Reinforcement learning tutorial

Wouter Caarls

Koroibot Summer School, September 25th, 2014

What is reinforcement learning?

[R. Nagel, 2008]

What is reinforcement learning?

Supervised learning

Given samples of $x \rightarrow y$, find $y = f(x)$ (regression)

Reinforcement learning

Given samples of $(x, y) \rightarrow z$, find $y = f(x)$ that maximizes z.

Unsupervised learning

Given a dataset x, find interesting patterns (clusters, relations, etc.)

What is reinforcement learning?

Supervised learning

Given samples of $x \rightarrow y$, find $y = f(x)$ (regression)

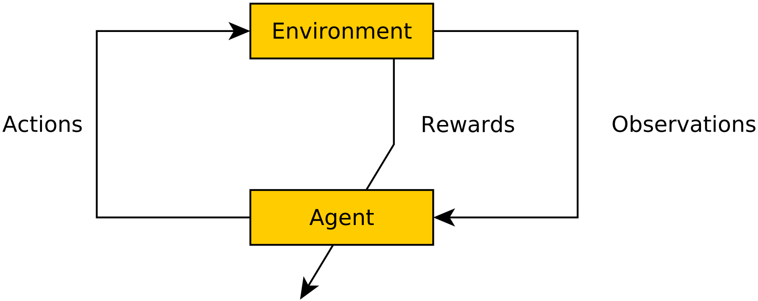
Reinforcement learning

Given samples of $(x, y) \rightarrow z$, find $y = f(x)$ that maximizes z.

Unsupervised learning

Given a dataset x, find interesting patterns (clusters, relations, etc.)

Learning by trial and error



Goal

Find actions that maximize the reward received over the lifetime of the agent.

Why reinforcement learning?

- Specify the problem, not the solution
  - Easier: desired outcome is usually known
  - Still tricky: reward engineering
- Can deal with delayed rewards
  - Optimizes over all future rewards
- Model-free
  - No mapping, system identification, etc. beforehand
  - Works in any (static) environment

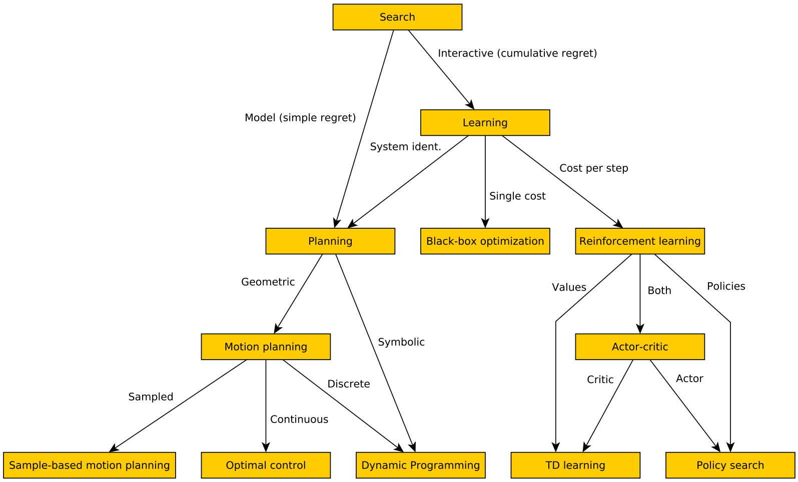
Why reinforcement learning?

- Specify the problem, not the solution
  - Easier: desired outcome is usually known
  - Still tricky: reward engineering
- Can deal with delayed rewards
  - Optimizes over all future rewards
- Model-free
  - No mapping, system identification, etc. beforehand
  - Works in any (static) environment

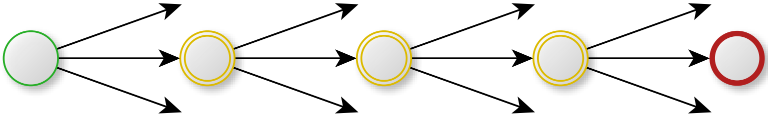
Why reinforcement learning?

- Specify the problem, not the solution
  - Easier: desired outcome is usually known
  - Still tricky: reward engineering
- Can deal with delayed rewards
  - Optimizes over all future rewards
- Model-free
  - No mapping, system identification, etc. beforehand
  - Works in any (static) environment

Context of learning control



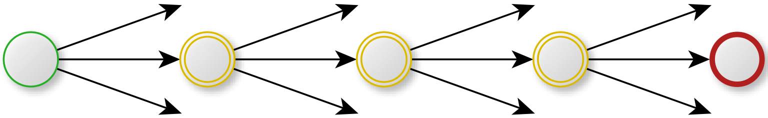
Interactive (no model)



All actions are sequentially executed in the real world.

- Sequentially
  - Observations only occur in trajectories
  - Cannot evaluate different actions in the same state
- Real world
  - Actions take real time
  - Actions must be selected as fast as possible
  - Dangerous actions can damage the system

Cost per step



Reward is available at intermediate steps and only based on the current transition.

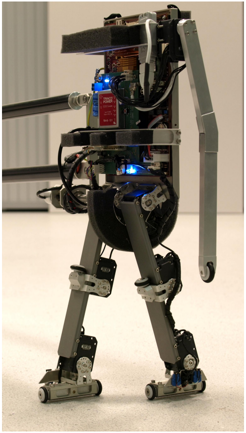
- Intermediate steps
  - Allows us to assign credit to particular actions
  - May be delayed
- Based on current transition
  - No integrated costs at the end of a trajectory
  - No "target arrival time" unless time can be observed

Example: LEO

Main characteristics:

- ‘2D‘ (boom keeps hip axis horizontal)
- $\sim$50 cm
- $\sim$1.7 kg
- 7 servo motors (Dynamixel)
- On-board computing
- Autonomous (except power)

Task:
Learn to walk

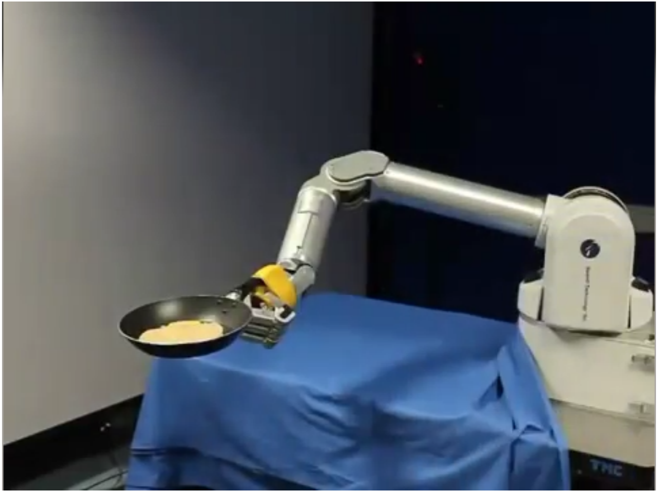


[Schuitema et al., 2010]

Example: LEO

[Schuitema, 2010]

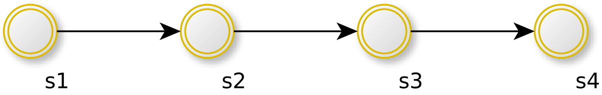
Example: Pancake flipping



[Kormuschev et al., 2010]

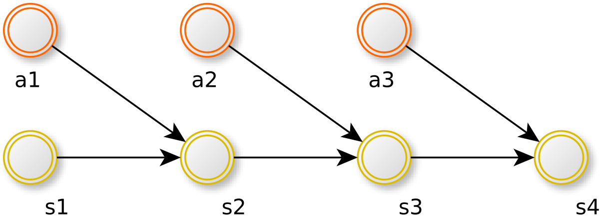
Outline

Markov chain



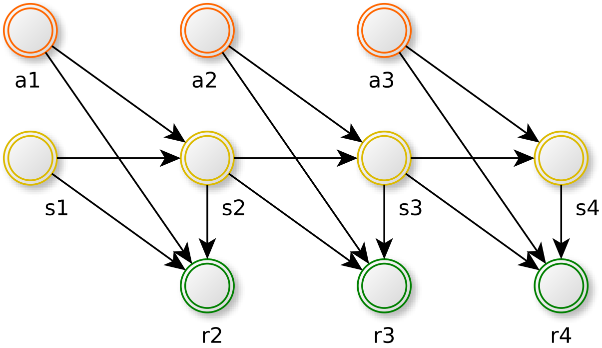
- Next state depends only on current state
- Transitions may be stochastic, but distribution must be static given the current state

Markov process



- Action also influences next state

Markov decision process

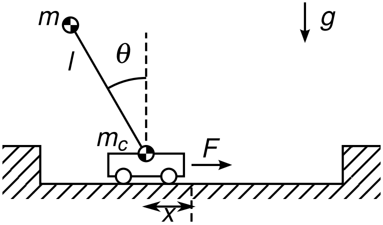


- Introduces optimization criterion: find actions that maximize reward
- Reward (cost) depends only on the state transition

States and actions

Next state depends only on current state and action

- Given current state and action, next state distribution is static
  $p(s'|s,a) = \mathscr{T}(s,a,s')$
- History-independent
  $p(s_n|s_1,a_1,s_2,a_2,\ldots,s_{n-1},a_{n-1}) = p(s_n|s_{n-1},a_{n-1})$

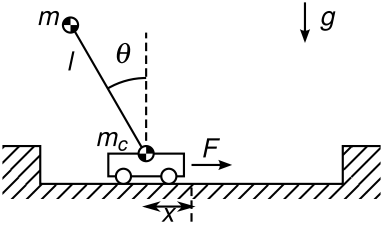


$g,\ m,\ \theta,\ \dot{\theta},\ \ddot{\theta},\ l,\ m_c,\ F,\ x,\ \dot{x},\ \ddot{x}$

