Reinforcement Learning

George Konidaris
gdk@cs.duke.edu

DUKE COMPUTER SCIENCE

Spring 2016

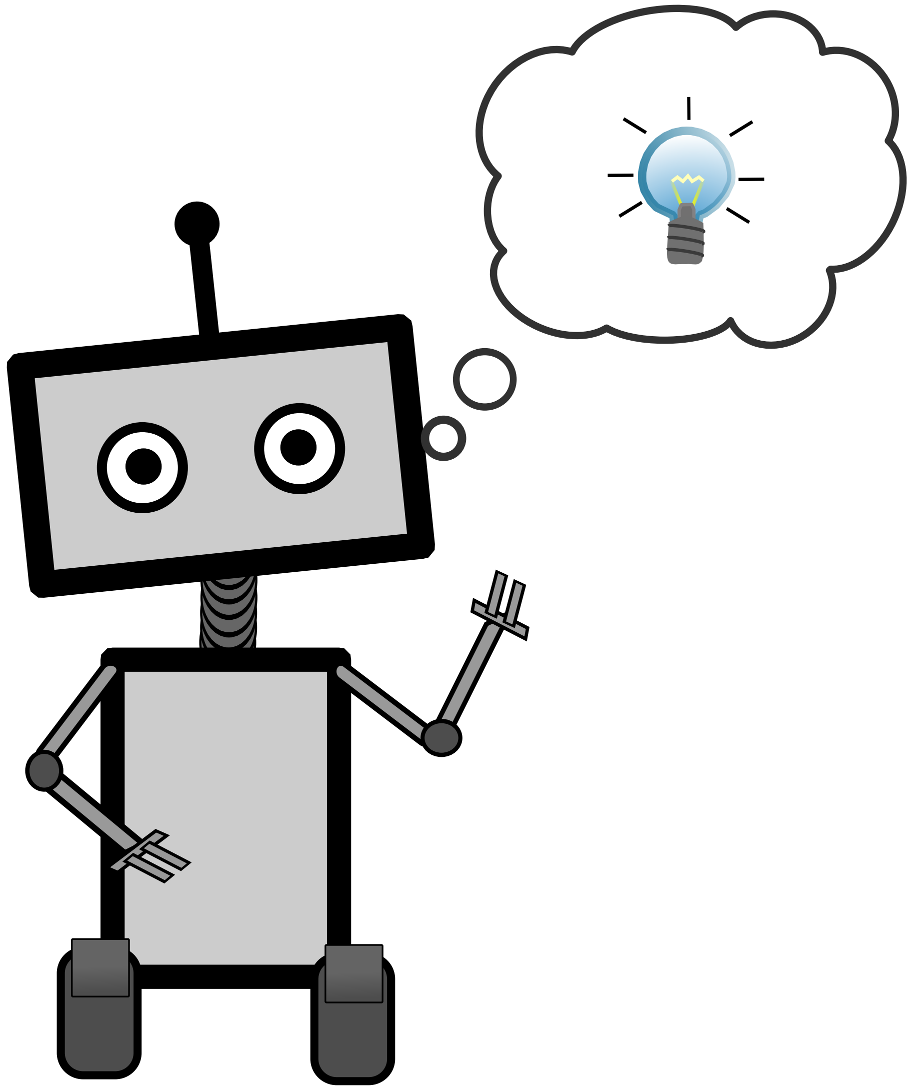
Machine Learning

Subfield of AI concerned with *learning from data*.

Broadly, using:

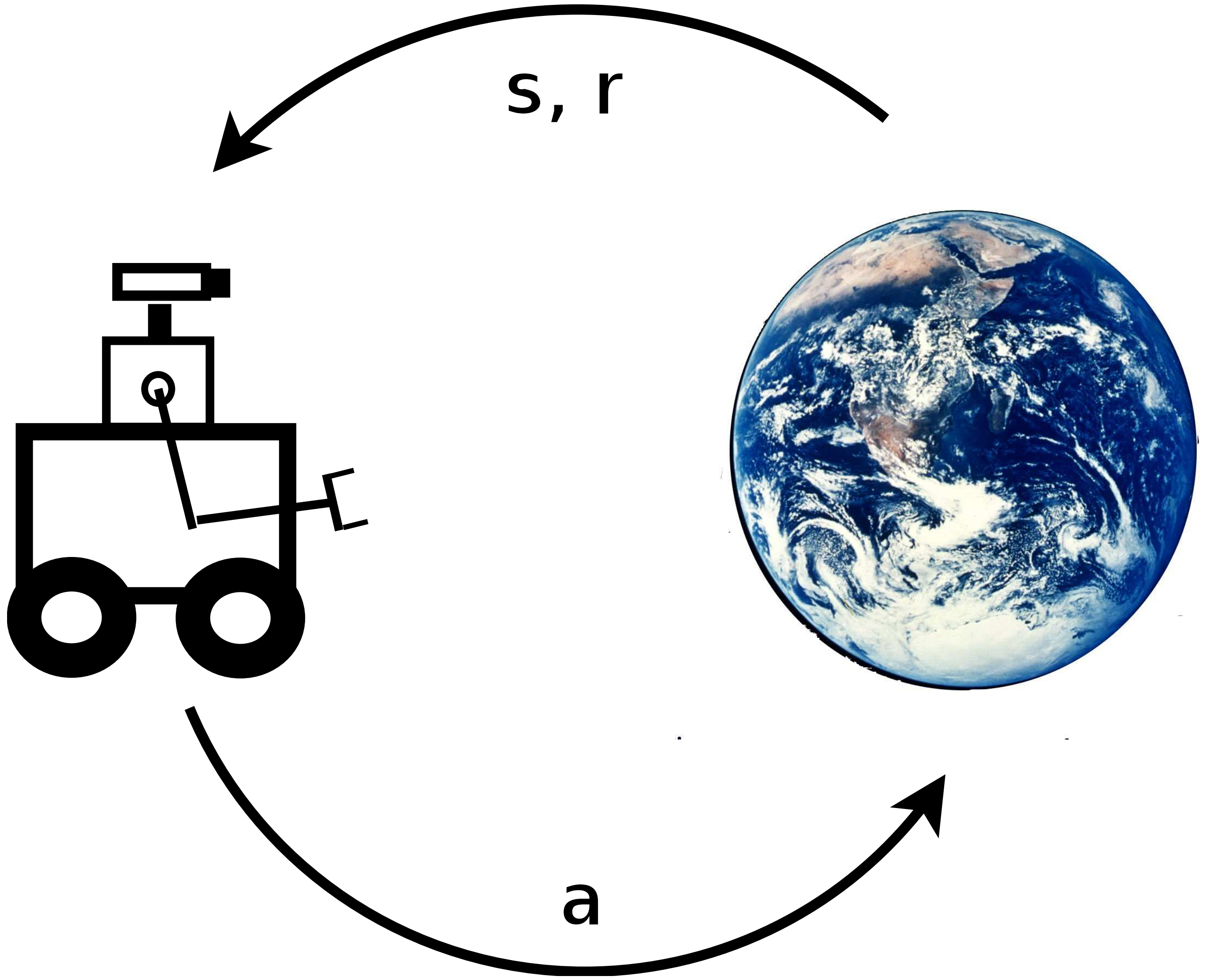
- ***Experience***
- To Improve ***Performance***
- On Some ***Task***

(Tom Mitchell, 1997)



Reinforcement Learning

Learning counterpart of planning.

