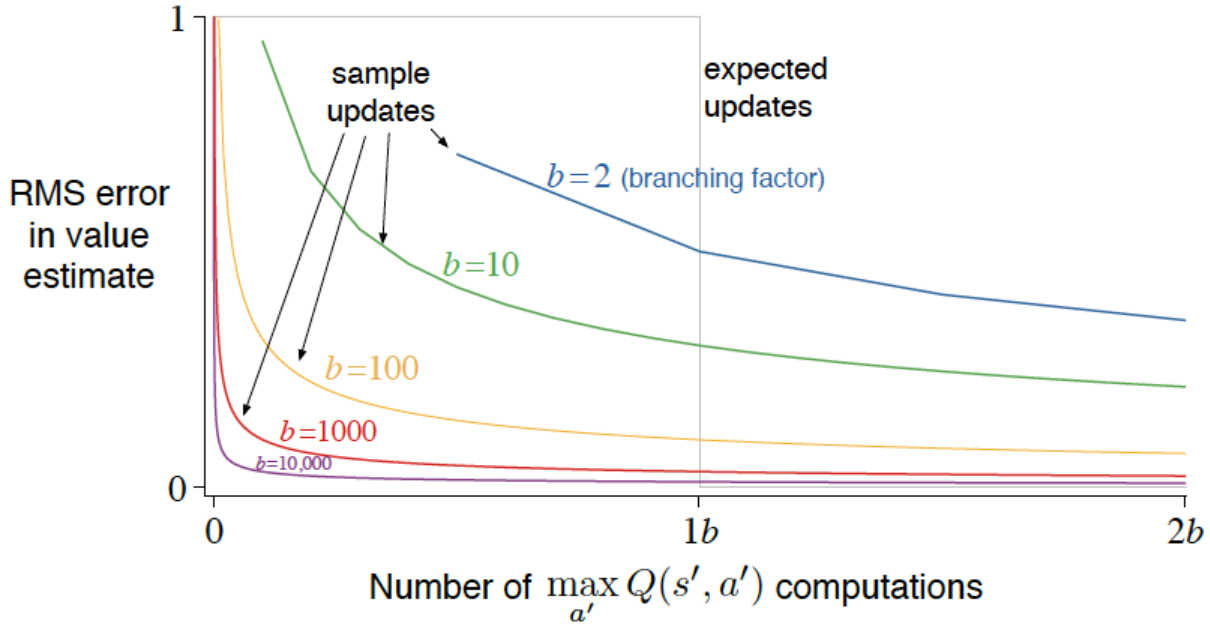


Figure from Sutton and Barto RL:AI

How to update

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

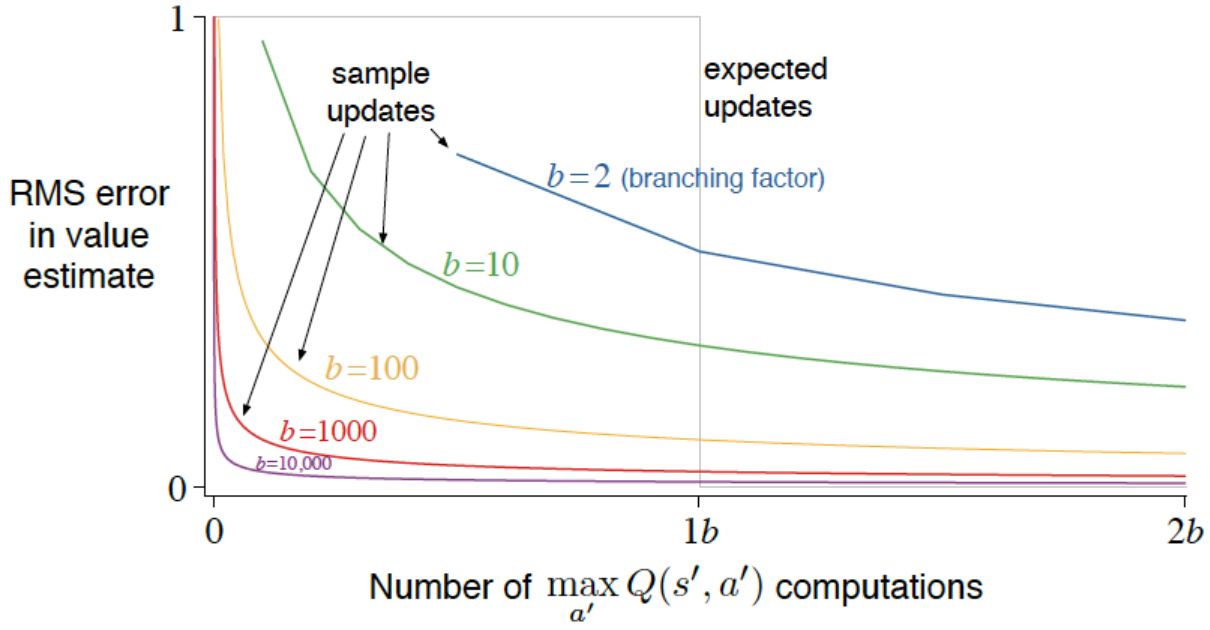
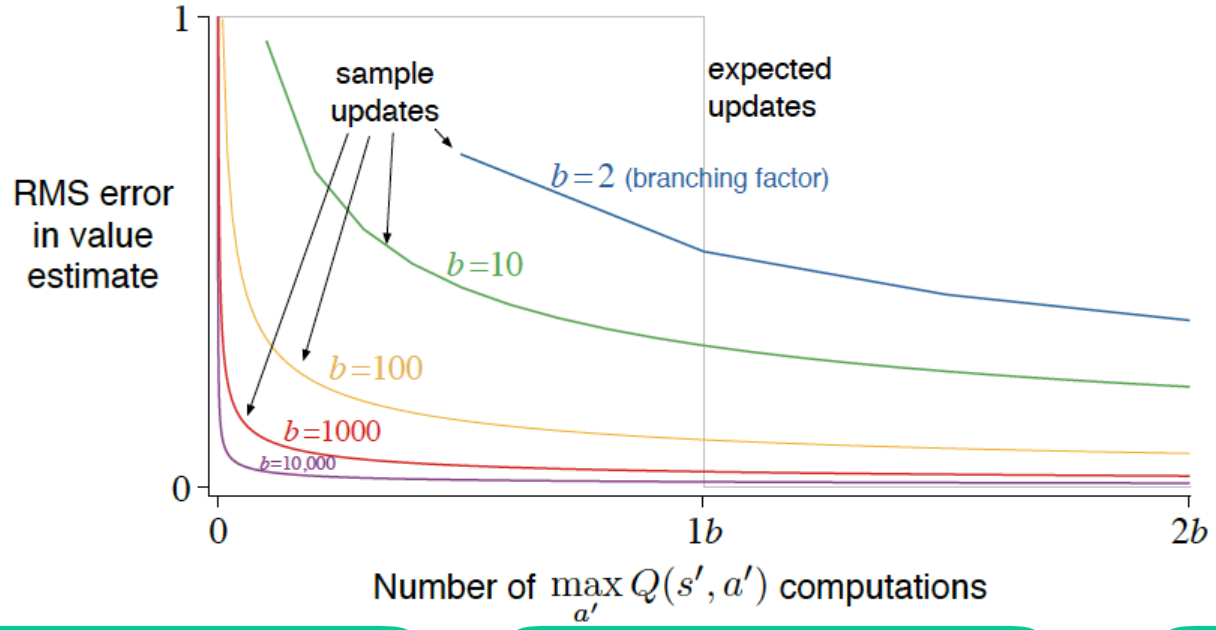


Figure from Sutton and Barto RL:AI

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 γ^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 γ^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