States and actions

Next state depends only on current state and action

- Given current state and action, next state distribution is static
  $p(s'|s,a) = \mathscr{T}(s,a,s')$
- History-independent
  $p(s_n|s_1,a_1,s_2,a_2,\ldots,s_{n-1},a_{n-1}) = p(s_n|s_{n-1},a_{n-1})$

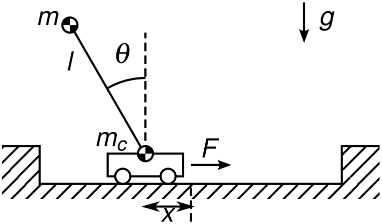


$g,\ m,\ \theta,\ \dot{\theta},\ \ddot{\theta},\ l,\ m_c,\ F,\ x,\ \dot{x},\ \ddot{x}$

States and actions

Next state depends only on current state and action

- Given current state and action, next state distribution is static
  $p(s'|s,a) = \mathscr{T}(s,a,s')$
- History-independent
  $p(s_n|s_1, a_1, s_2, a_2, \ldots, s_{n-1}, a_{n-1}) = p(s_n|s_{n-1}, a_{n-1})$

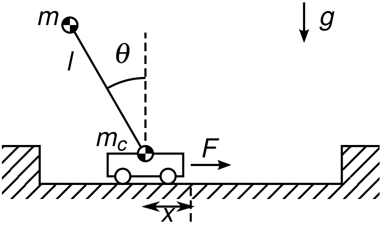


g, m, θ, $\dot{\theta}$, $\ddot{\theta}$, l, m_c, F, x, $\dot{x}$, $\ddot{x}$

States and actions

Next state depends only on current state and action

- Given current state and action, next state distribution is static
  $p(s'|s,a) = \mathscr{T}(s,a,s')$
- History-independent
  $p(s_n|s_1,a_1,s_2,a_2,\ldots,s_{n-1},a_{n-1}) = p(s_n|s_{n-1},a_{n-1})$

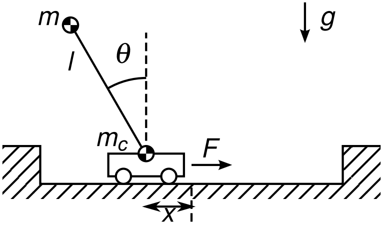


g, m, θ, $\dot{\theta}$, $\ddot{\theta}$, l, m_c, F, x, $\dot{x}$, $\ddot{x}$

States and actions

Next state depends only on current state and action

- Given current state and action, next state distribution is static
  $p(s'|s,a) = \mathscr{T}(s,a,s')$
- History-independent
  $p(s_n|s_1,a_1,s_2,a_2,\ldots,s_{n-1},a_{n-1}) = p(s_n|s_{n-1},a_{n-1})$



g, m, θ, $\dot{\theta}$, $\ddot{\theta}$, l, m_c, F, x, $\dot{x}$, $\ddot{x}$

Rewards

Determine the task

- Depend only on the current state transition
  $p(r|s,a,s') = \rho(s,a,s')$
- Learning goal is to maximize the expected return
  $R_t = r_{t+1} + r_{t+2} + \cdots + r_T$
- Should reward desirable behaviors and punish undesirable ones
- In practice, often given by designer instead of environment

Define the problem, not the solution (reward engineering).

Episodic and continuing tasks

- Some tasks are naturally episodic:
  Reaching, grasping, standing up, kicking, etc.
- Others are continuing:
  Holding, balancing, walking, etc.
- Continuing tasks ($T = \infty$) may have infinite returns
- Introduce discounted return:

$$
\begin{aligned}
R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \\
&= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}
\end{aligned}
$$

- Discount factor $\gamma \in [0, 1]$ determines task horizon

Control policy

Specifies which action to take in each state

- Due to Markov property, only has to depend on current state $\pi(a|s) = f(s)$
- Optimal policy π^* maximizes expected return

$$\pi^* = \arg\max_{\pi} E_{\pi}\left\{R_t\right\}$$

Optimal subproblems

In an acyclic MDP, the optimal policy for a state s is independent of the optimal policy for the preceeding states, leading to recursive solutions.

Control policy

Specifies which action to take in each state

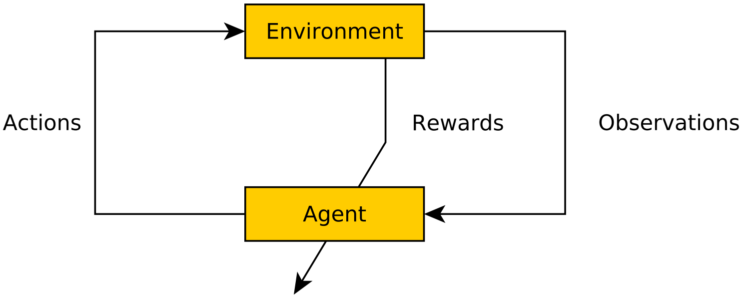
- Due to Markov property, only has to depend on current state
  $\pi(a|s) = f(s)$
- Optimal policy π^* maximizes expected return

$$\pi^* = \arg\max_{\pi} E_{\pi}\{R_t\}$$

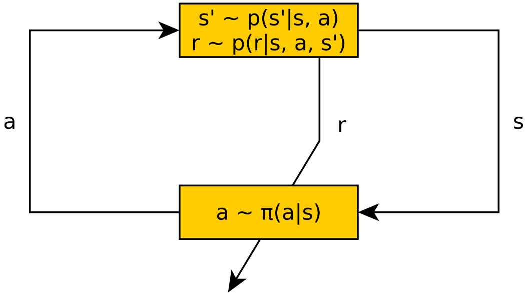
Optimal subproblems

In an acyclic MDP, the optimal policy for a state s is independent of the optimal policy for the preceeding states, leading to recursive solutions.

Control scheme

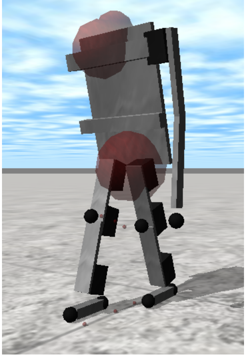


Control scheme



Robot task modeled as an MDP

- States:
  $s = [\theta, \dot{\theta}]$
- Actions:
  $a = \tau$
- Transition function:
  $s' = s + \int_0^t \mathrm{eom}(s,a)dt + \mathcal{N}(\mu, \sigma)$
- Rewards:
  $r = -sQs^T - Pa^2$
- Policy:
  $a = \pi(s)$



Value functions

Learning goal: find optimal policy π^* that maximizes the expected return R_t

$$\pi^* = \arg\max_{\pi} E_{\pi}\left\{R_t\right\}$$

Define a value function V^{π} that stores the expected return for each state s under a certain policy π:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{R_t | s_t = s\right\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \end{aligned}$$

then

$$\pi^* = \arg\max_{\pi} V^{\pi}$$

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s)$$

Value functions

Learning goal: find optimal policy π^* that maximizes the expected return R_t

$$\pi^* = \arg\max_{\pi} E_\pi \{R_t\}$$

Define a value function V^π that stores the expected return for each state s under a certain policy π:

$$\begin{aligned} V^\pi(s) &= E_\pi \{R_t | s_t = s\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \end{aligned}$$

then

$$\pi^* = \arg\max_{\pi} V^\pi$$

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^\pi(s)$$

Value functions

Learning goal: find optimal policy π^* that maximizes the expected return R_t

$$\pi^* = \arg\max_{\pi} E_\pi \left\{ R_t \right\}$$

Define a value function V^π that stores the expected return for each state s under a certain policy π:

$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ R_t | s_t = s \right\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \end{aligned}$$

then

$$\pi^* = \arg\max_{\pi} V^\pi$$

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^\pi(s)$$

Recursive definition

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\}$$

$$= E_{\pi}\left\{r_{t+1} + \sum_{k=1}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\}$$

$$= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s\right\}$$

$$= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t = s\right\}$$

Allows us to write the optimal policy in terms of the optimal value function

$$\pi^*(s) = \arg\max_a E_{\pi}\left\{r_{t+1} + \gamma V^*(s_{t+1})|s_t = s, a_t = a\right\}$$

Recursive definition

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle| s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \sum_{k=1}^{\infty}\gamma^k r_{t+k+1}\middle| s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\middle| s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1})\middle| s_t = s\right\}
\end{aligned}
$$

Allows us to write the optimal policy in terms of the optimal value function

$$
\pi^*(s) = \arg\max_a E_{\pi}\left\{r_{t+1} + \gamma V^*(s_{t+1})\middle| s_t = s, a_t = a\right\}
$$

Recursive definition

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \sum_{k=1}^{\infty}\gamma^k r_{t+k+1} | s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} | s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\right\}
\end{aligned}
$$

Allows us to write the optimal policy in terms of the optimal value function

$$\pi^*(s) = \arg\max_a E_{\pi}\left\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right\}$$

Recursive definition

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \middle| s_t = s\right\}$$

$$= E_{\pi}\left\{r_{t+1} + \sum_{k=1}^{\infty}\gamma^k r_{t+k+1} \middle| s_t = s\right\}$$

$$= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \middle| s_t = s\right\}$$

$$= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\right\}$$

Allows us to write the optimal policy in terms of the optimal value function

$$\pi^*(s) = \arg\max_a E_{\pi}\left\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right\}$$

Recursive definition

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+1}|s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \sum_{k=1}^{\infty}\gamma^{k}r_{t+k+1}|s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+2}|s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t = s\right\}
\end{aligned}
$$

Allows us to write the optimal policy in terms of the optimal value function

$$\pi^*(s) = \arg\max_{a} E_{\pi}\left\{r_{t+1} + \gamma V^*(s_{t+1})|s_t = s, a_t = a\right\}$$