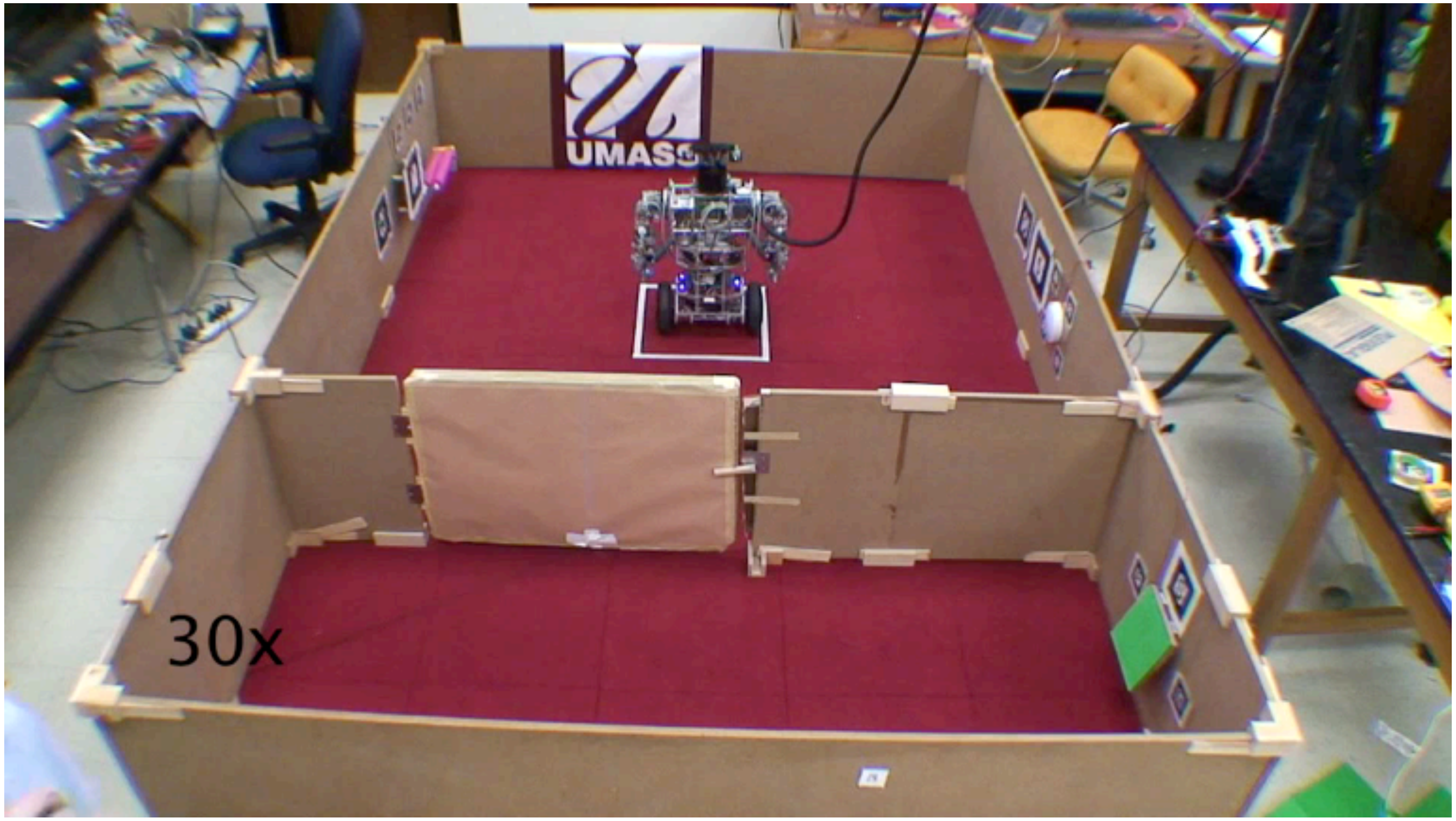


$\pi : S \rightarrow A$

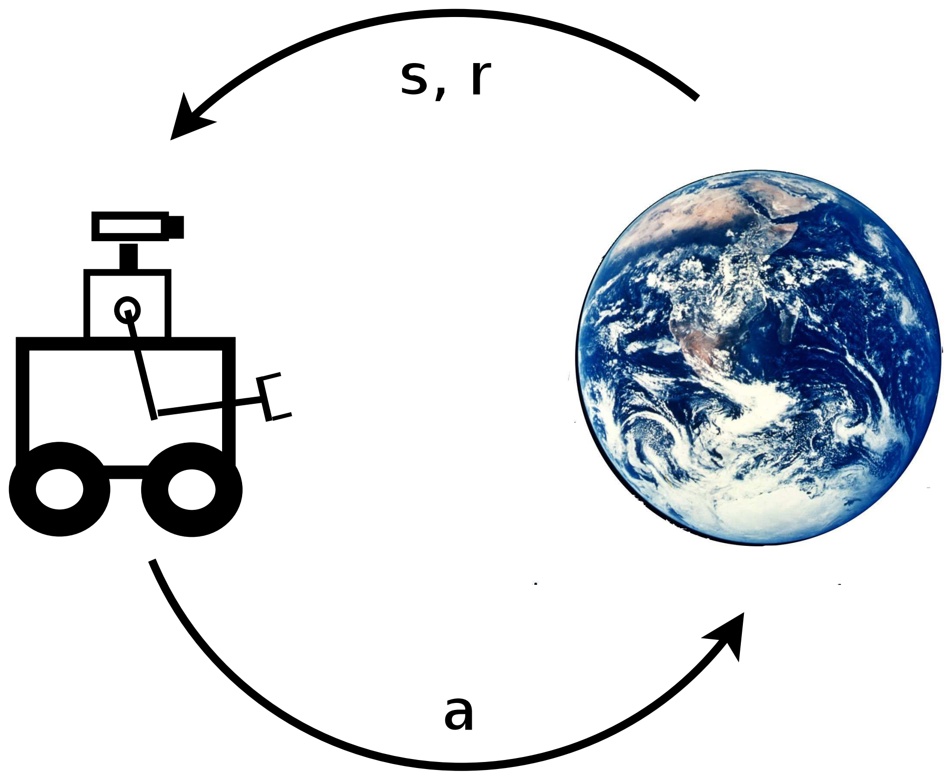
$$\max_{\pi} R = \sum_{t=0}^{\infty} \gamma^t r_t$$

RL

The problem of **learning how to interact** with an environment to **maximize reward**.



RL



Agent interacts with an environment

At each time t:

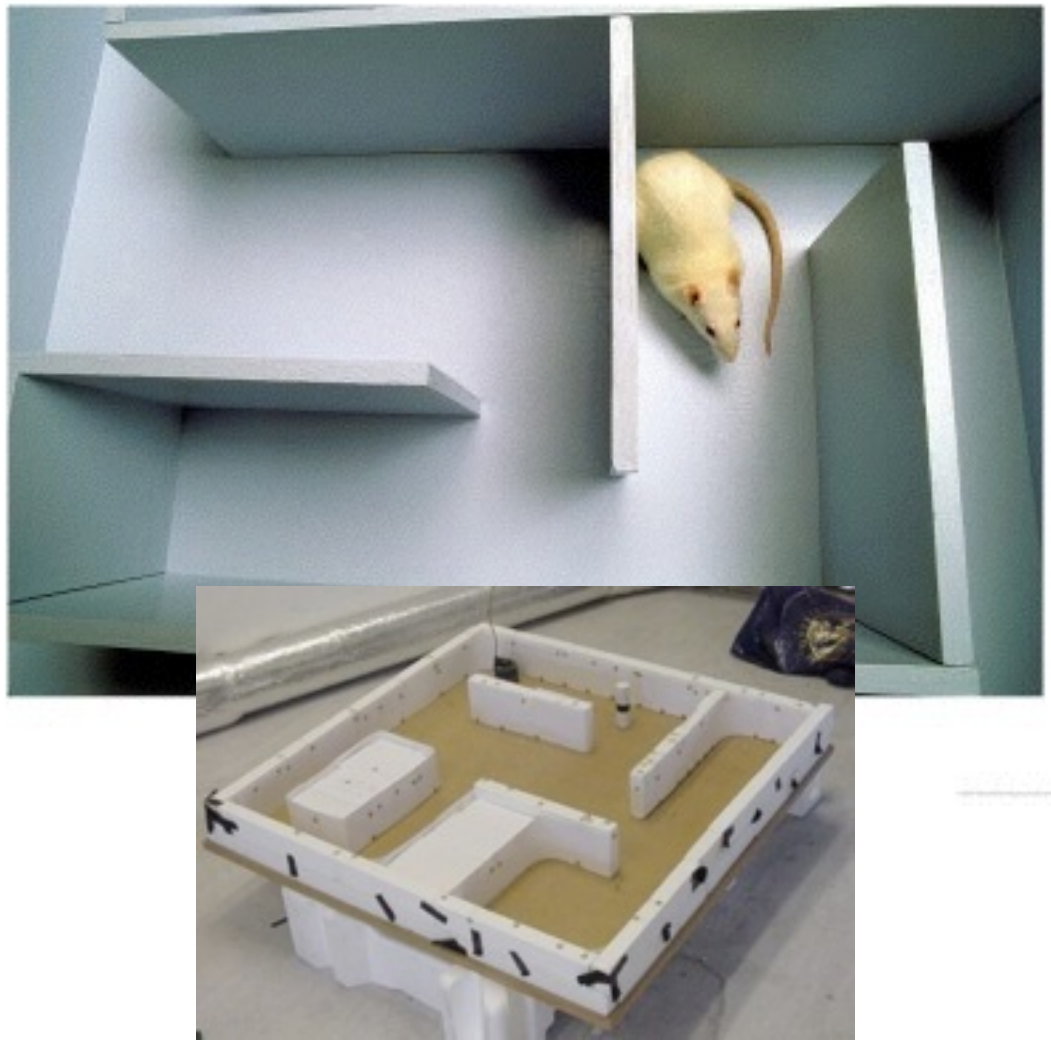
- Receives sensor signal s_t
- Executes action a_t
- *Transition*:
  - new sensor signal s_{t+1}
  - reward r_t