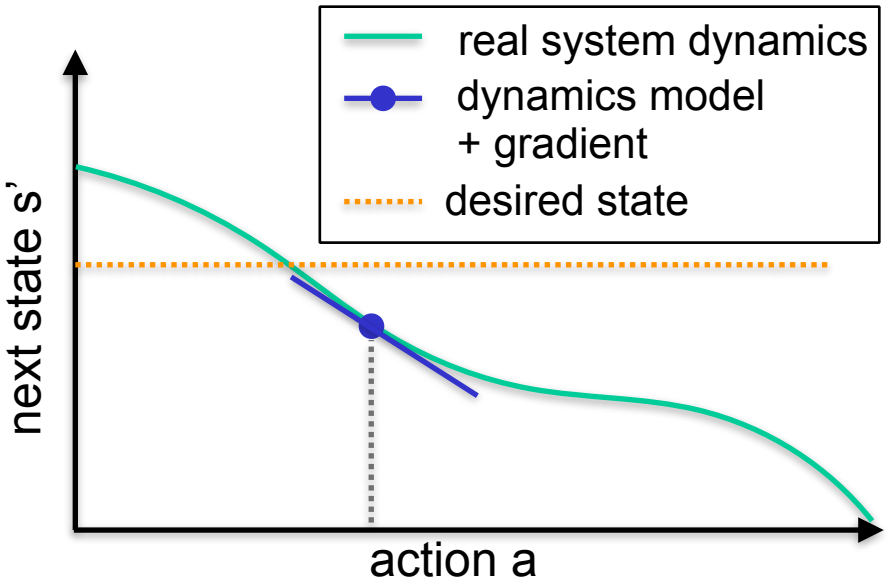
$$\boldsymbol{s}_{t+1}, r_{t+1} = f\left(\boldsymbol{s}_t, \boldsymbol{a}_t\right), \quad \boldsymbol{s}_0 \sim p\left(\boldsymbol{s}_0\right)$$

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

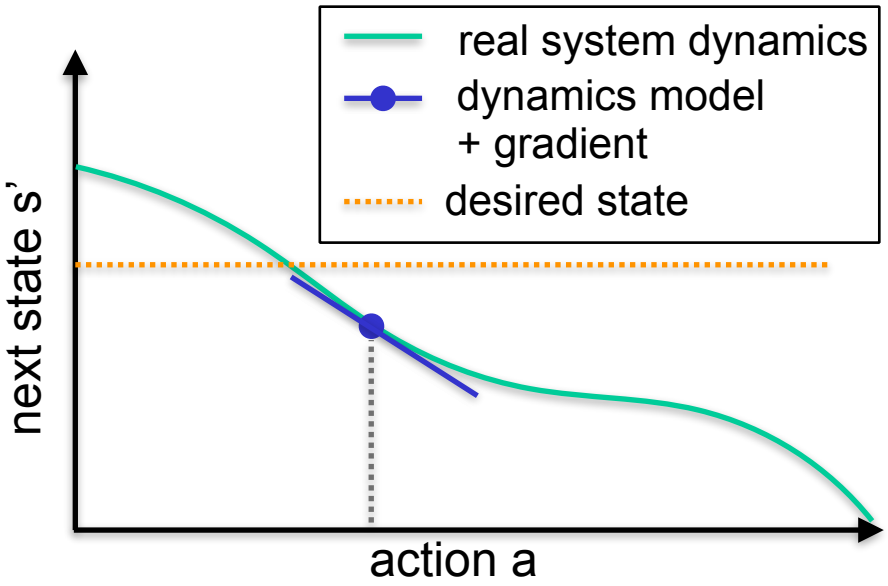
Use f to:

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



Environment model with gradient



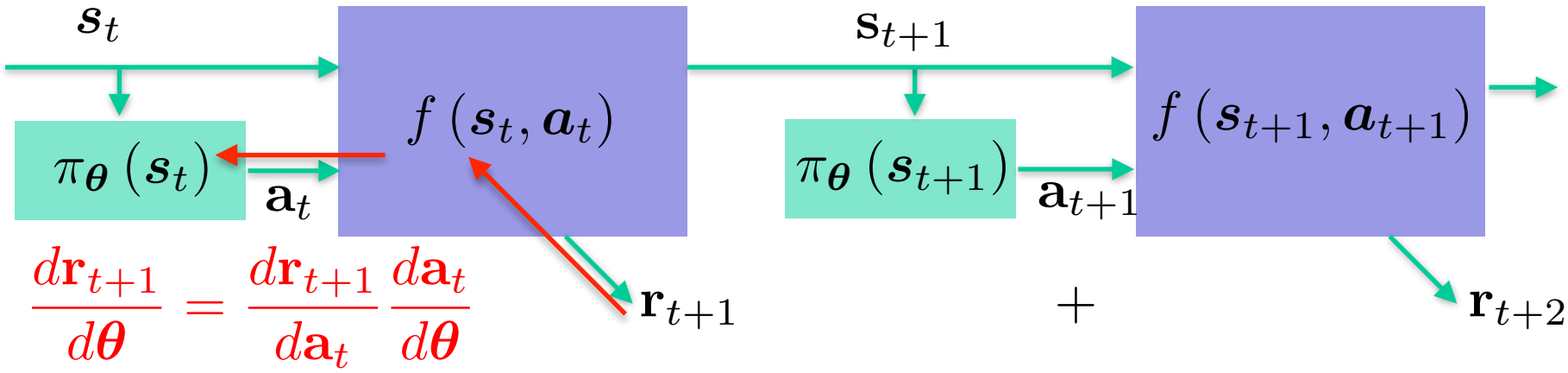
Since the dynamics model *f* has multiple inputs *(s,a)* and outputs *(s',r)* there are multiple derivatives to consider:

$$\frac{ds'}{da}, \frac{ds'}{ds}, \frac{dr}{da}, \frac{dr}{ds}$$

Policy update strategies

Backpropagation with deterministic system and policy

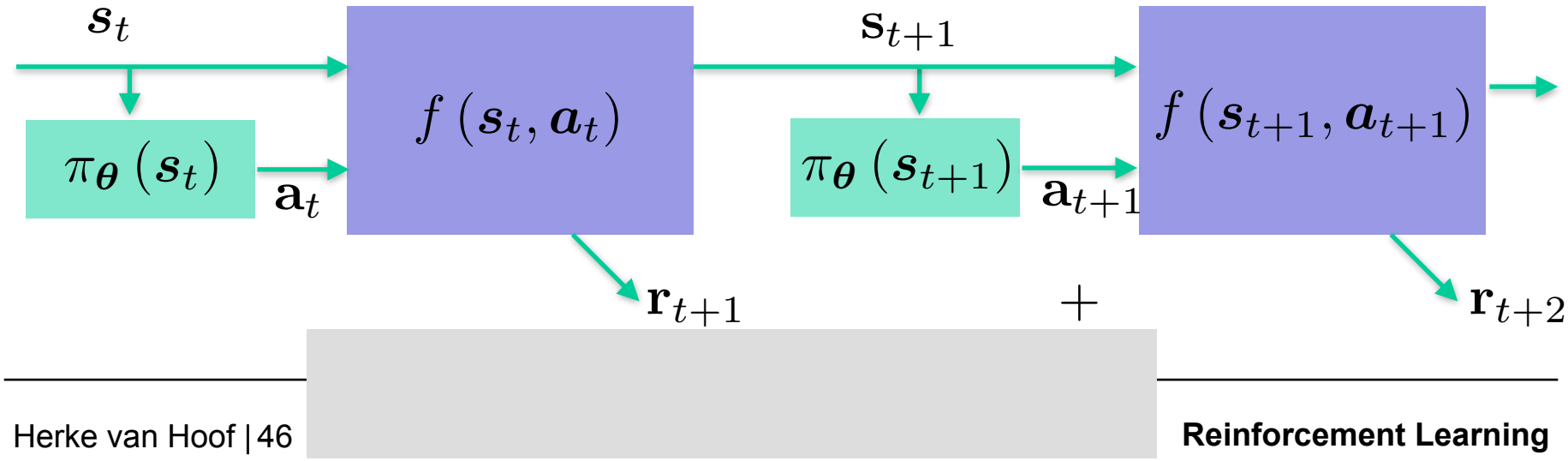
$$V(\mathbf{s}_t) = r_{t+1} + \gamma r_{t+2} + \ldots$$
$$\nabla_{\boldsymbol{\theta}} V(\mathbf{s}_t) = \nabla_{\boldsymbol{\theta}} r_{t+1} + \gamma \nabla_{\boldsymbol{\theta}} r_{t+2} + \ldots$$