Recursive definition

$$
\begin{aligned}
V^\pi(s) &= E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s\right\} \\
&= E_\pi\left\{r_{t+1} + \sum_{k=1}^{\infty}\gamma^k r_{t+k+1} | s_t = s\right\} \\
&= E_\pi\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} | s_t = s\right\} \\
&= E_\pi\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\right\}
\end{aligned}
$$

Allows us to write the optimal policy in terms of the optimal value function

$$
\pi^*(s) = \arg\max_a E_\pi\left\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right\}
$$

State-action value functions

Stores expected return for each state-action combination

$$
\begin{aligned}
Q^{\pi}(s,a) &= E_{\pi}\left\{R_t | s_t = s, a_t = a\right\} \\
&= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right\}
\end{aligned}
$$

Allows us to find optimal actions without calculating expectation

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

$$\pi^*(s) = \arg\max_{a} Q^*(s,a)$$

State-action value functions

Stores expected return for each state-action combination

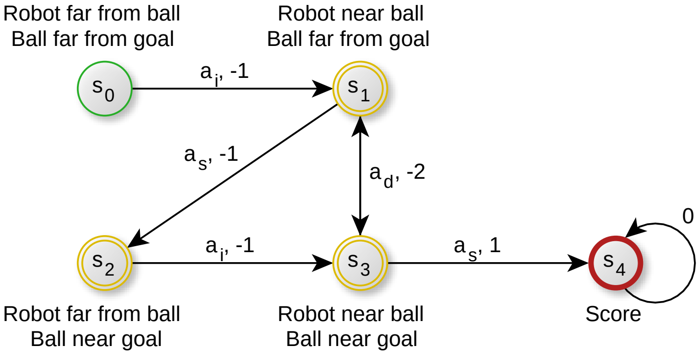
$$
\begin{aligned}
Q^{\pi}(s,a) &= E_{\pi}\left\{R_t | s_t = s, a_t = a\right\} \\
&= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right\}
\end{aligned}
$$

Allows us to find optimal actions without calculating expectation

$$
\begin{aligned}
Q^{*}(s,a) &= \max_{\pi} Q^{\pi}(s,a) \\
\pi^{*}(s) &= \arg\max_{a} Q^{*}(s,a)
\end{aligned}
$$

Example MDP

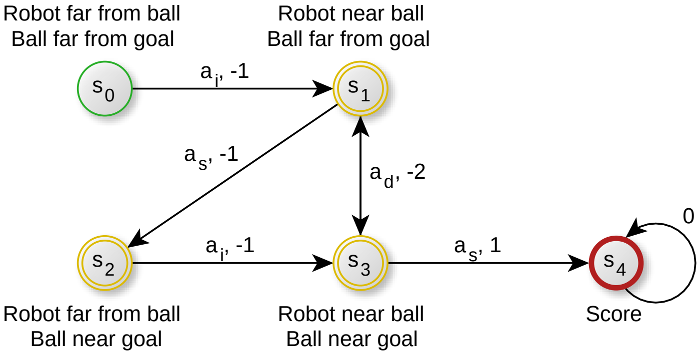
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$Q^{\pi}(s_0, a_i) = -1 + \gamma \cdot -2 + \gamma^2 \cdot 1$$
$$= -1 - 1.8 + 0.81$$
$$= -1.99$$

Example MDP

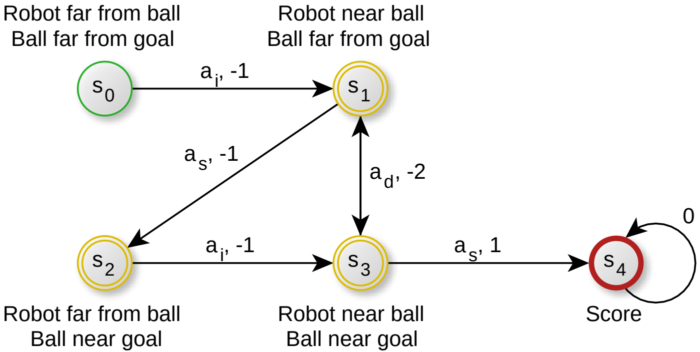
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$Q^\pi(s_0, a_i) = -1 + \gamma \cdot -2 + \gamma^2 \cdot 1$$
$$= -1 - 1.8 + 0.81$$
$$= -1.99$$

Example MDP

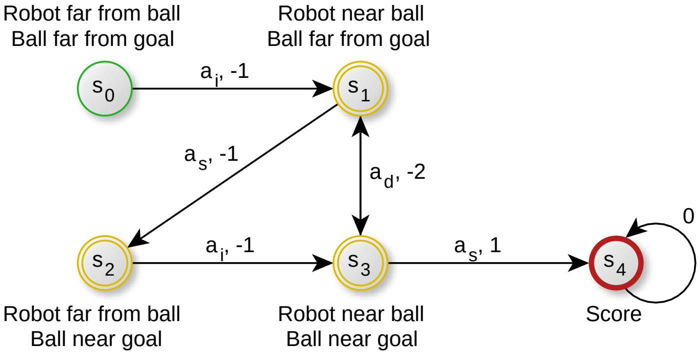
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$Q^\pi(s_0, a_i) = -1 + \gamma \cdot -2 + \gamma^2 \cdot 1$$
$$= -1 - 1.8 + 0.81$$
$$= -1.99$$

Example MDP

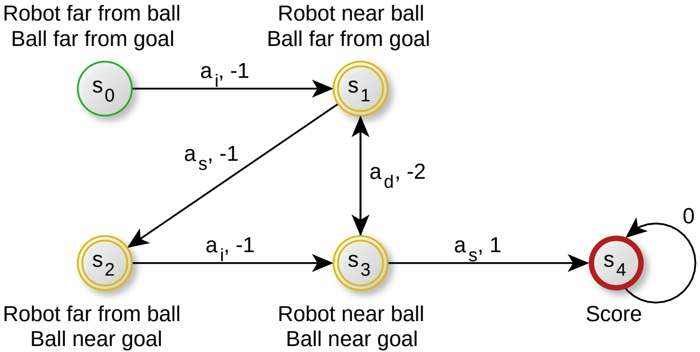
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$Q^\pi(s_0, a_i) = -1 + \gamma \cdot -2 + \gamma^2 \cdot 1$$
$$= -1 - 1.8 + 0.81$$
$$= -1.99$$

Example MDP

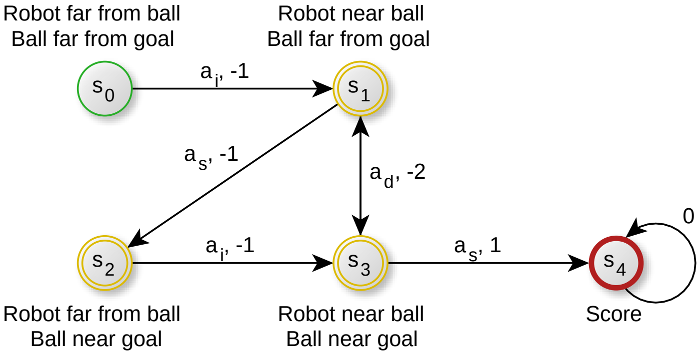
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$Q^{\pi}(s_0, a_i) = -1 + \gamma \cdot -2 + \gamma^2 \cdot 1$$
$$= -1 - 1.8 + 0.81$$
$$= -1.99$$

Example MDP

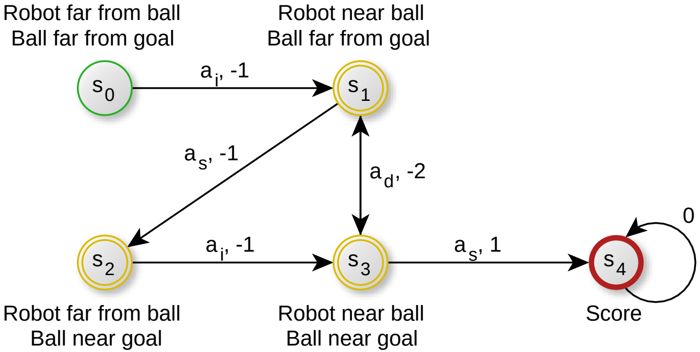
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$\begin{aligned} Q^\pi(s_0, a_i) &= -1 + \gamma \cdot -2 + \gamma^2 \cdot 1 \\ &= -1 - 1.8 + 0.81 \\ &= -1.99 \end{aligned}$$

Example MDP, recursive version

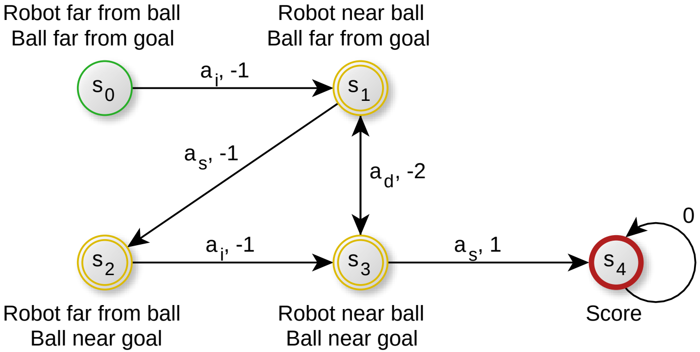
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$Q^\pi(s_4, *) = 0 + \gamma Q^\pi(s_4, *) \qquad = 0$$
$$Q^\pi(s_3, a_s) = 1 + \gamma Q^\pi(s_4, *) \qquad = 1$$
$$Q^\pi(s_1, a_d) = -2 + \gamma Q^\pi(s_3, a_s) \qquad = -1.1$$
$$Q^\pi(s_0, a_i) = -1 + \gamma Q^\pi(s_1, a_d) \qquad = -1.99$$