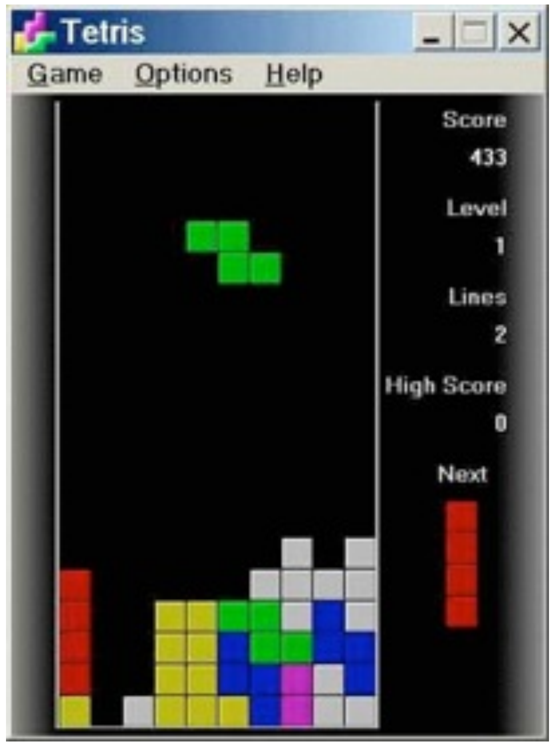
Goal: find policy π that maximizes expected return (sum of discounted future rewards):

$$\max_{\pi} \mathbb{E}\left[R = \sum_{t=0}^{\infty} \gamma^t r_t\right]$$

RL

This formulation is general enough to encompass a wide variety of learned control problems.





Markov Decision Processes

S : set of *states*

A : set of actions

γ : discount factor

$$< S, A, \gamma, R, T >$$

R : reward function

$R(s, a, s')$ is the reward received taking action a from state s and transitioning to state s'.

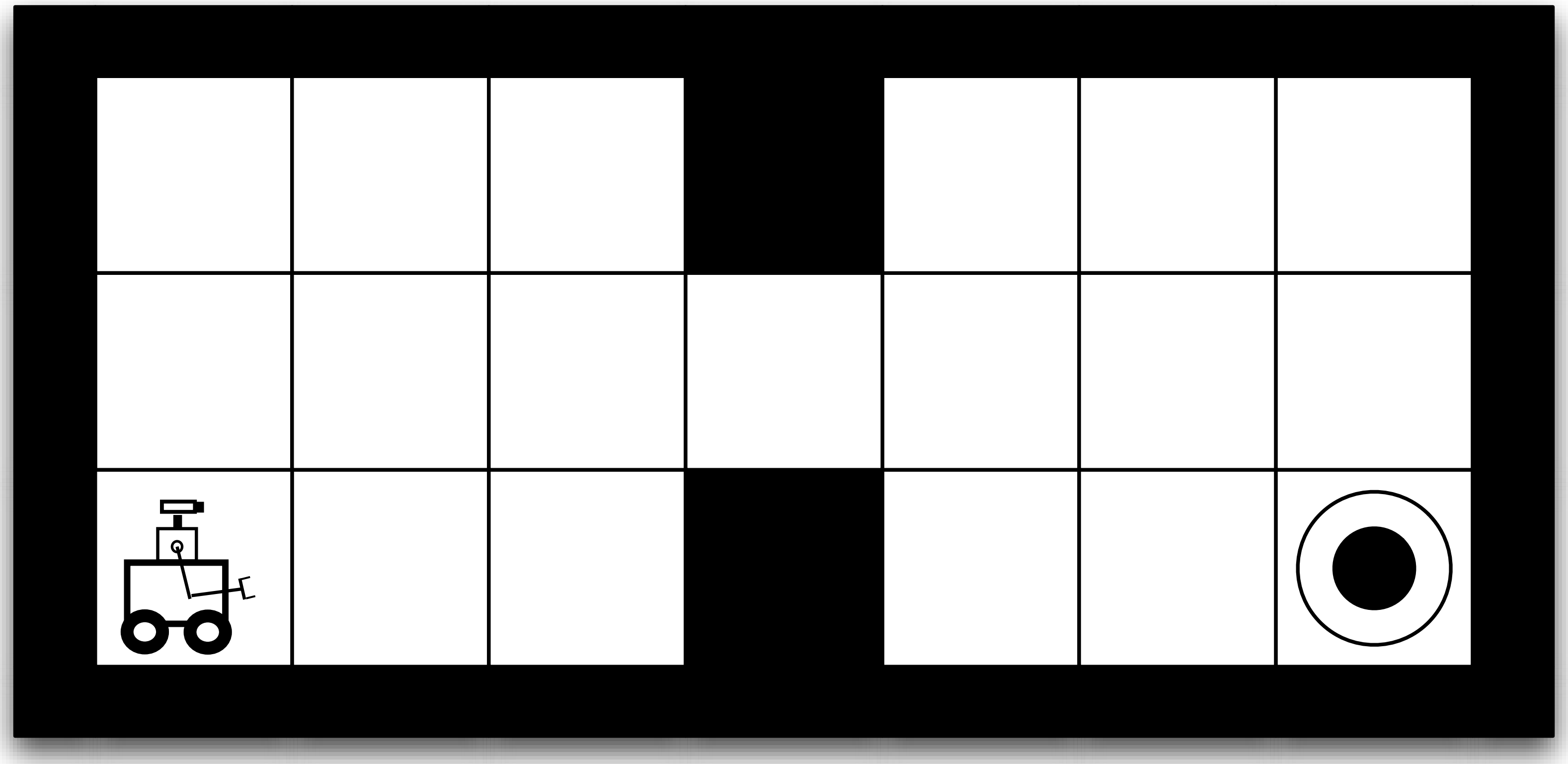
T : transition function

$T(s'|s, a)$ is the probability of transitioning to state s' after taking action a in state s.

<u>RL: *one or both of T, R unknown.*</u>

RL

Example:



States: set of grid locations

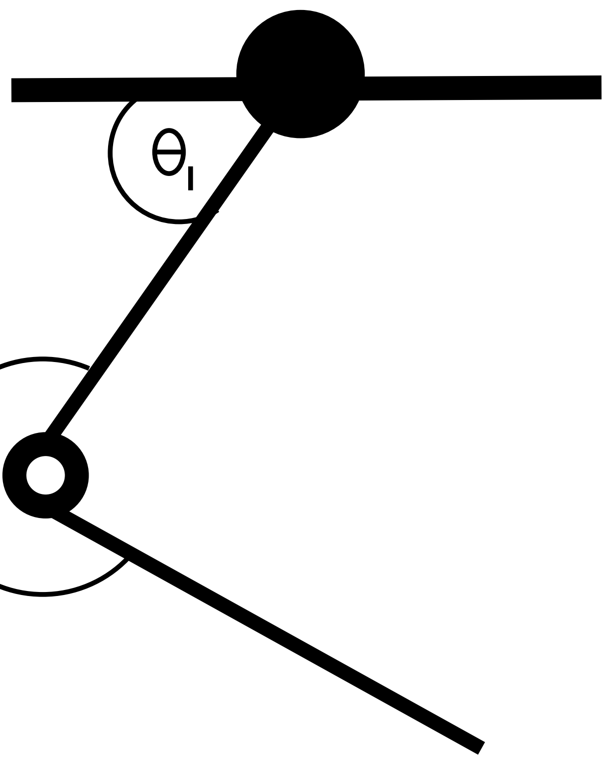
Actions: up, down, left, right

Transition function: move in direction of action with p=0.9

Reward function: -1 for every step, 1000 for finding the goal

RL

Example:



States: $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ (real-valued vector)
Actions: +1, -1, 0 units of torque added to elbow
Transition function: physics!
Reward function: -1 for every step

MDPs

Our target is a **policy**:

$$\pi : S \rightarrow A$$

A policy maps *states* to *actions*.