Policy update strategies

Backpropagation with deterministic system and policy

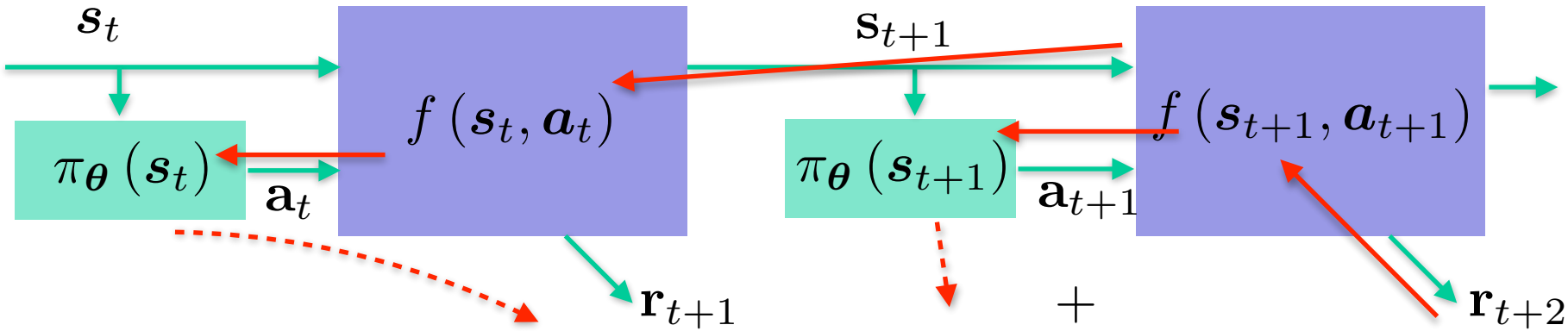
$$V(\mathbf{s}_t) = r_{t+1} + \gamma r_{t+2} + \ldots$$
$$\nabla_{\boldsymbol{\theta}} V(\mathbf{s}_t) = \nabla_{\boldsymbol{\theta}} r_{t+1} + \gamma \nabla_{\boldsymbol{\theta}} r_{t+2} + \ldots$$



Policy update strategies

Backpropagation with deterministic system and policy

$$V(\mathbf{s}_t) = r_{t+1} + \gamma r_{t+2} + \dots$$
$$\nabla_{\boldsymbol{\theta}} V(\mathbf{s}_t) = \nabla_{\boldsymbol{\theta}} r_{t+1} + \gamma \nabla_{\boldsymbol{\theta}} r_{t+2} + \dots$$



$$\frac{dr_{t+2}}{d\boldsymbol{\theta}} = \frac{dr_{t+2}}{d\mathbf{s}_{t+1}} \frac{d\mathbf{s}_{t+1}}{d\mathbf{a}_t} \frac{d\mathbf{a}_t}{d\boldsymbol{\theta}} + \frac{dr_{t+2}}{d\mathbf{a}_{t+1}} \frac{\partial \mathbf{a}_{t+1}}{\partial \boldsymbol{\theta}}$$

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