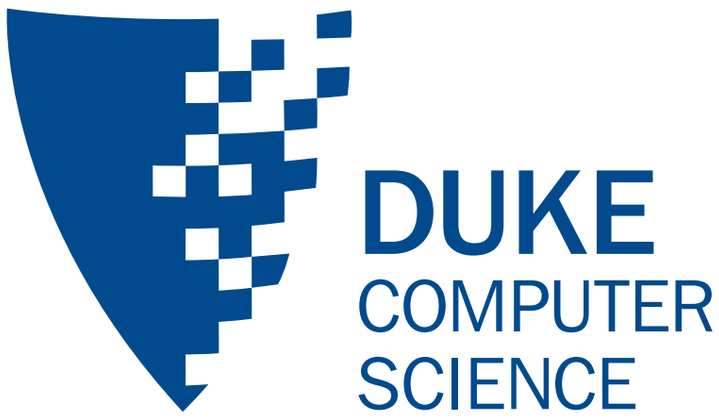
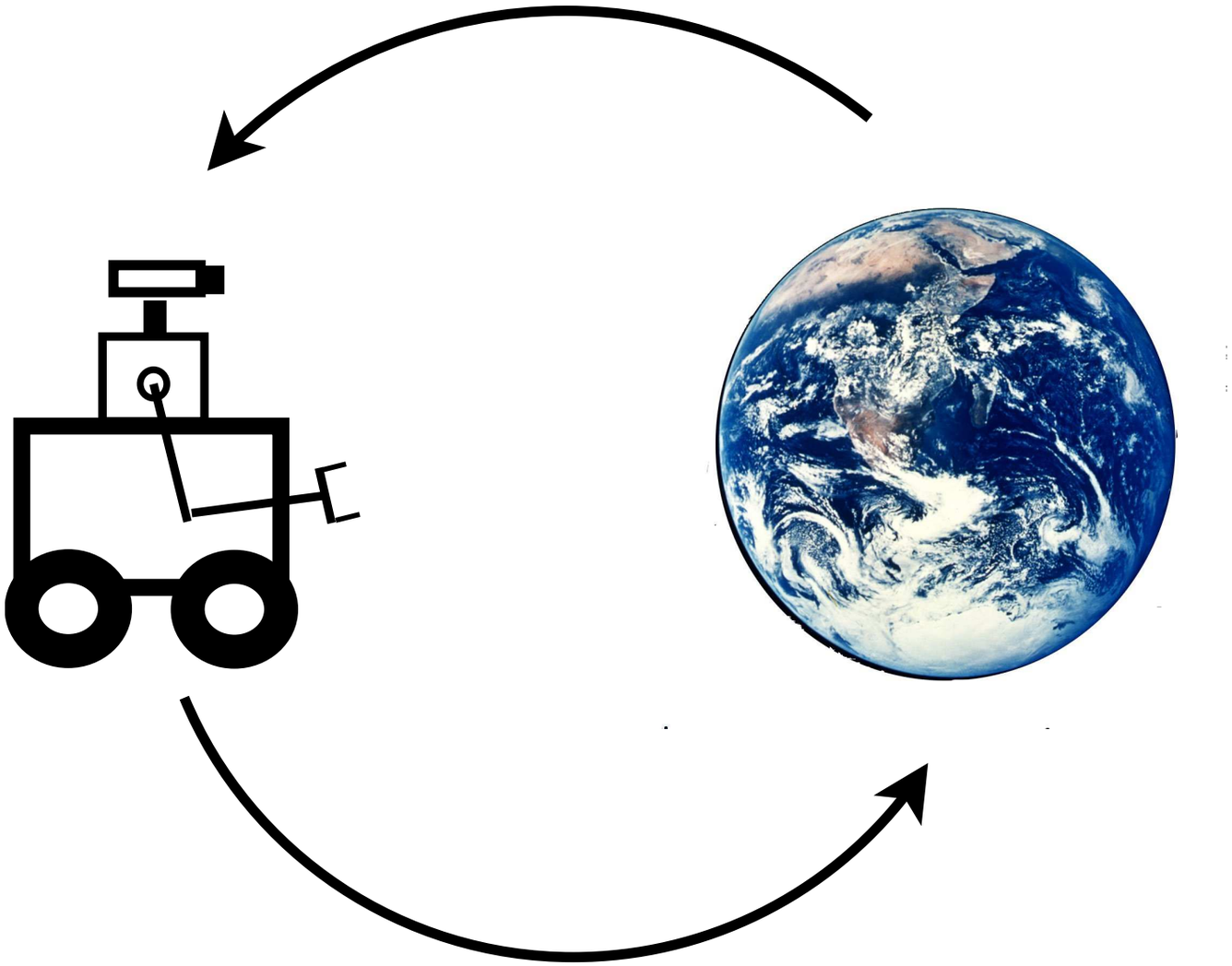
# Decision Making for Robots and Autonomous Systems

Fall 2015




George Konidaris
gdk@cs.duke.edu

# The World Reacts



# Markov Decision Processes

$S$ : set of *states*
$A$ : set of actions
$\gamma$ : 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 $s$.