The **optimal policy** maximizes:

$$\max_{\pi} \forall s, \mathbb{E}\left[R(s) = \sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s \right]$$

This means that we wish to find a policy that maximizes the **return** from **every state.**

Value Functions

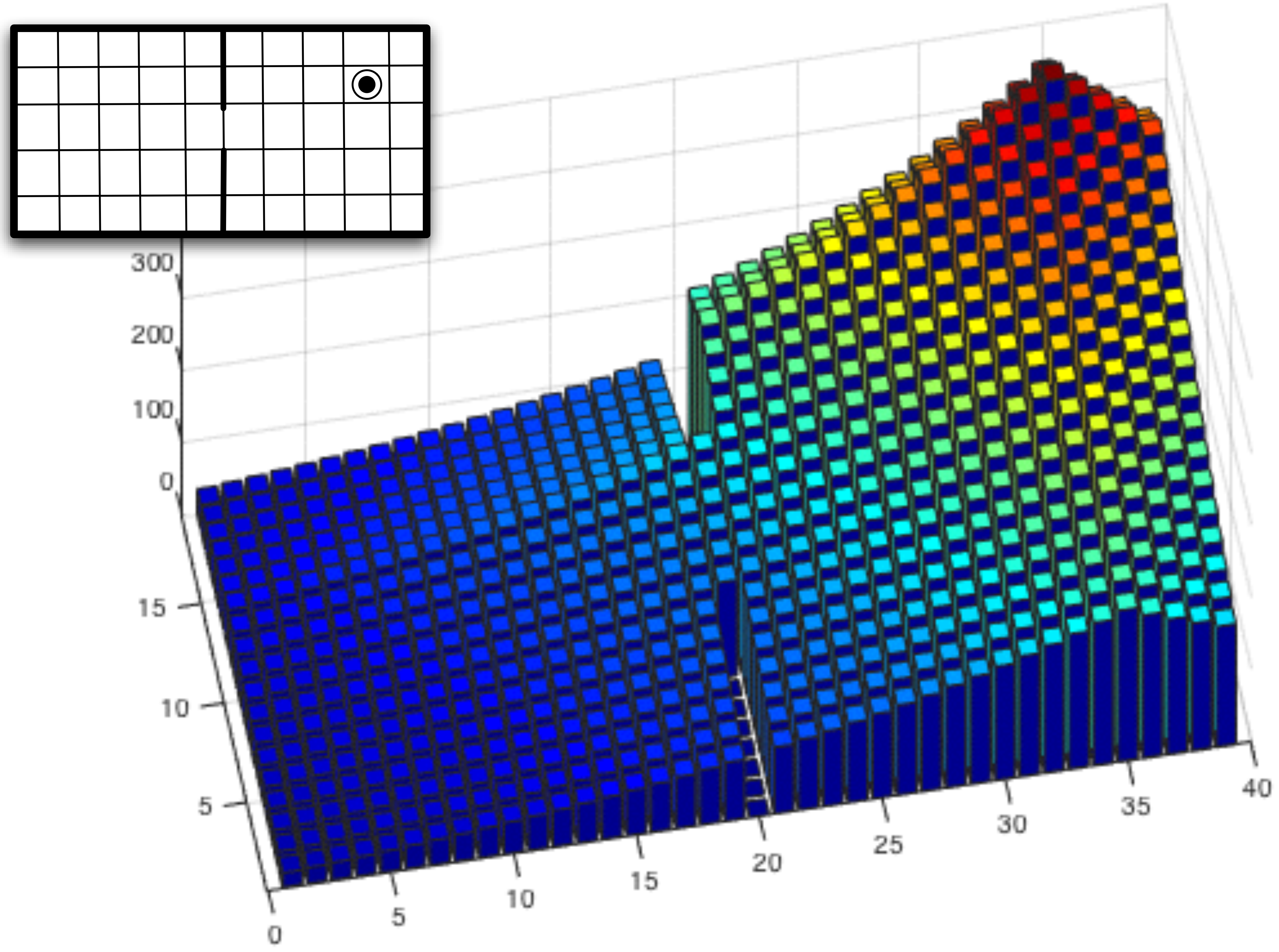
Given a policy, we can estimate of $R(s)$ for every state.

- This is a *value function*.
- It can be used to improve our policy.

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, s_0 = s\right]$$

This is the value of state s **under policy** π.

Value Functions



Value Functions

Similarly, we define a *state-action value function* as follows:

$$Q_\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, s_0 = s, a_0 = a\right]$$

This is the value of executing a in state s, *then following* π.

Note that:

$$Q_\pi(s, \pi(s)) = V_\pi(s)$$

Policy Iteration

Recall that we seek the policy that maximizes $V_\pi(s), \forall s$.

Therefore we know that, for the optimal policy π^*:

$$V_{\pi^*}(s) \geq V_\pi(s), \forall \pi, s$$

$$Q_{\pi^*}(s,a) \geq Q_\pi(s,a), \forall \pi, s, a$$

This means that any change to π that increases Q anywhere obtains a better policy.

Policy Iteration

This leads to a general policy improvement framework:

1. Start with a policy π
2. Learn Q_π
3. Improve π
    a. $\pi(s) = \max_a Q(s, a), \forall s$

Repeat

This is known as **policy iteration**.
It is guaranteed to converge to the optimal policy.

Steps 2 and 3 can be interleaved as rapidly as you like.
Usually, perform 3a *every time step*.

Value Function Learning

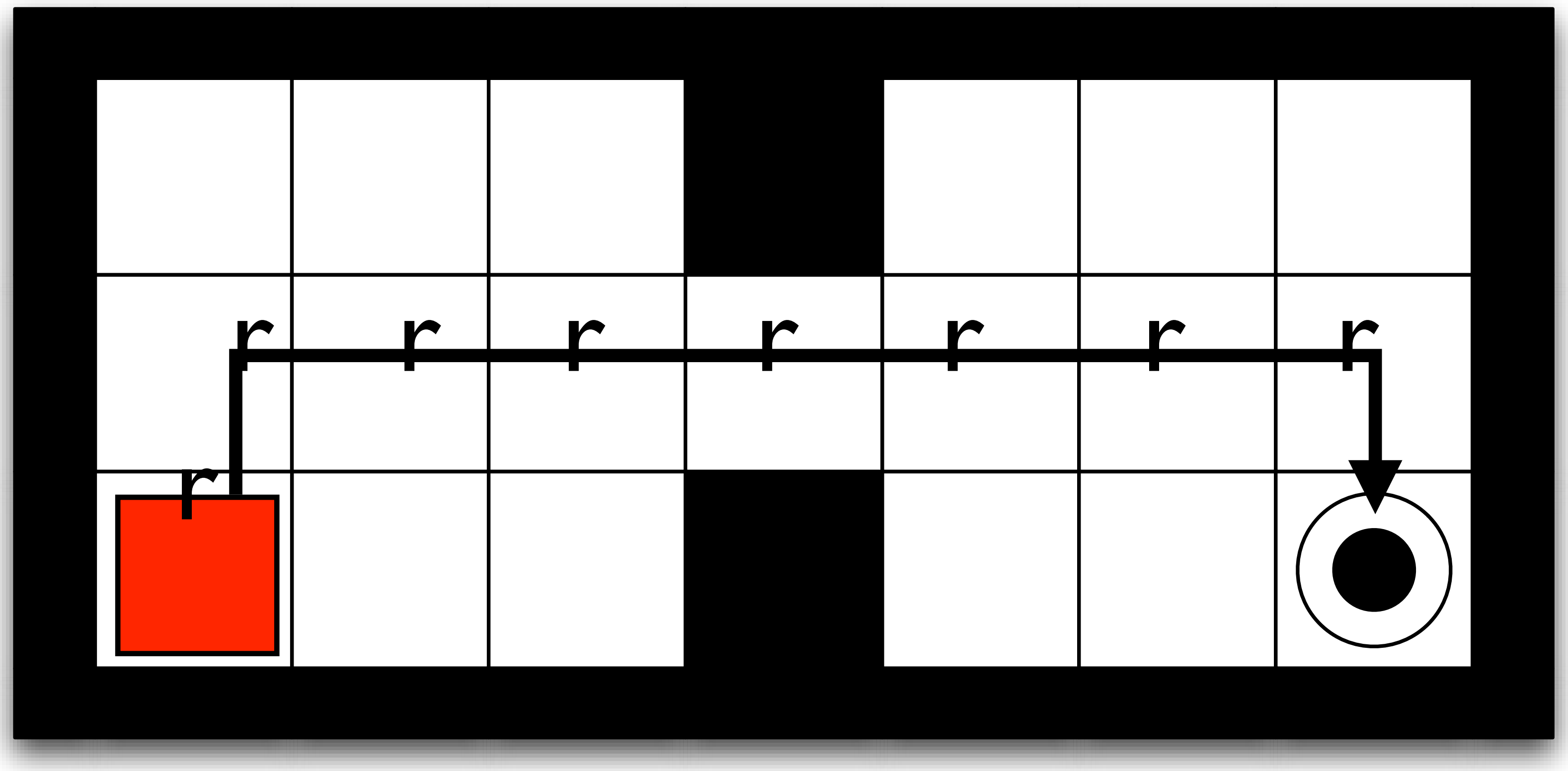
Learning proceeds by gathering samples of $Q(s, a)$.

Methods differ by:

- How you get the samples.
- How you use them to update Q .

Monte Carlo

Simplest thing you can do: sample $R(s)$.