Example MDP, recursive version

$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$Q^\pi(s_4,*) = 0 + \gamma Q^\pi(s_4,*) \qquad = 0$$
$$Q^\pi(s_3,a_s) = 1 + \gamma Q^\pi(s_4,*) \qquad = 1$$
$$Q^\pi(s_1,a_d) = -2 + \gamma Q^\pi(s_3,a_s) \qquad = -1.1$$
$$Q^\pi(s_0,a_i) = -1 + \gamma Q^\pi(s_1,a_d) \qquad = -1.99$$

Example MDP, recursive version

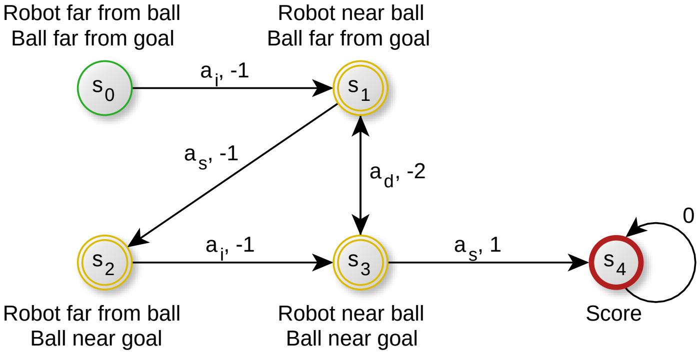
$\gamma = 0.9$

$\pi(s_0) = a_i$

$\pi(s_1) = a_d$

$\pi(s_2) = a_i$

$\pi(s_3) = a_s$



$$Q^\pi(s_4, *) = 0 + \gamma Q^\pi(s_4, *) \qquad = 0$$

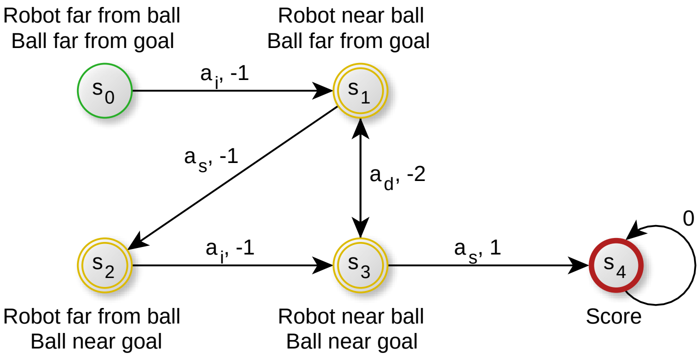
$$Q^\pi(s_3, a_s) = 1 + \gamma Q^\pi(s_4, *) \qquad = 1$$

$$Q^\pi(s_1, a_d) = -2 + \gamma Q^\pi(s_3, a_s) \qquad = -1.1$$

$$Q^\pi(s_0, a_i) = -1 + \gamma Q^\pi(s_1, a_d) \qquad = -1.99$$

Example MDP, recursive version

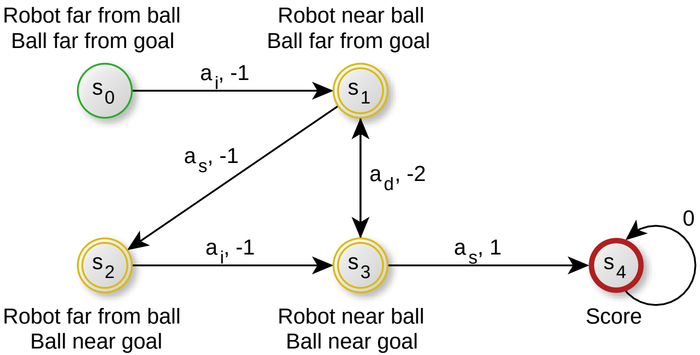
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$Q^\pi(s_4,*) = 0 + \gamma Q^\pi(s_4,*) \qquad = 0$$
$$Q^\pi(s_3,a_s) = 1 + \gamma Q^\pi(s_4,*) \qquad = 1$$
$$Q^\pi(s_1,a_d) = -2 + \gamma Q^\pi(s_3,a_s) \qquad = -1.1$$
$$Q^\pi(s_0,a_i) = -1 + \gamma Q^\pi(s_1,a_d) \qquad = -1.99$$

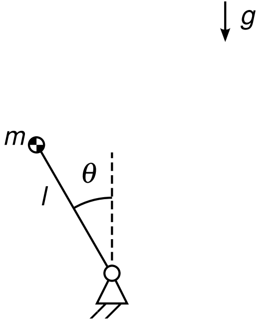
Example MDP, recursive version

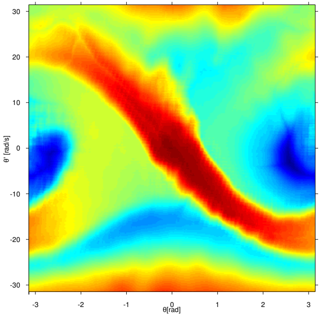
$$\gamma = 0.9$$
$$\pi(s_0) = a_i$$
$$\pi(s_1) = a_d$$
$$\pi(s_2) = a_i$$
$$\pi(s_3) = a_s$$



$$\begin{aligned}
Q^\pi(s_4, *) &= 0 + \gamma Q^\pi(s_4, *) && = 0 \\
Q^\pi(s_3, a_s) &= 1 + \gamma Q^\pi(s_4, *) && = 1 \\
\color{red}{Q^\pi(s_1, a_d)} &= -2 + \gamma Q^\pi(s_3, a_s) && = -1.1 \\
Q^\pi(s_0, a_i) &= -1 + \gamma \color{red}{Q^\pi(s_1, a_d)} && = -1.99
\end{aligned}$$

Value function for pendulum swing-up





- System starts in down position, goal is to stabilize in the up position
- Can apply torque to the rotation axis

Control

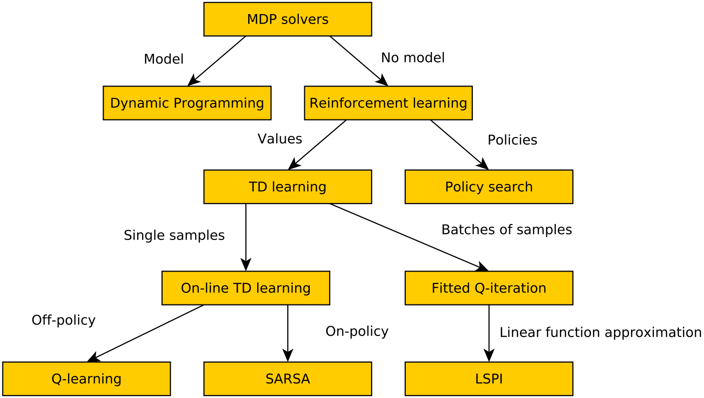
Can now evaluate value functions for arbitrary policies, but how do we find the optimal policy?

- Optimal policy is greedy with respect to optimal value function $\pi^*(s) = \arg\max_a Q^*(s,a)$
- Let $\pi'(s) = \arg\max_a Q^\pi(s,a)$, then $Q^{\pi'} \geq Q^\pi$

Leads to iterative dynamic programming solutions

Outline

Overview


MDP solvers
Model
No model
Dynamic Programming
Reinforcement learning
Values
Policies
TD learning
Policy search
Single samples
Batches of samples
On-line TD learning
Fitted Q-iteration
Off-policy
On-policy
Linear function approximation
Q-learning
SARSA
LSPI

Dynamic programming

Solves problems by breaking them down into simpler subproblems

- For MDPs, the solution to a future state is a subproblem to the solution for the current state

Follow the policy that is greedy with respect to the optimal value function, which satisfies the Bellman equation

$$
\begin{aligned}
V^*(s) &= \max_a E_{s'}\left\{\rho(s,a,s') + \gamma V^*(s')\right\} \\
&= \max_a \sum_{s'} p(s'|s,a)\left(\rho(s,a,s') + \gamma V^*(s')\right)
\end{aligned}
$$

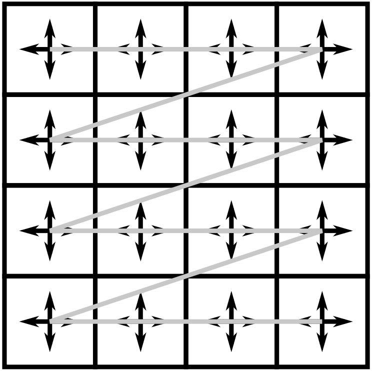
Value iteration

Elementary solution method: iteratively approximate optimal value function

function ValueIteration
  $t \leftarrow 0$
  $V_0(s) \leftarrow 0$ for all s
  repeat
    for each s **do**
      $V_{t+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left(\rho(s,a,s') + \gamma V_t(s')\right)$
    end for
    $t \leftarrow t+1$
  until $\|V_t - V_{t-1}\| < \varepsilon$
  return V
end function

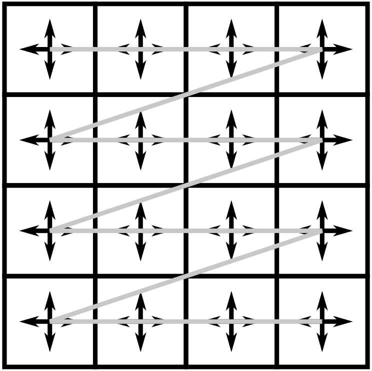
Bootstrapping method: updates estimates based on estimates

Ditching the model



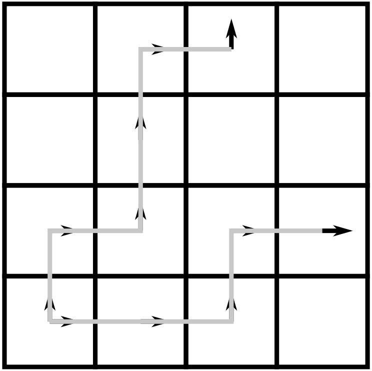
- Observations only occur in trajectories
- Cannot evaluate different actions in the same state

Ditching the model



- Observations only occur in trajectories
- Cannot evaluate different actions in the same state