(some states are absorbing - execution stops)

# MDPs

Our goal is to find a policy:

$$\pi : S \rightarrow A$$

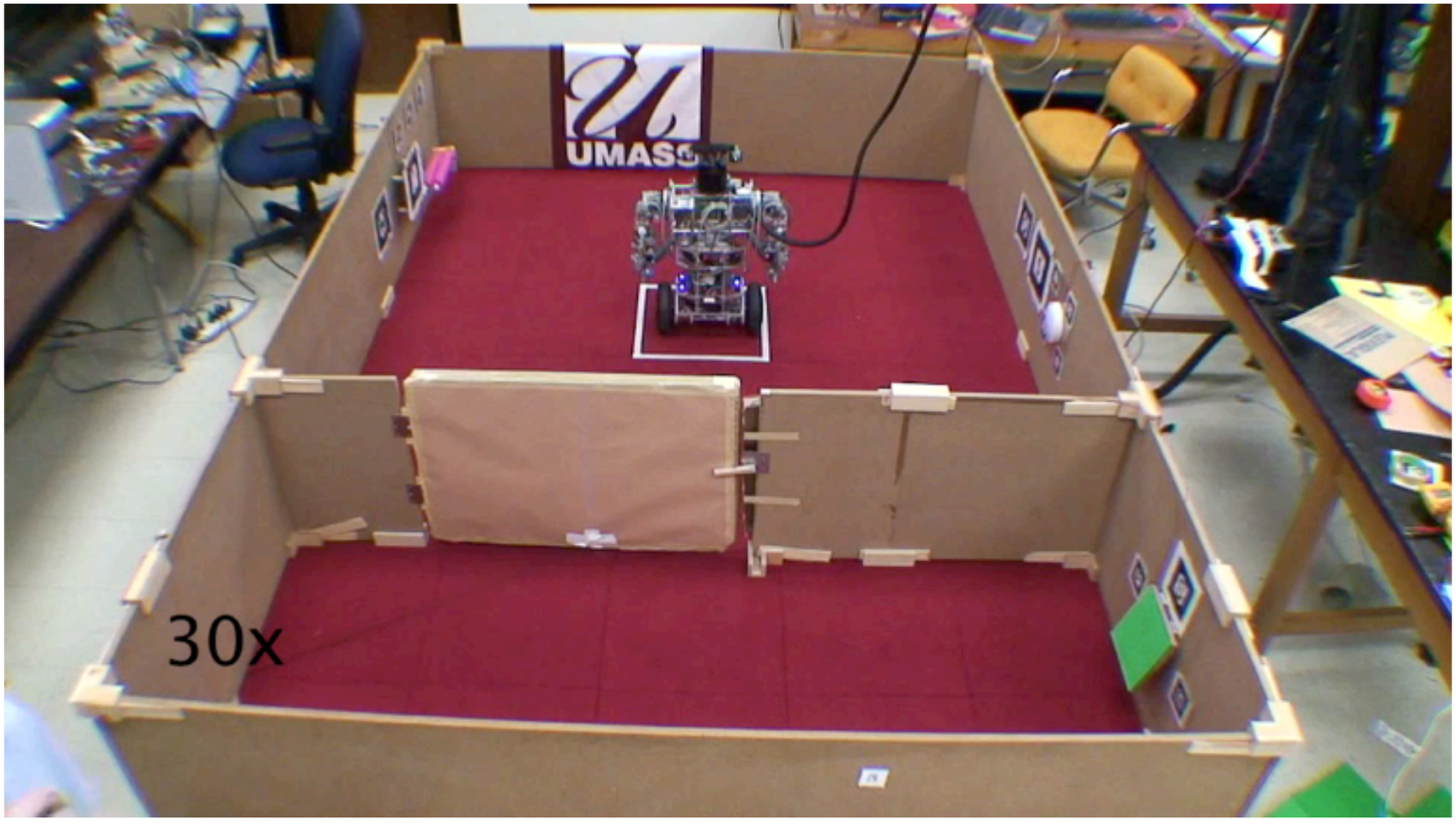
… that **maximizes *return*: expected sum of rewards.**
(equiv: min sum of costs)

$$\sum_{i=1}^{\infty} \mathbb{E}[\gamma^i r_i]$$

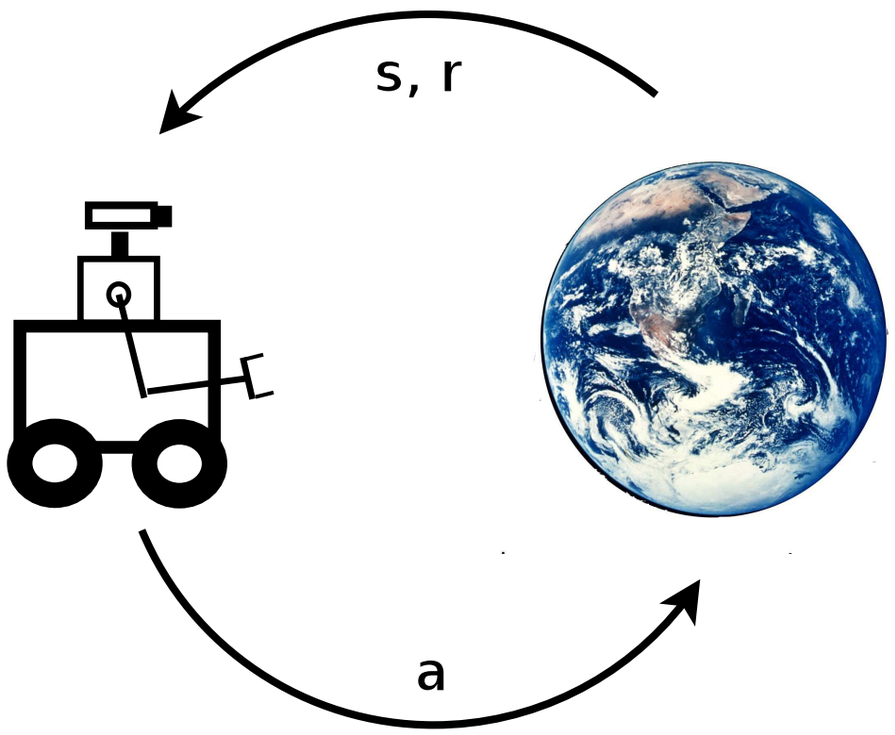
# Reinforcement Learning

DUKE COMPUTER SCIENCE

***What if we don't know T or R (or both?)***



# 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 $r_t$

**Cannot *query* a transition - must actually *do it*, in order to get transition data.**

# How do we do this!

Same as before - learn *V*!