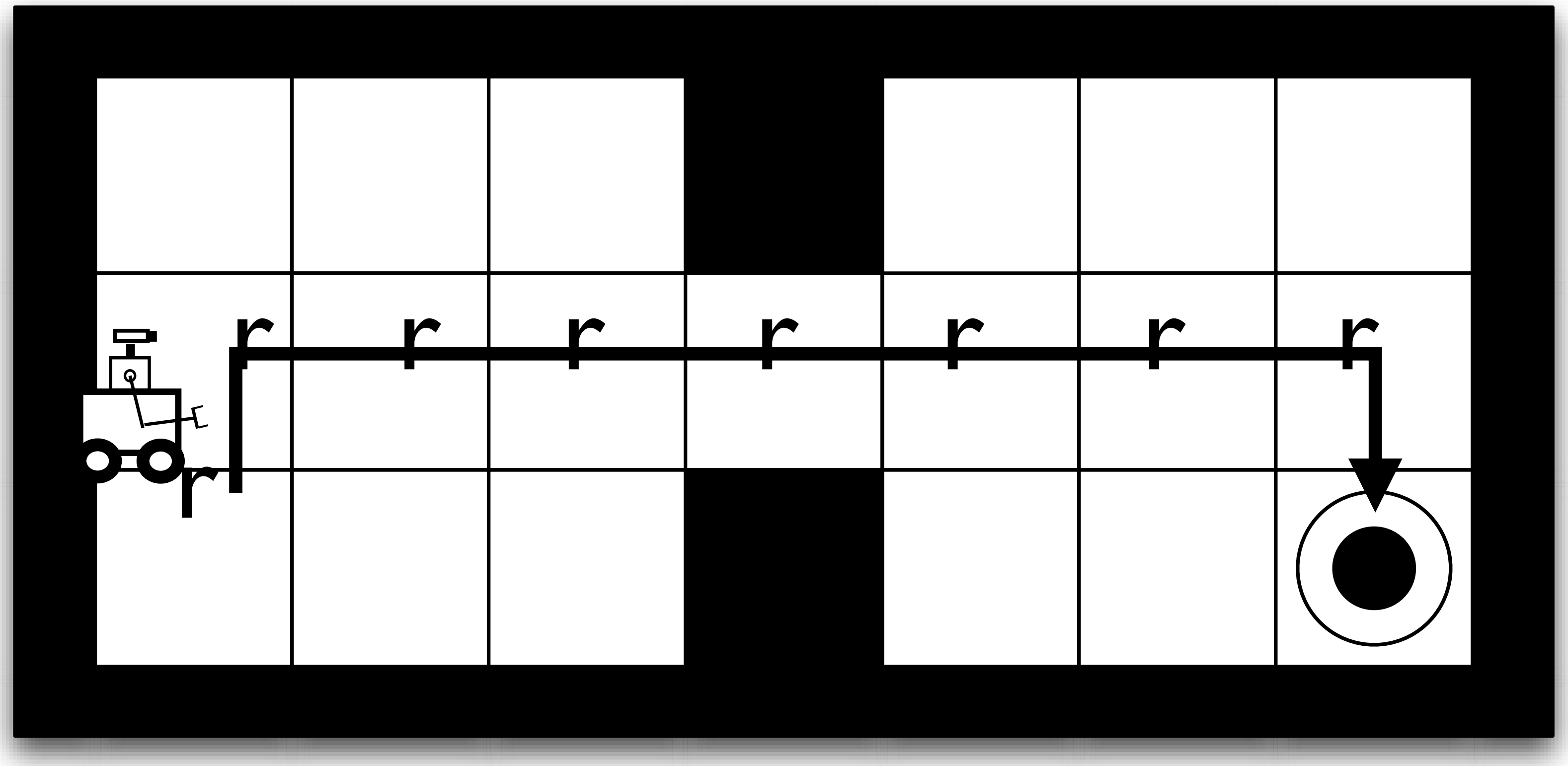
Do this repeatedly, average values:

$$Q(s,a) = \frac{R_1(s) + R_2(s) + ... + R_n(s)}{n}$$

Temporal Difference Learning

Where can we get more (immediate) samples?

Idea: *there is an important relationship between temporally successive states.*



TD Learning

Ideally and in expectation:

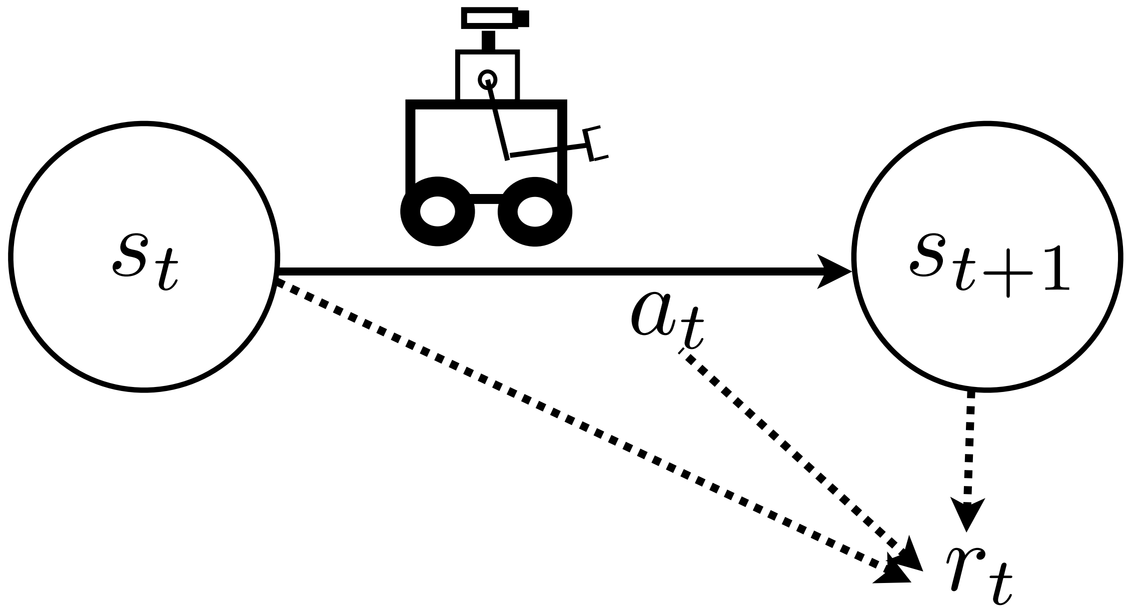
$$r_t + \gamma V(s_{t+1}) - V(s_t) = 0$$

V is correct if this holds in expectation for all states.

When it does not, it is known as a *temporal difference error.*

TD Learning

What does this look like?



$$V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$$
$$Q(s_t, a_t) \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1})$$

Sarsa

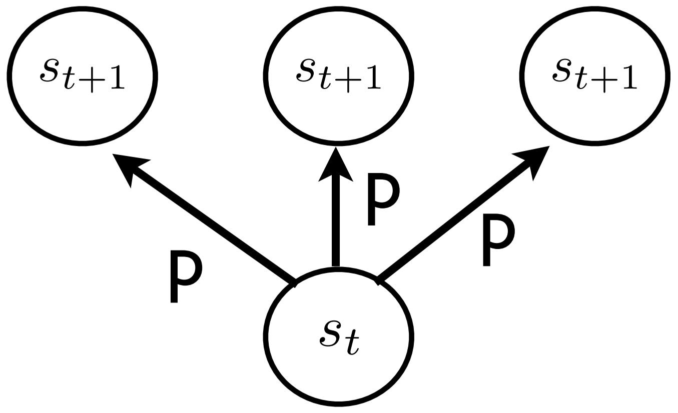
Sarsa: very simple algorithm

1. Initialize Q(s, a)
2. For *n* episodes
   - observe transition (s, a, r, s', a')
   - compute TD error $\delta = r + \gamma Q(s', a') - Q(s, a)$
   - update Q: $Q(s, a) = Q(s, a) + \alpha\delta$
   - select and execute action based on Q

TD

In Sarsa, we use a sample transition: (s, a, r, s', a')
This is a *sample backup*.