Ditching the model



- Observations only occur in trajectories
- Cannot evaluate different actions in the same state

Temporal difference learning

Value iteration requires known transition model $p(s'|s,a)$. Temporal difference learning only samples it:

function TDLEARNING(π)
  $s \leftarrow s_0$
  $V^\pi(s) \leftarrow 0$ for all s
  repeat
    Sample $s' \sim p(s'|s,\pi(s))$ and reward $r = \rho(s,a,s')$
    $V^\pi(s) \leftarrow V^\pi(s) + \alpha\left(r + \gamma V^\pi(s') - V^\pi(s)\right)$
    $s \leftarrow s'$ Temporal difference error
  until convergence
  return V
end function

α is a learning rate that acts as an exponential moving average filter

Temporal difference learning

Value iteration requires known transition model $p(s'|s,a)$. Temporal difference learning only samples it:

function TDLEARNING(π)
  $s \leftarrow s_0$
  $V^\pi(s) \leftarrow 0$ for all s
  repeat
    Sample $s' \sim p(s'|s,\pi(s))$ and reward $r = \rho(s,a,s')$
    $V^\pi(s) \leftarrow V^\pi(s) + \alpha\left(r + \gamma V^\pi(s') - V^\pi(s)\right)$
    $s \leftarrow s'$ Temporal difference error
  until convergence
  return V
end function

α is a learning rate that acts as an exponential moving average filter

Q-learning: Temporal difference control

TD learning estimates state-value function $V^{\pi}(s)$. For control, we must estimate state-action value function $Q(s,a)$ instead. Q-learning estimates the optimal state-action value function $Q^*(s,a)$:

function QLearning
  $s \leftarrow s_0$
  $Q(s,a) \leftarrow 0$ for all s, a
  repeat
    $a \leftarrow \arg\max_a Q(s,a)$
    Sample $s' \sim p(s'|s,a)$ and reward $r = \rho(s,a,s')$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
    $s \leftarrow s'$
  until convergence
  return Q
end function

Exploration

To guarantee convergence, every state has to be sampled an infinite number of times.

- In dynamic programming, this is guaranteed by sweeping the entire state space.
- When only trajectory sampling is available, requires exploration
- Use a stochastic exploration policy derived from $Q(s, a)$, e.g. (ε-greedy)

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|} & \text{if} a = \arg\max_{a'} Q(s, a') \\ \frac{\varepsilon}{|A|} & \text{otherwise} \end{cases}$$

Exploration-exploitation trade-off

Choose between exploiting current knowledge (refining current known best policy) or exploring new knowledge (finding better policies)

Exploration

To guarantee convergence, every state has to be sampled an infinite number of times.

- In dynamic programming, this is guaranteed by sweeping the entire state space.
- When only trajectory sampling is available, requires exploration
- Use a stochastic exploration policy derived from $Q(s, a)$, e.g. (ε-greedy)

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|} & \text{if} a = \arg\max_{a'} Q(s, a') \\ \frac{\varepsilon}{|A|} & \text{otherwise} \end{cases}$$

Exploration-exploitation trade-off

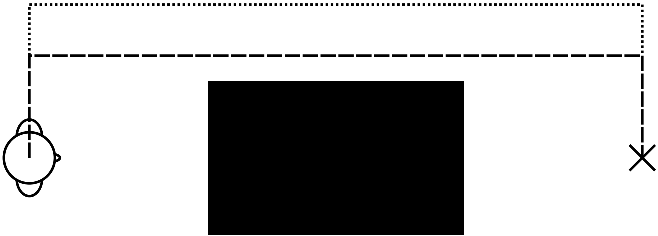
Choose between exploiting current knowledge (refining current known best policy) or exploring new knowledge (finding better policies)

SARSA: on-policy TD control

Q-learning finds the value function of the optimal policy π^* while following a stochastic exploratory policy π. SARSA finds the value function of π itself:

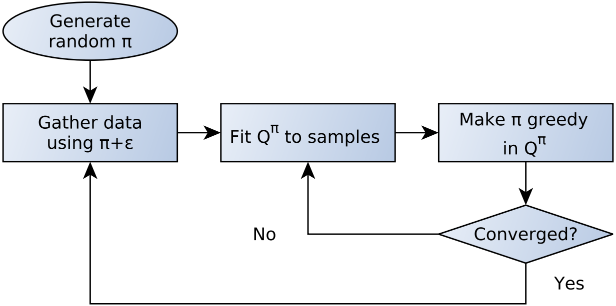
$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma Q(s',a') - Q(s,a)\right)$$

- Better convergence properties when using function approximation
- Safer during learning



Batch techniques

Q-learning and SARSA treat each sample only once, but that is inefficient. We can instead use all samples $(s_i, a_i) \rightarrow (r_i, s'_i)$ simultaneously (fitted Q-iteration)



Fitted Q iteration

function FITTEDQITERATION
  $k \leftarrow 0$, $\hat{Q}_k \leftarrow 0$
  repeat
    for each sample i **do**
      $\pi(s'_i) = \arg\max_a \hat{Q}_k(s'_i, a)$
      $\hat{Q}_{k+1}(s_i, a_i) \leftarrow r_i + \gamma \hat{Q}_k(s'_i, \pi(s'_i))$
    end for
    $k \leftarrow k + 1$
  until $\|\hat{Q}_k - \hat{Q}_{k-1}\| < \varepsilon$
  return $\hat{Q}_k$
end function

$\hat{Q}$ can be approximated by any supervised learning algorithm.

Least squares approximation

We know that for every sample $(s_i, a_i) \to (r_i, s_i')$, $\hat{Q}^\pi$ has to satisfy

$$\hat{Q}^\pi(s_i, a_i) = r_i + \gamma \hat{Q}^\pi(s_i', \pi(s_i'))$$

Given features $\phi(s, a)$ and a linear function approximator $\hat{Q}(s, a) = \phi(s, a)^T \theta$ with parameters θ, every sample i induces a constraint

$$\phi(s_i, a_i)^T \theta = r_i + \gamma \phi(s_i', \pi(s_i'))^T \theta, \text{such that}$$

$$\left(\phi(s_i, a_i)^T - \gamma \phi(s_i', \pi(s_i'))^T\right) \theta = r_i$$

In this case, we can estimate $\hat{Q}^\pi$ directly using least squares fitting, although we still need to iterate $\pi(s) \leftarrow \max_a \phi(s, a)^T \theta$ to get $\hat{Q}^*$.

Least squares approximation

We know that for every sample $(s_i, a_i) \to (r_i, s'_i)$, $\hat{Q}^\pi$ has to satisfy

$$\hat{Q}^\pi(s_i, a_i) = r_i + \gamma \hat{Q}^\pi(s'_i, \pi(s'_i))$$

Given features $\phi(s, a)$ and a linear function approximator $\hat{Q}(s, a) = \phi(s, a)^T \theta$ with parameters θ, every sample i induces a constraint

$$\phi(s_i, a_i)^T \theta = r_i + \gamma \phi(s'_i, \pi(s'_i))^T \theta, \text{such that}$$

$$\left(\phi(s_i, a_i)^T - \gamma \phi(s'_i, \pi(s'_i))^T\right) \theta = r_i$$

In this case, we can estimate $\hat{Q}^\pi$ directly using least squares fitting, although we still need to iterate $\pi(s) \leftarrow \max_a \phi(s, a)^T \theta$ to get $\hat{Q}^*$.

Least squares approximation

We know that for every sample $(s_i, a_i) \rightarrow (r_i, s_i')$, $\hat{Q}^\pi$ has to satisfy

$$\hat{Q}^\pi(s_i, a_i) = r_i + \gamma \hat{Q}^\pi(s_i', \pi(s_i'))$$

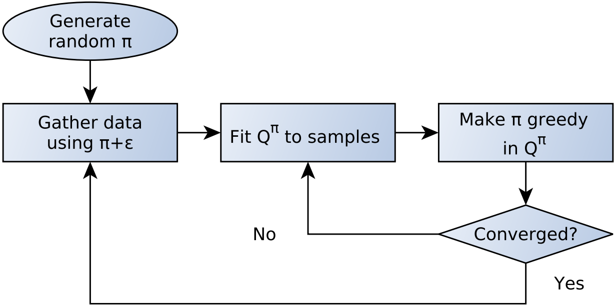
Given features $\phi(s,a)$ and a linear function approximator $\hat{Q}(s,a) = \phi(s,a)^T\theta$ with parameters θ, every sample i induces a constraint

$$\phi(s_i, a_i)^T\theta = r_i + \gamma\phi(s_i', \pi(s_i'))^T\theta, \text{such that}$$
$$\left(\phi(s_i, a_i)^T - \gamma\phi(s_i', \pi(s_i'))^T\right)\theta = r_i$$

In this case, we can estimate $\hat{Q}^\pi$ directly using least squares fitting, although we still need to iterate $\pi(s) \leftarrow \max_a \phi(s,a)^T\theta$ to get $\hat{Q}^*$.

Direct policy search

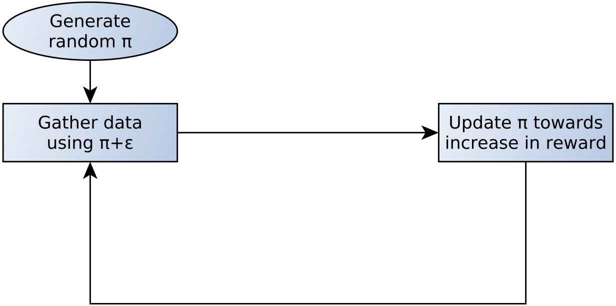
Why use a value function?



Policy search works directly on the parameters θ of a parameterized policy $\pi(s;\theta)$.

Direct policy search

Why use a value function?

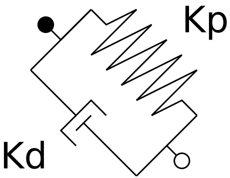


Policy search works directly on the parameters θ of a parameterized policy $\pi(s; \theta)$.