But! Before, we could do this during policy iteration:

$$\pi(s) = \max_a \left[ R(s,a) + \gamma \sum_{s'} T(s'|s,a) V(s') \right]$$

… now we cannot. So instead of V, we learn Q:

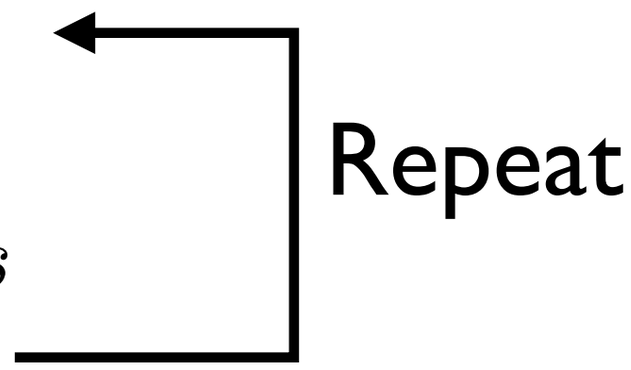
$$Q_\pi(s,a) = \mathbb{E}\left[ \sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, s_0 = s, a_0 = a \right]$$

This is the value of executing $a$ in state $s$, ***then following*** $\pi$.
Note that: $Q^\pi(s, \pi(s)) = V^\pi(s)$

# Policy Iteration

General policy improvement framework:

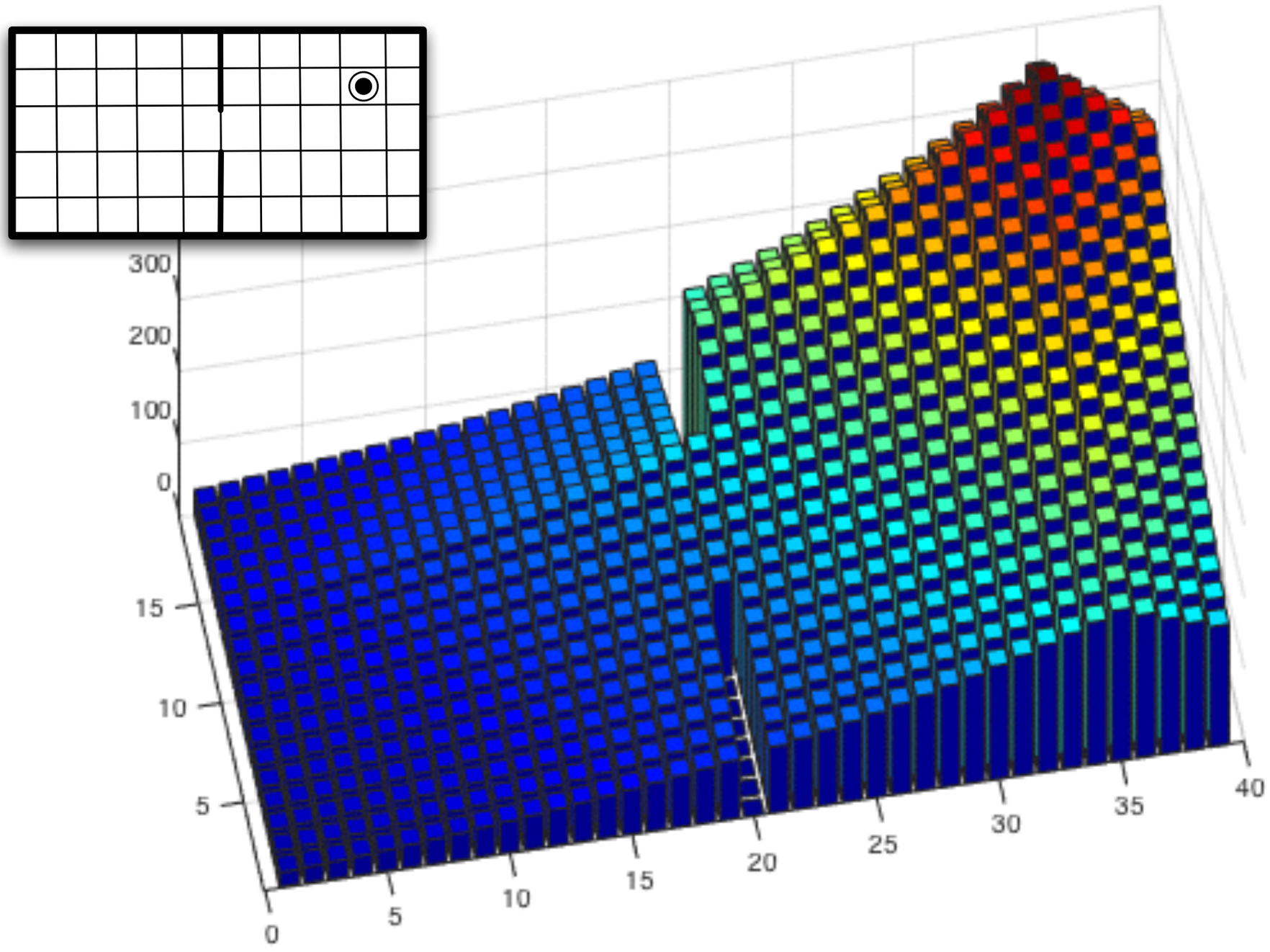
1. Start with a policy $\pi$
2. Learn $Q_\pi$
3. Improve $\pi$
   a. $\pi(s) = \max_a Q(s, a), \forall s$