Given T, could replace with the full expectation:



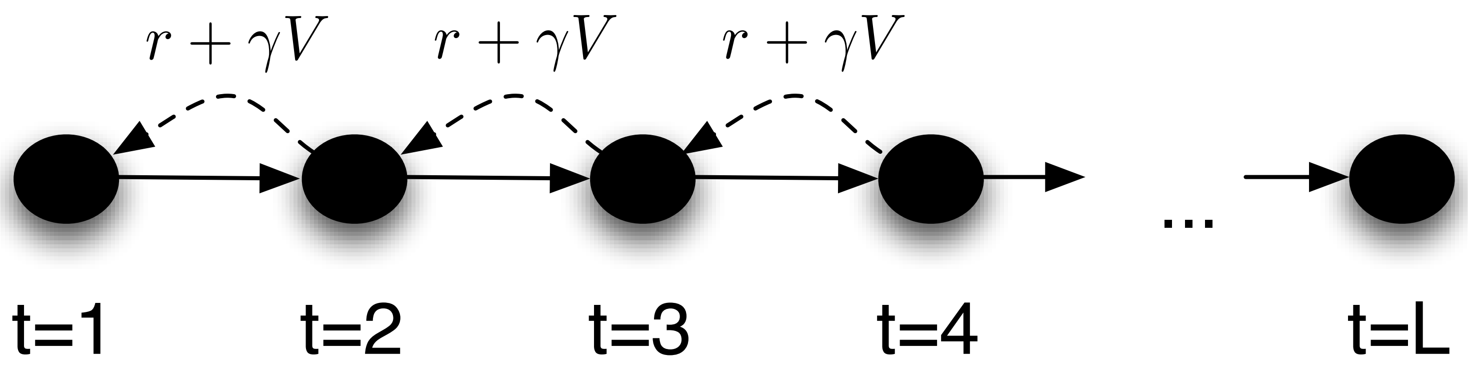
$$\delta = \mathbb{E}_{\pi,T}\left[r + \gamma Q(s', a')\right] - Q(s, a)$$

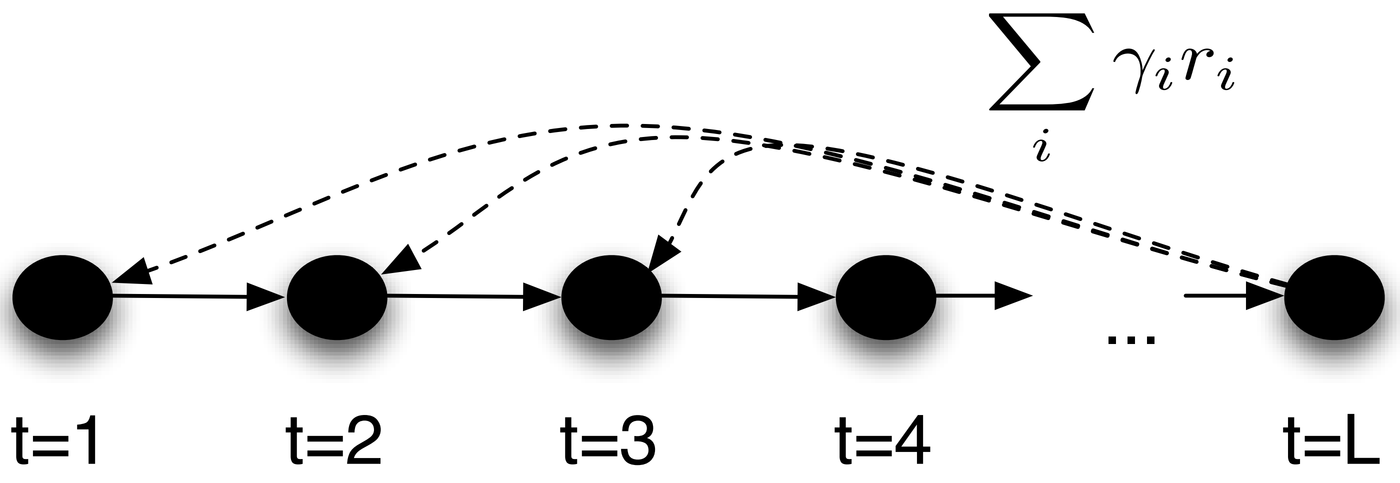
This is known as a *full backup* - **dynamic programming.**

Finds an optimal policy in time polynomial in $|S|$ and $|A|$.
(There are $|A|^{|S|}$ possible policies.)

TD vs. MC

TD and MC two extremes of obtaining samples of Q:





Generalizing TD

We can generalize this to the idea of an n-step rollout:

$$R_{s_t}^{(n)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n})$$

Each tells us something about the value function.

- We can combine *all* n-step rollouts.
- This is known as a *complex backup*.

TD(λ)

Weighted sum:

$$R^{(1)} = r_0 + \gamma V(s_1)$$
$$R^{(2)} = r_0 + \gamma r_1 + \gamma^2 V(s_2)$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$R^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_i + \gamma^n V(s_n)$$

$$\begin{tikzcd} 1 & \\ \lambda & \\ & \text{weights} \arrow[uul] \arrow[ul] \arrow[dl] \\ \lambda^n & \end{tikzcd}$$

Estimator:

$$R_{s_t}^{\lambda} = (1-\lambda) \sum_{n=0}^{\infty} \lambda^n R_{s_t}^{(n+1)}$$

TD(λ)

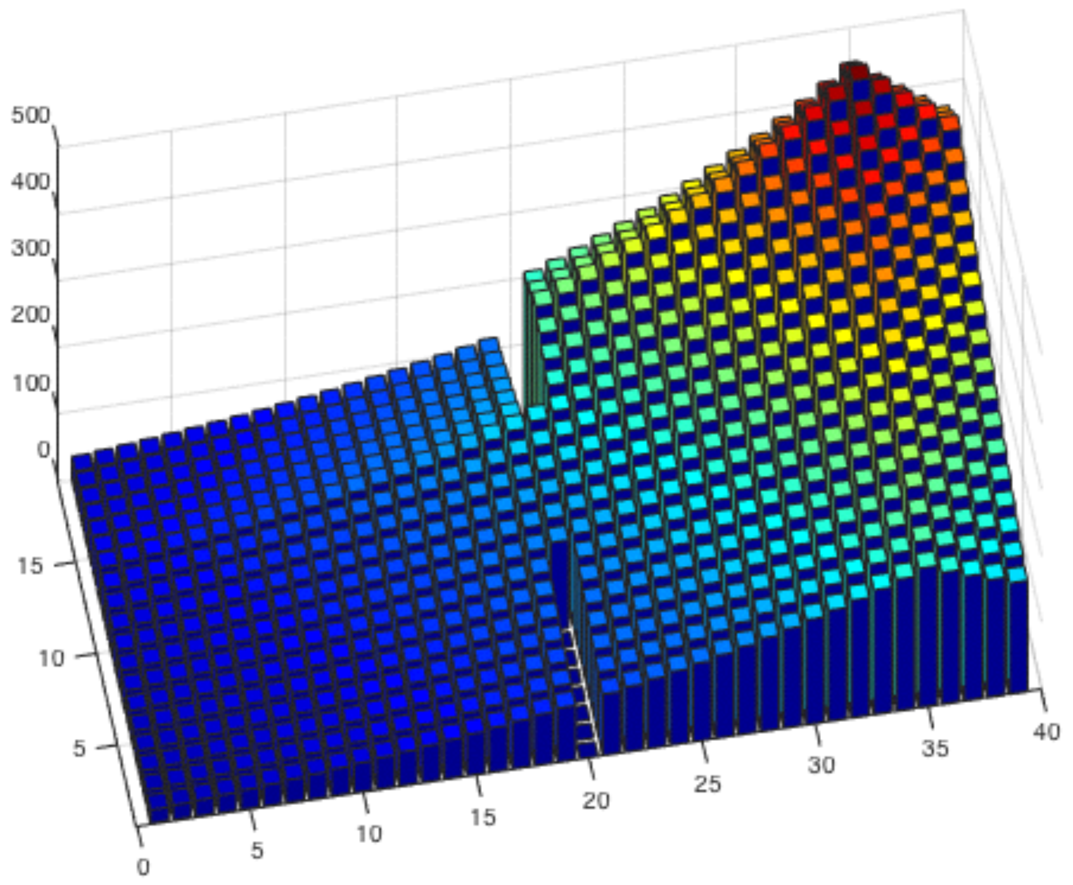
This is called the λ-return.

- At λ=0 we get TD, at λ=1 we get MC.
- Intermediate values of λ usually best.
- TD(λ) family of algorithms

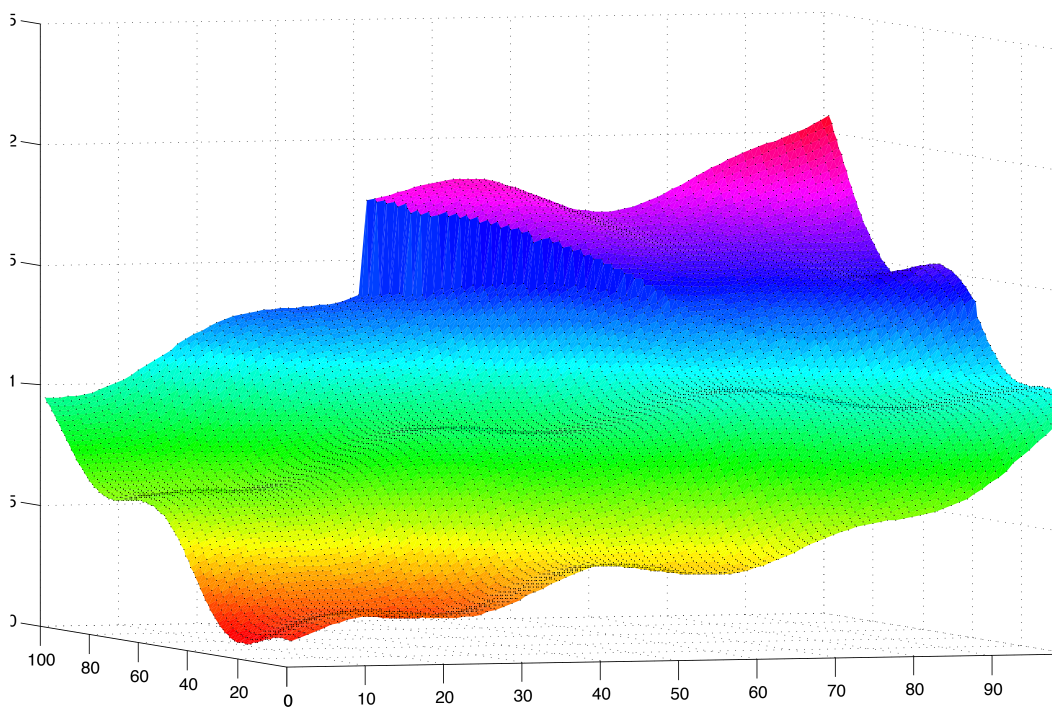
Real-Valued States

What if the states are real-valued?