Policy parameterization

What form should the policy take?

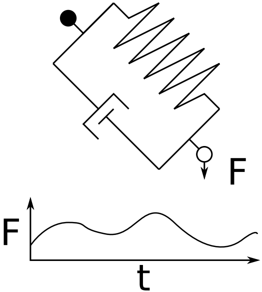
- Low-level
  - PD?
  - Dynamic movement primitives?
  - Radial basis functions?
  - Neural network?
- High-level
  - Via point placement?
  - Center of mass trajectory?
  - etc.



Policy parameterization

What form should the policy take?

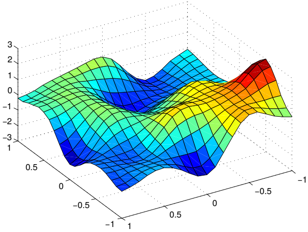
- Low-level
  - PD?
  - Dynamic movement primitives?
  - Radial basis functions?
  - Neural network?
- High-level
  - Via point placement?
  - Center of mass trajectory?
  - etc.



Policy parameterization

What form should the policy take?

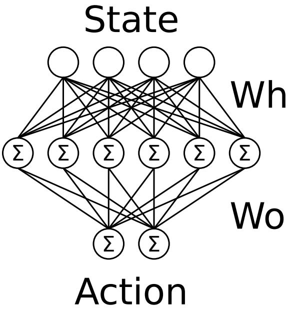
- Low-level
  - PD?
  - Dynamic movement primitives?
  - Radial basis functions?
  - Neural network?
- High-level
  - Via point placement?
  - Center of mass trajectory?
  - etc.



Policy parameterization

What form should the policy take?

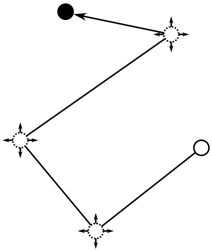
- Low-level
  - PD?
  - Dynamic movement primitives?
  - Radial basis functions?
  - Neural network?
- High-level
  - Via point placement?
  - Center of mass trajectory?
  - etc.



Policy parameterization

What form should the policy take?

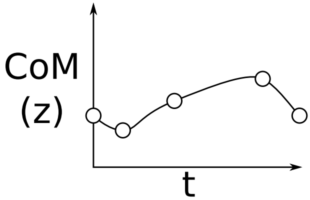
- Low-level
  - PD?
  - Dynamic movement primitives?
  - Radial basis functions?
  - Neural network?
- High-level
  - Via point placement?
  - Center of mass trajectory?
  - etc.



Policy parameterization

What form should the policy take?

- Low-level
  - PD?
  - Dynamic movement primitives?
  - Radial basis functions?
  - Neural network?
- High-level
  - Via point placement?
  - Center of mass trajectory?
  - etc.



Black-box PI^2

Black-box optimization routine, competitive with state-of-the-art Markov-based schemes.

function BLACKBOXPI^2
  $g \leftarrow 0$
  Initialize θ_0 randomly
  repeat
    for k in K **do**
      $\varepsilon_k = \mathcal{N}(0, \Sigma)$, $R_k = \text{rollout}(\theta_g + \varepsilon_k)$
    end for
    for k in K **do**
      $w_k = \frac{e^{\frac{1}{K} R_k}}{\sum_{k=1}^{K} e^{\frac{1}{K} R_k}}$
    end for
    $\theta_{g+1} = \theta_g + \sum_{k=1}^{K} w_k(\theta_g + \varepsilon_k)$
    $g \leftarrow g + 1$
  until converged
end function

Policy search versus TD learning

Advantages

- Smoother policies
- Parameter space could be smoother
- Parameter space is generally smaller

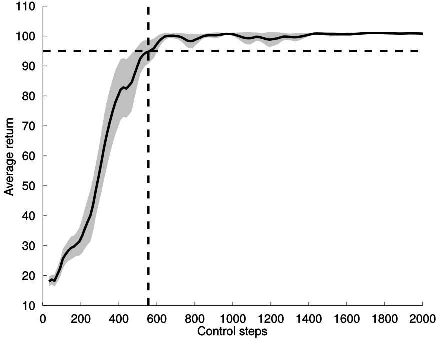
Disadvantages

- Parameterization requires system knowledge
- Only locally optimal (may get stuck in local minima)
- Rollouts are noisy

Outline

Sample complexity

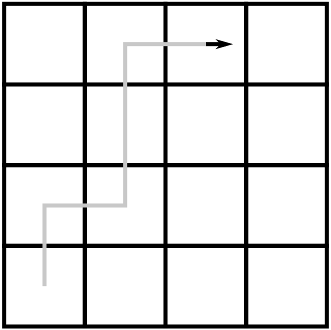
Sample complexity is the number of samples required to learn the task (interaction time).



Does not include computation time (computational complexity)

Eligibility traces

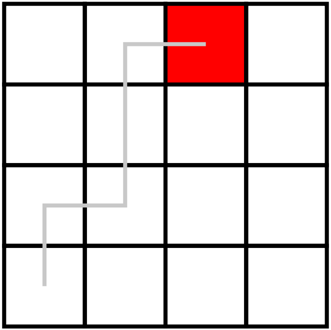
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

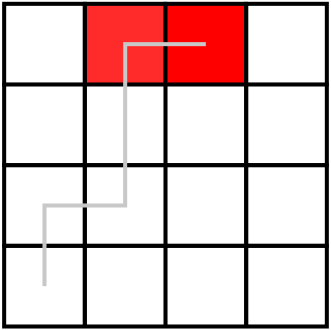
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

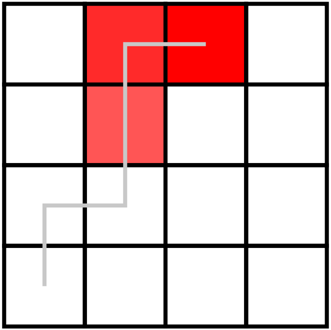
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

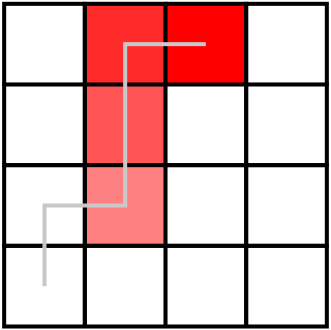
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

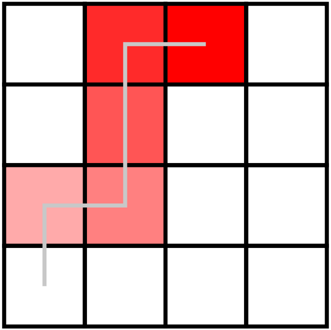
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

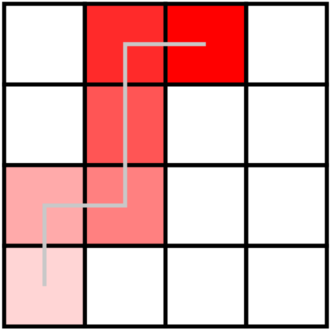
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

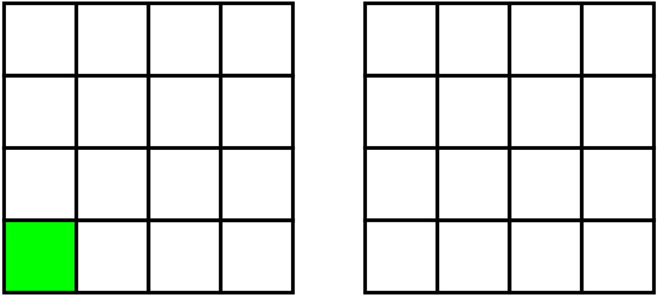
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

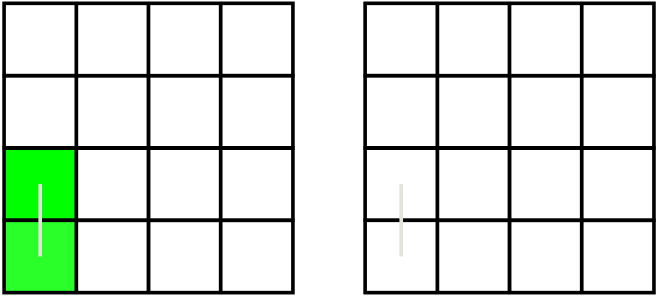
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

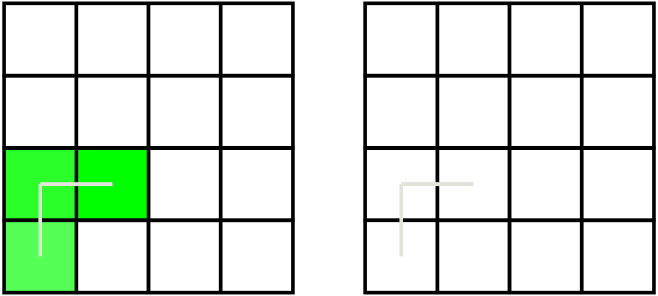
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

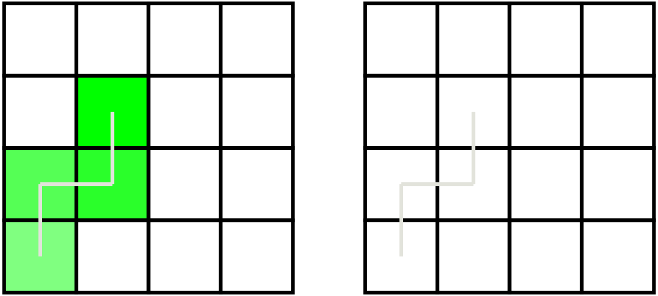
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

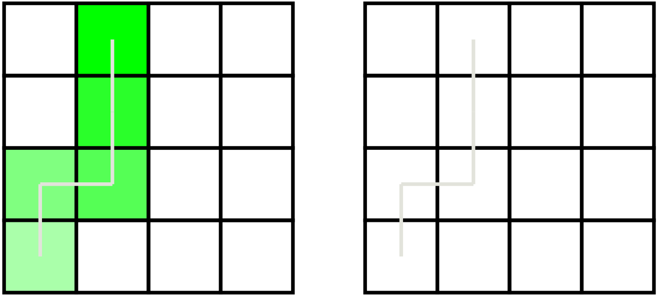
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

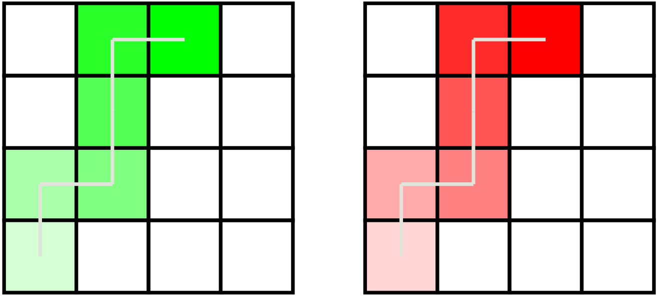
Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Eligibility traces

Standard TD methods update only the current state.