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 Functions



# Value Function Learning

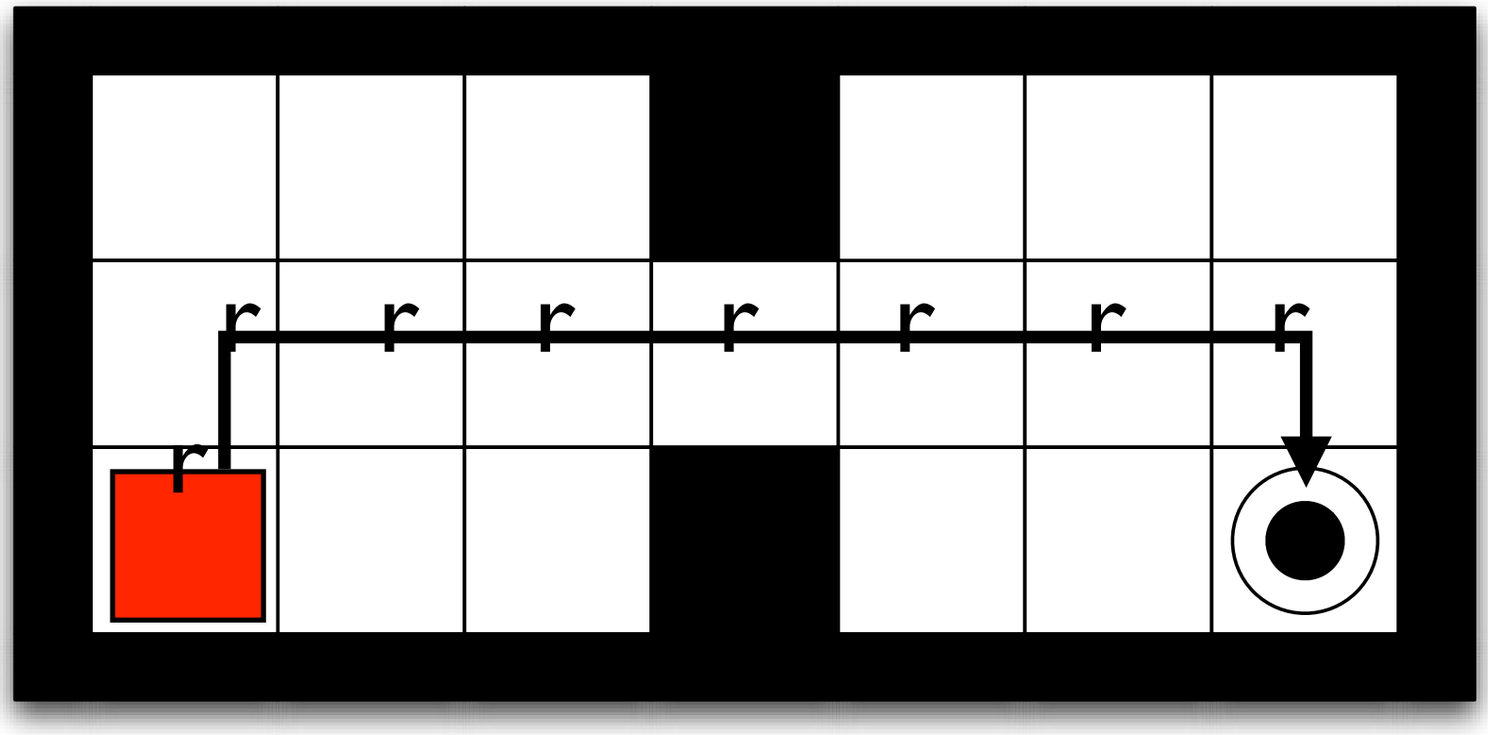
Learning proceeds by gathering samples of $Q(s, a)$.

Methods differ by:

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

# Monte Carlo

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

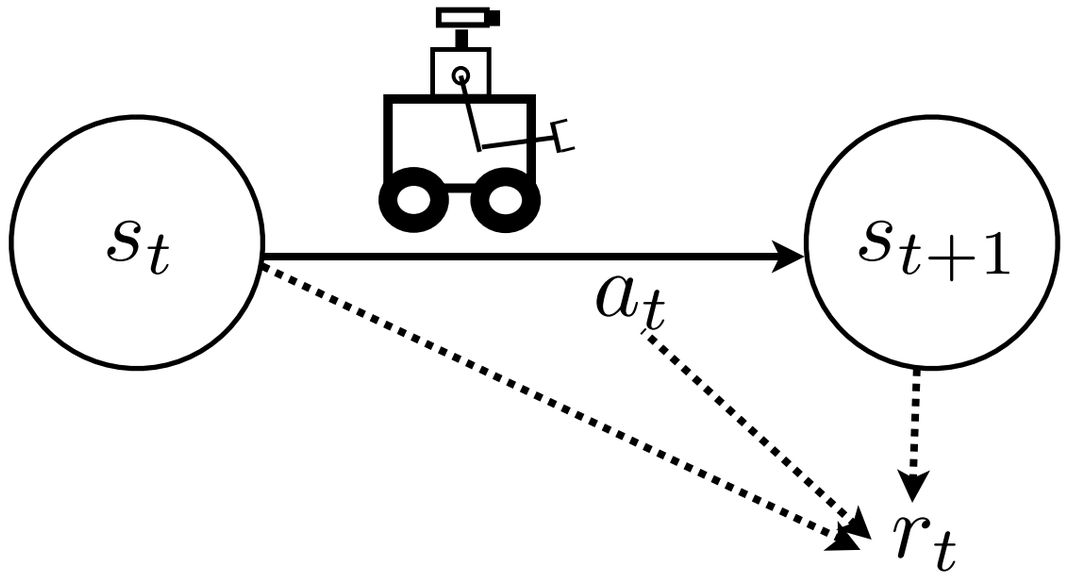



Do this repeatedly, average values:

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

# TD Learning

Exploit the Bellman equation: ***temporal difference error.***




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

# Sarsa

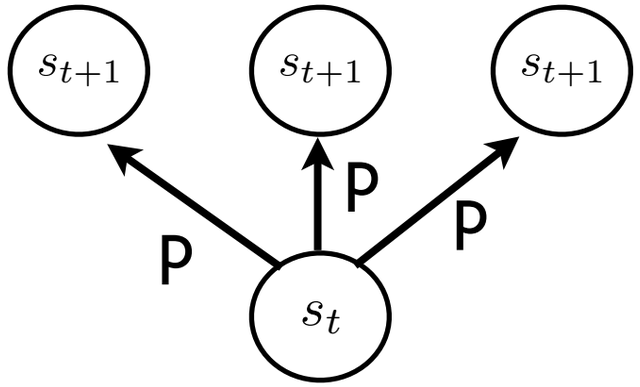
Sarsa: very simple algorithm

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*.

Compare with the full expectation (given *T*):