- Cannot use table to represent Q.
- States may never repeat: must *generalize*.



vs



Function Approximation

How do we represent general function of state variables?

Many choices:

- Most popular is *linear value function approximation*.
- Use set of basis functions $\phi_1, \ldots, \phi_m$
- Define linear function of them:

$$\bar{V}(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

Learning task is to find vector of weights **w** to best approximate V.

Function Approximation

One choice of basis functions:

- Just use state variables directly: $[1, x, y]$

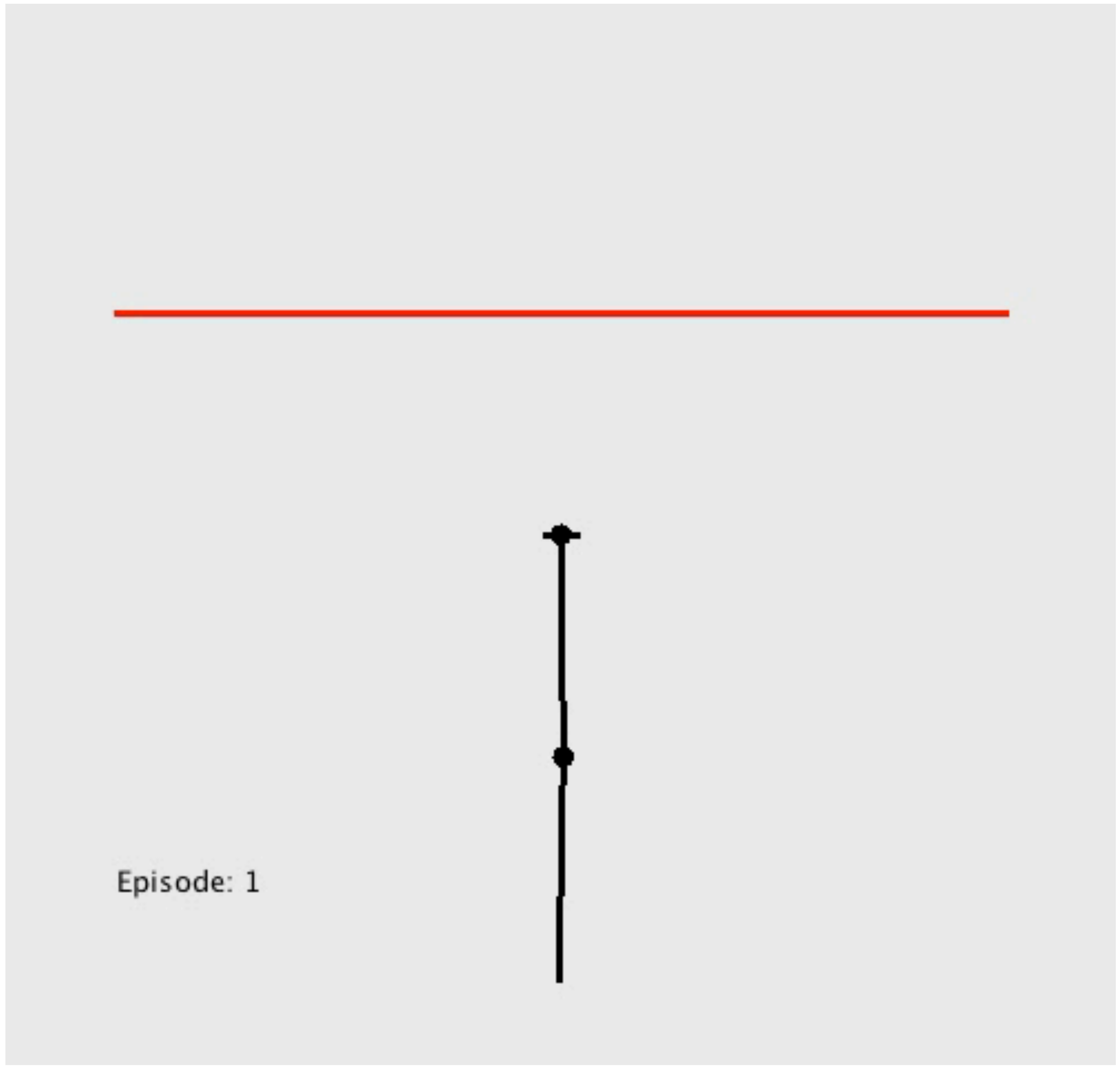
Another:

- Polynomials in state variables.
- E.g., $[1, x, y, xy, x^2, y^2, xy^2, x^2yx^2y^2]$
- This is like a Taylor expansion.

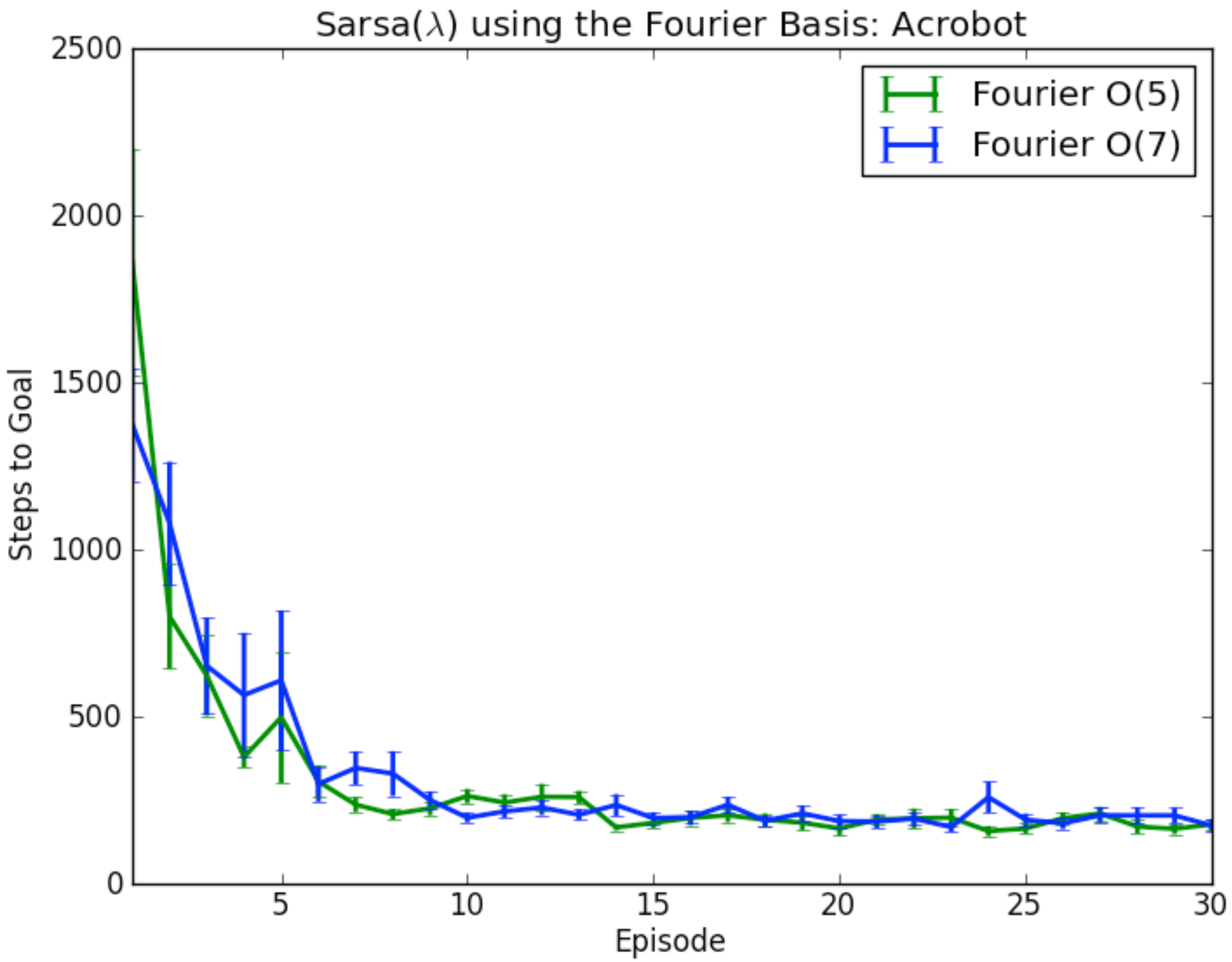
Another:

- Fourier terms on state variables.
- E.g., $[1, cos(\pi x), cos(\pi y), \cos(\pi[x+y])]$
- This is like a Fourier Series expansion.