Eligibility trace methods keep a list of recently visited, eligible states, and update based on the eligibility.

Value function approximation

Robots work in continuous state spaces. How to discretize the value function?

- LEO: 12-dimensional state-action space. Assume each discretized in N steps $\rightarrow N^{12}$ values!
- Say $N = 10$, `float` values $\rightarrow$ 4 TB memory to be stored and filled.

Generalization

To be sample-efficient, experience must be generalized.
Generalization always has a bias-variance trade-off.

Value function approximation

Robots work in continuous state spaces. How to discretize the value function?

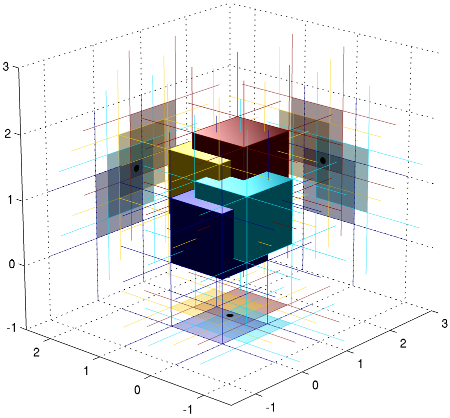
- LEO: 12-dimensional state-action space. Assume each discretized in N steps $\rightarrow N^{12}$ values!
- Say $N = 10$, `float` values $\rightarrow$ 4 TB memory to be stored and filled.

Generalization

To be sample-efficient, experience must be generalized.
Generalization always has a bias-variance trade-off.

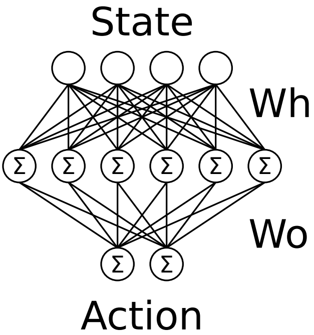
Tile coding

Linear function approximator using overlapping grids.



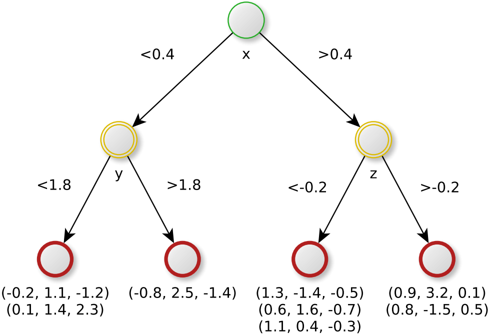
Convergence guarantees due to linearity $\hat{Q}(s) = \phi(s,a)^T \theta$.

Neural networks



- Less parameters (faster generalization)
- Updates have global, non-linear effects (no convergence guarantees, *forgetting*)

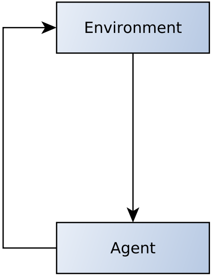
Trees



- Local changes
- *Forests* provide finer granularity and variance estimation.

Indirect reinforcement learning

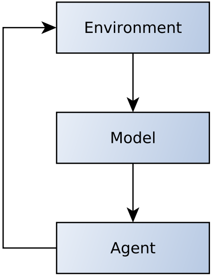
Direct reinforcement learning solves MDPs by interacting directly with the environment



Indirect reinforcement learning learns a model as an intermediary step.

Indirect reinforcement learning

Direct reinforcement learning solves MDPs by interacting directly with the environment



Indirect reinforcement learning learns a model as an intermediary step.

Model approximation

Generalizes in a different space than value function approximation. It approximates the transition function $p(s'|s,a) = \mathscr{T}(s,a,s')$. Since in a Markov process this is static, it is a supervised learning problem.

- Allows simulating additional transitions (mental rehearsal)
- Provides an estimate of $\frac{\partial s}{\partial a}$ for better policy updates
- Allows derivation of π from state-value function V instead of state-action value function Q.

Model learning versus system identification

Model learning is performed on-line and therefore has an exploration-exploitation trade-off. Model fidelity should be high along the trajectory of the optimal policy.

Model approximation

Generalizes in a different space than value function approximation. It approximates the transition function $p(s'|s,a) = \mathscr{T}(s,a,s')$. Since in a Markov process this is static, it is a supervised learning problem.

- Allows simulating additional transitions (mental rehearsal)
- Provides an estimate of $\frac{\partial s}{\partial a}$ for better policy updates
- Allows derivation of π from state-value function V instead of state-action value function Q.

Model learning versus system identification

Model learning is performed on-line and therefore has an exploration-exploitation trade-off. Model fidelity should be high along the trajectory of the optimal policy.

Model approximation

Generalizes in a different space than value function approximation. It approximates the transition function $p(s'|s,a) = \mathscr{T}(s,a,s')$. Since in a Markov process this is static, it is a supervised learning problem.

- Allows simulating additional transitions (mental rehearsal)
- Provides an estimate of $\frac{\partial s}{\partial a}$ for better policy updates
- Allows derivation of π from state-value function V instead of state-action value function Q.

Model learning versus system identification

Model learning is performed on-line and therefore has an exploration-exploitation trade-off. Model fidelity should be high along the trajectory of the optimal policy.

Locally weighted regression

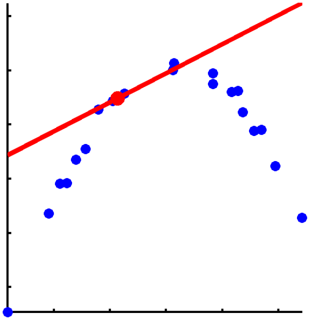
Very successful model approximator for robotics

- Find *k* nearest neighbors
  - Approximate nearest neighbor search
- Weigh according to distance

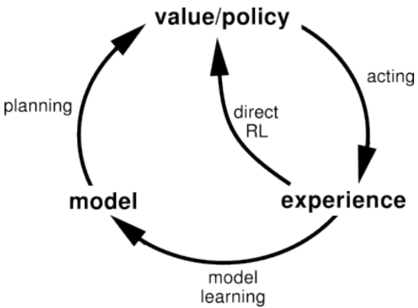
$$w(p) = e^{-\left(\frac{|p-q|_2}{h}\right)^2}$$

where *h* is the distance to the *k*th nearest neighbor

- Fit linear model using least-squares regression



DYNA

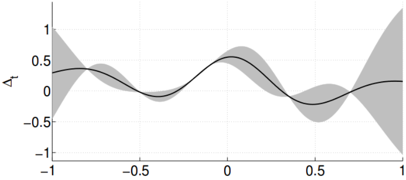


[Sutton & Barto, 1998]

- Perform K simulated updates per control step
- Mix of direct and indirect reinforcement learning
- Simulated updates can be done anywhere (prioritized sweeping)

PILCO

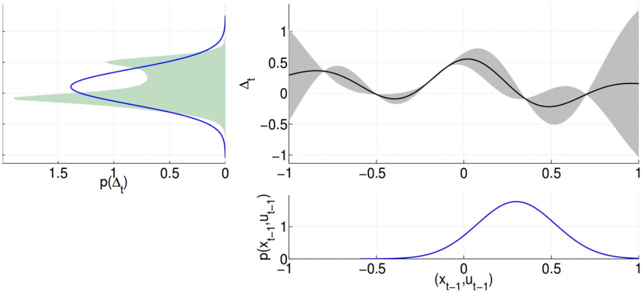
Model-learning policy search. By approximating the system dynamics with a gaussian process model, can analytically calculate $\frac{dR}{d\theta}$.



[Deisenroth & Rasmussen, 2011]

PILCO

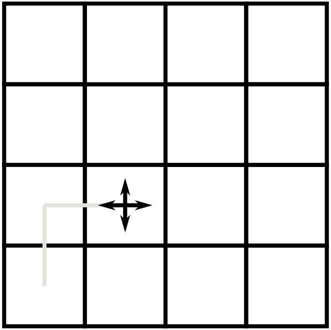
Model-learning policy search. By approximating the system dynamics with a gaussian process model, can analytically calculate $\frac{dR}{d\theta}$.



[Deisenroth & Rasmussen, 2011]

Local bias

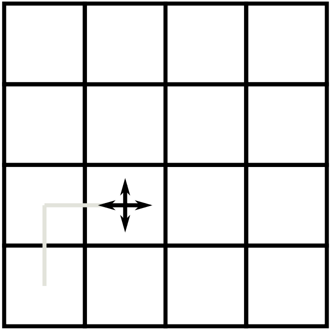
When controlling a system, we always want to find the best action for the current state.