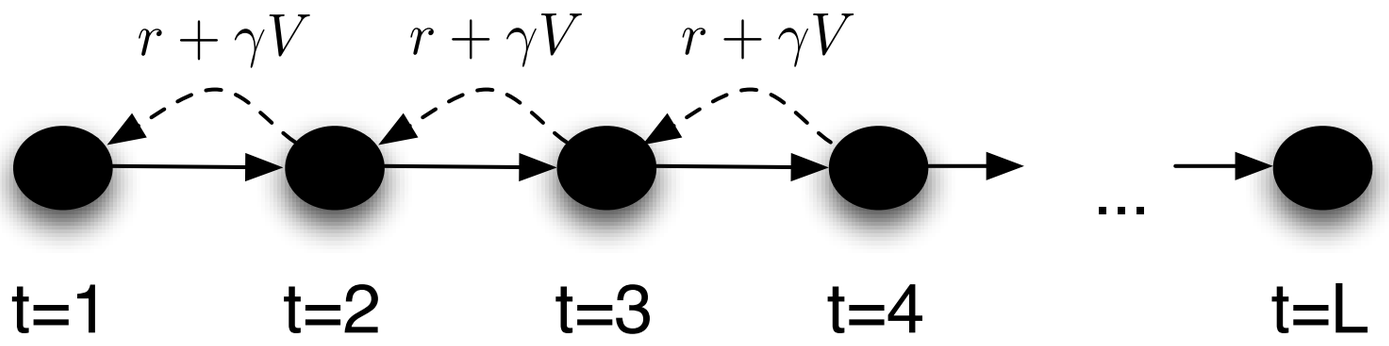


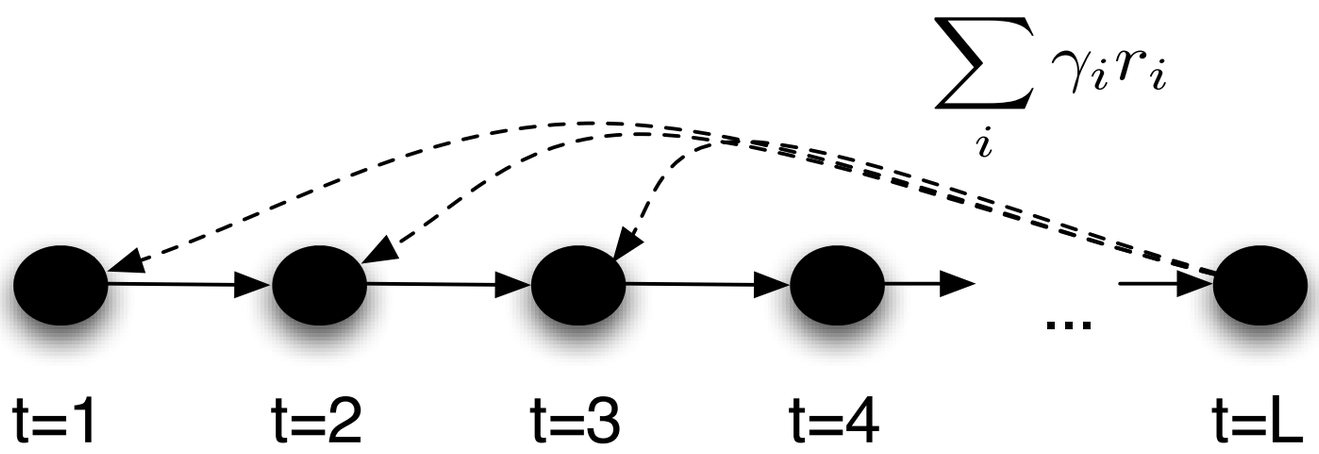

$$\delta = \mathbb{E}_{\pi,T}\left[r + \gamma Q(s', a')\right] - Q(s, a)$$

*(full backup)*

# 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:

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

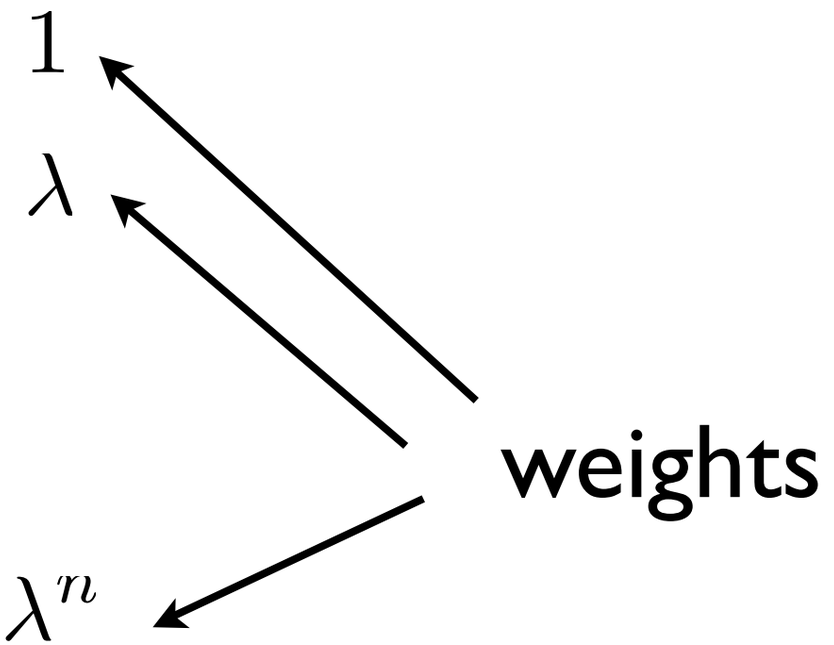
Each tells us something about the value function.

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

# TD(λ)

Weighted sum:

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




Estimator:

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

# TD(λ)

DUKE COMPUTER SCIENCE

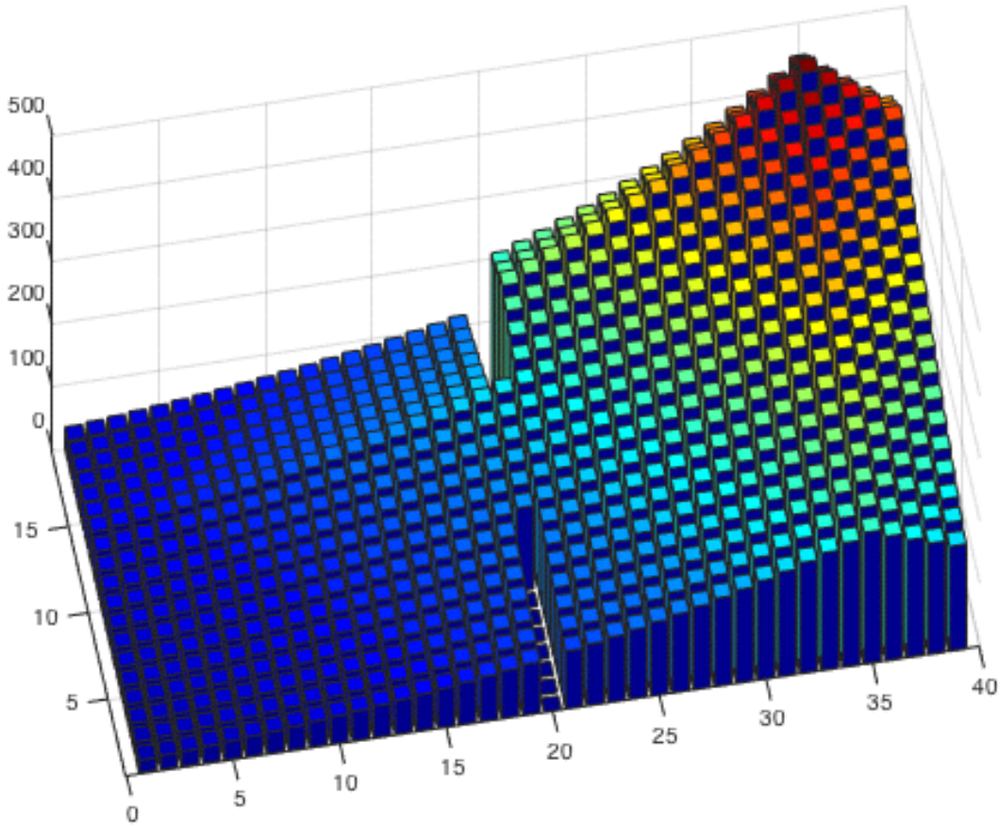
This is called the λ-return.