Acrobot



Acrobot


Sarsa(λ) using the Fourier Basis: Acrobot
Fourier O(5)
Fourier O(7)
Steps to Goal
Episode
0
500
1000
1500
2000
2500
5
10
15
20
25
30

Function Approximation

TD-Gammon: Tesauro (circa 1992-1995)

- At or near best human level
- Learn to play Backgammon through self-play
- 1.5 million games
- Neural network function approximator
- TD(λ)

Changed the way the best human players played.

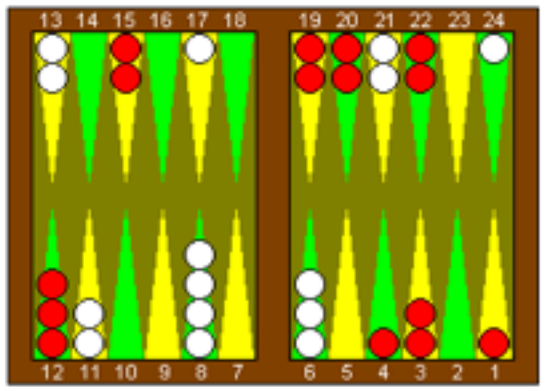


Figure 3. A complex situation where TD-Gammon's positional judgment is apparently superior to traditional expert thinking. White is to play 4-4. The obvious human play is 8-4*, 8-4, 11-7, 11-7. (The asterisk denotes that an opponent checker has been hit.) However, TD-Gammon's choice is the surprising 8-4*, 8-4, 21-17, 21-17! TD-Gammon's analysis of the two plays is given in Table 3.

Policy Search

So far: improve policy via value function.

Sometimes policies are simpler than value functions:

- Parametrized program $\pi(s, a|\theta)$

Sometimes we wish to search in space of restricted policies.

In such cases it makes sense to search directly in *policy-space* rather than trying to learn a value function.

Policy Search

Can apply *any* generic optimization method for θ.

One particular approach: policy gradient.

- Compute and ascend $\partial R / \partial \theta$
- This is the gradient of return w.r.t policy parameters

Policy gradient theorem:

$$\frac{\partial R}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta} (Q^\pi(s,a) - b(s))$$

Therefore, one way is to learn Q and then ascend gradient.
Q need only be defined using basis functions computed from θ.

Aibo Gait Optimization

from Kohl and Stone, ICRA 2004.

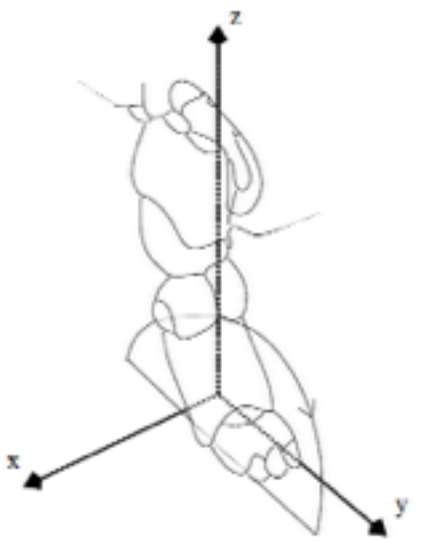
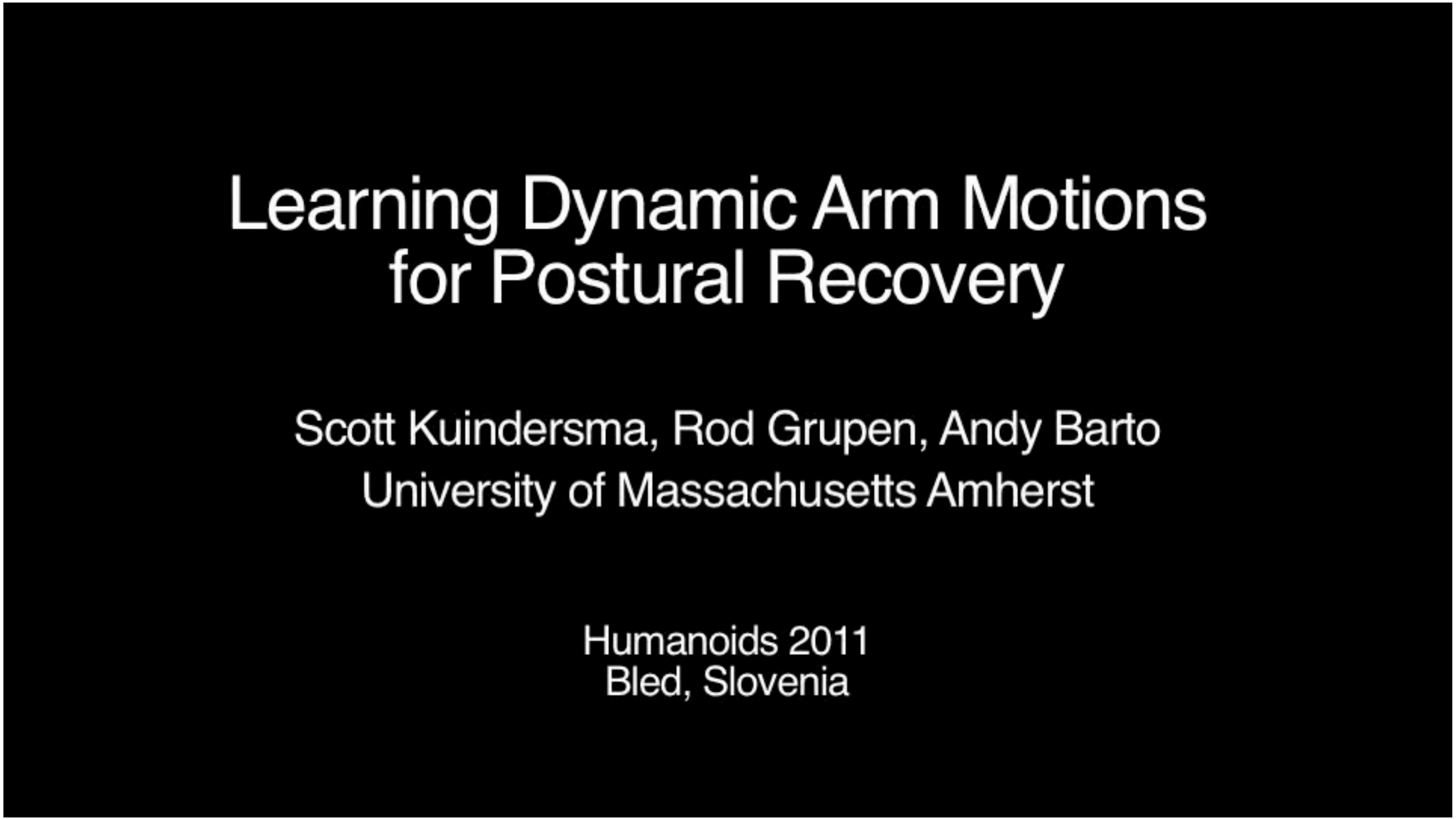


Fig. 2. The elliptical locus of the Aibo's foot. The half-ellipse is defined by length, height, and position in the x-y plane.

All told, the following set of 12 parameters define the Aibo's gait [10]:

- The front locus (3 parameters: height, x-pos., y-pos.)
- The rear locus (3 parameters)
- Locus length
- Locus skew multiplier in the x-y plane (for turning)
- The height of the front of the body
- The height of the rear of the body
- The time each foot takes to move through its locus
- The fraction of time each foot spends on the ground

Postural Recovery



Learning Dynamic Arm Motions for Postural Recovery

Scott Kuindersma, Rod Grupen, Andy Barto
University of Massachusetts Amherst

Humanoids 2011
Bled, Slovenia

Reinforcement Learning

Machine Learning for control.

Very active area of current research, applications in:

- Robotics
- Operations Research
- Computer Games
- Theoretical Neuroscience

AI

- The primary function of the brain is control.