Therefore it makes sense to concentrate planning around the current state

Local bias

When controlling a system, we always want to find the best action for the current state.



Therefore it makes sense to concentrate planning around the current state

Some algorithms that use local bias

	RTDP	TEXPLORE	DYNA-2
Model	Given	Learned	Learned
Planning	Best-first search	Monte-carlo tree search (UCT)	TD learning
Value function	Updated from experience + model	Updated from model	Separate for experience + model
Control	From value function	From UCT	From weighted sum of value functions

Recap

- Reinforcement learning is model-free optimal control
- Value-based RL is sampled dynamic programming
  - First estimates a value function, after which the optimal policy is one-step greedy
- Policy search directly searches in the space of parameterized policies, can solve larger problems, but is only locally optimal
  - Requires suitable initialization
- Many approaches exist to make RL practical on real systems

Hands-on

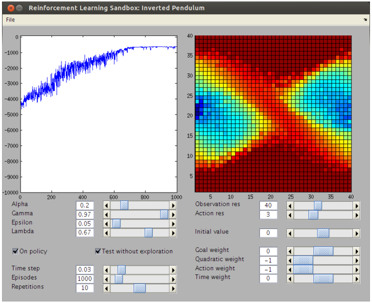
Experiment with parameter settings for reinforcement learning for the pendulum swing-up problem

Manual
http://wouter.caarls.org/files/kss2014_tutorial.pdf

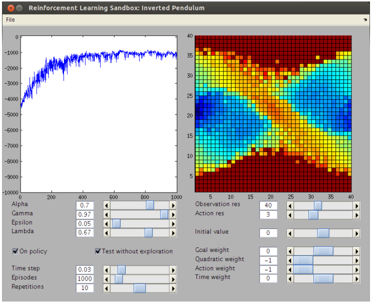
Matlab toolbox
http://wouter.caarls.org/files/kss2014_tutorial.tgz

Run `pendgui` from the toolbox directory

Q1: Learning rate α



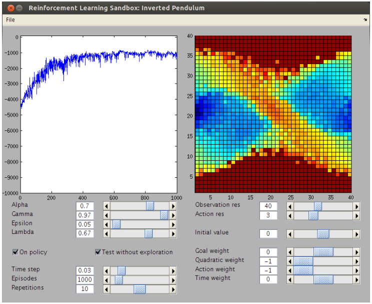
$\alpha = 0.2$



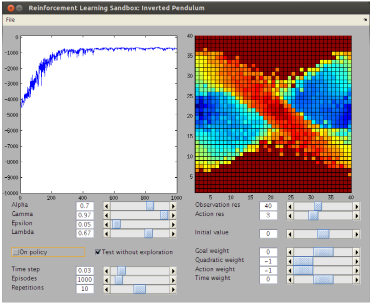
$\alpha = 0.7$

Q2: On-policy vs off-policy learning

$$\varepsilon = 0.05$$



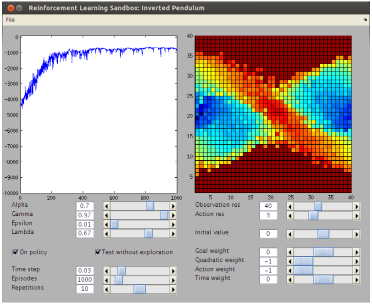
On-policy



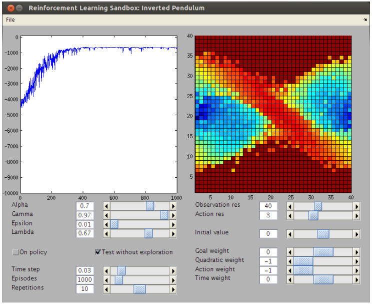
Off-policy

Q2: On-policy vs off-policy learning

$$\varepsilon = 0.01$$



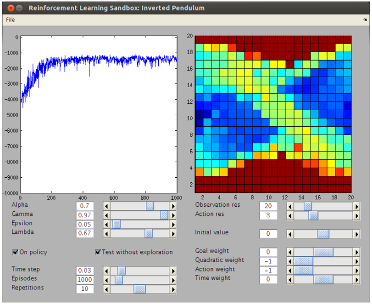
On-policy



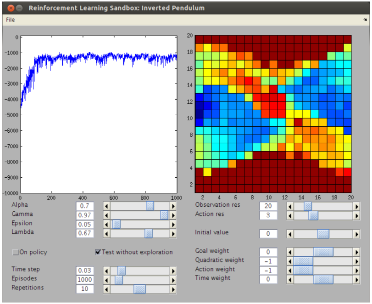
Off-policy

Q3: State space resolution

$$\alpha = 0.7$$



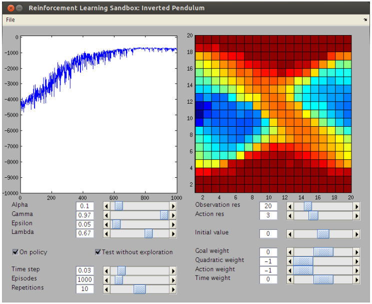
On-policy

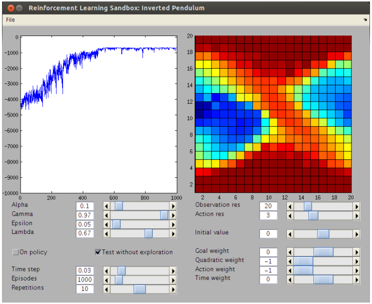


Off-policy

Q3: State space resolution

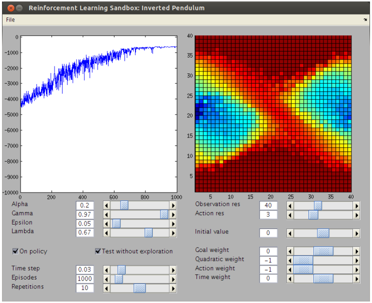
$$\alpha = 0.1$$



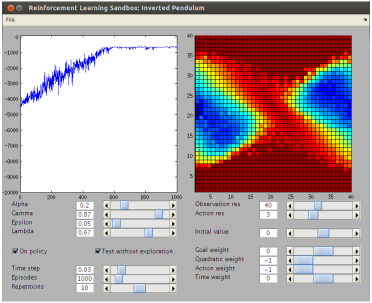


On-policy Off-policy

Q4: Discount rate γ

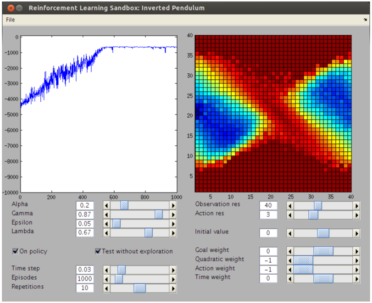


$\gamma = 0.97$

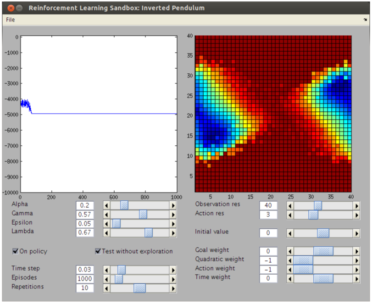


$\gamma = 0.87$

Q5: Discount rate γ (2)



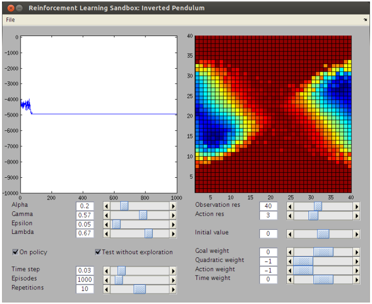
$\gamma = 0.87$



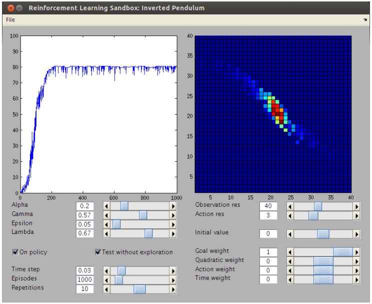
$\gamma = 0.57$

Q6: Reward function

$$\gamma = 0.57$$



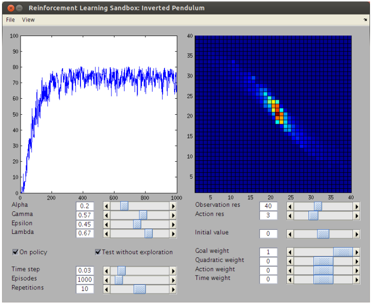
Path rewards



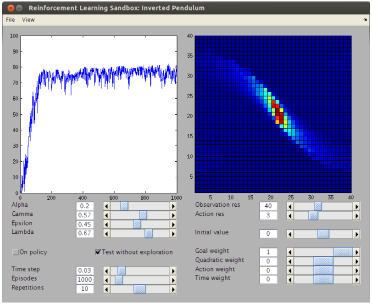
Goal rewards

Q7: On-policy vs off-policy learning (2)

$\gamma = 0.57$, goal rewards, $\varepsilon = 0.45$



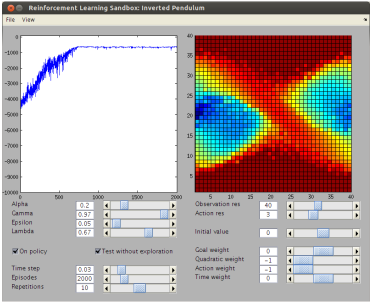
On-policy



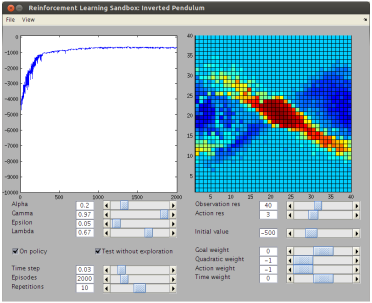
Off-policy

Q8: Initial value

Episodes = 2000



Initial value 0



Initial value -500