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

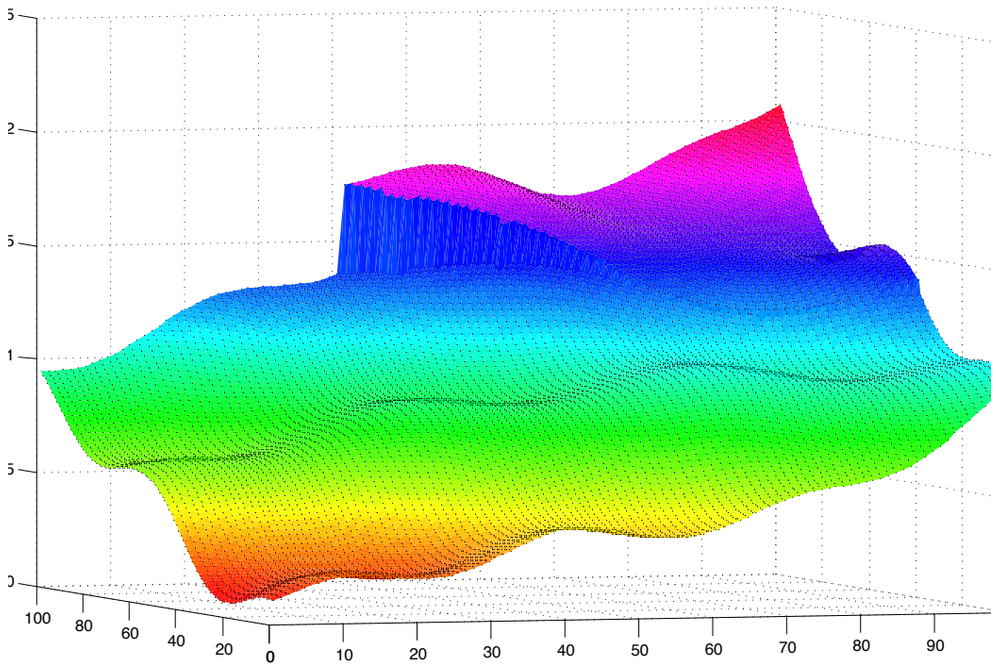
# Real-Valued States

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

DUKE COMPUTER SCIENCE

One choice of basis functions:

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

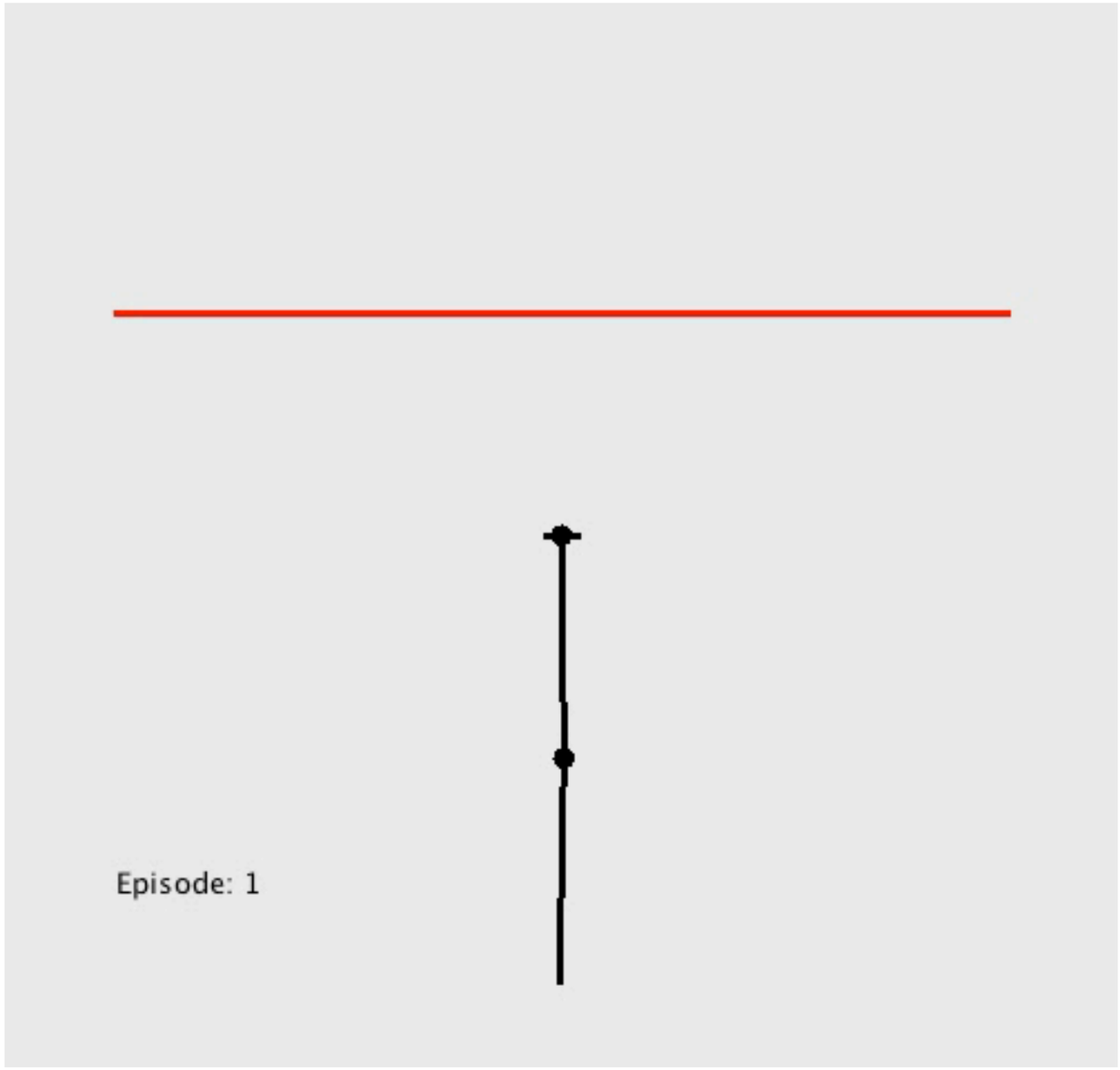
Another:

- 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.

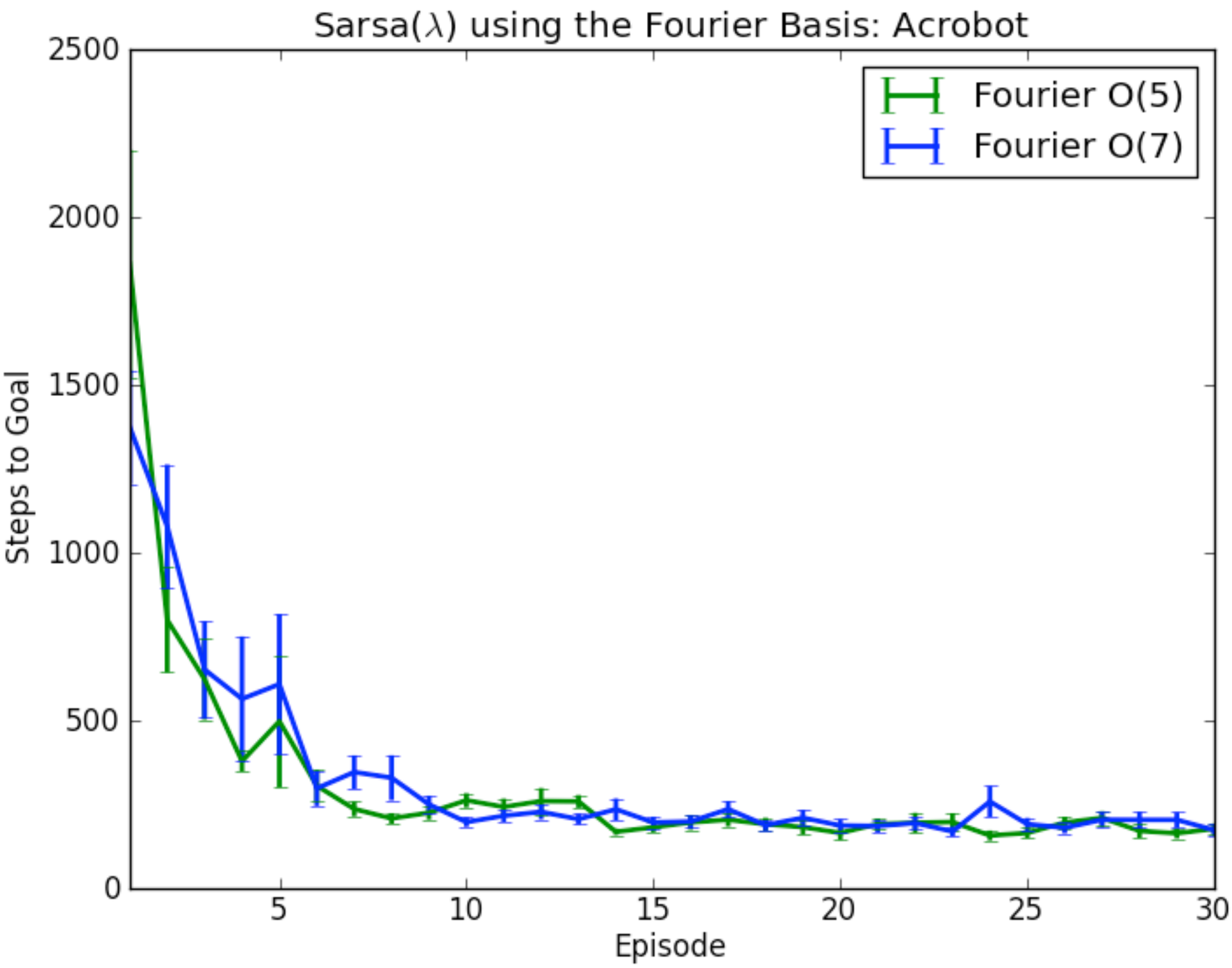
Another:

- 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
2500
2000
1500
1000
500
0
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.

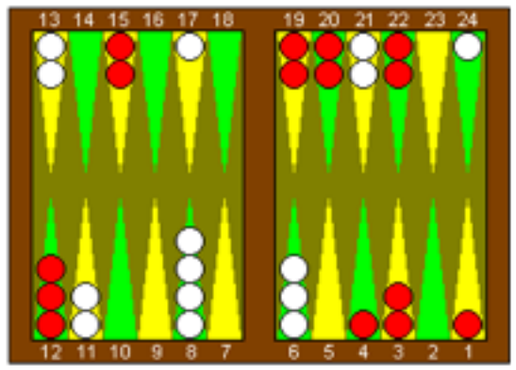



**Figure 3.** A complex situation where TD-Gammon's positional judgment is apparently superior to traditional expert thinking. White is to play 4-4. The obvious human play is 8-4*, 8-4, 11-7, 11-7. (The asterisk denotes that an opponent checker has been hit.) However, TD-Gammon's choice is the surprising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