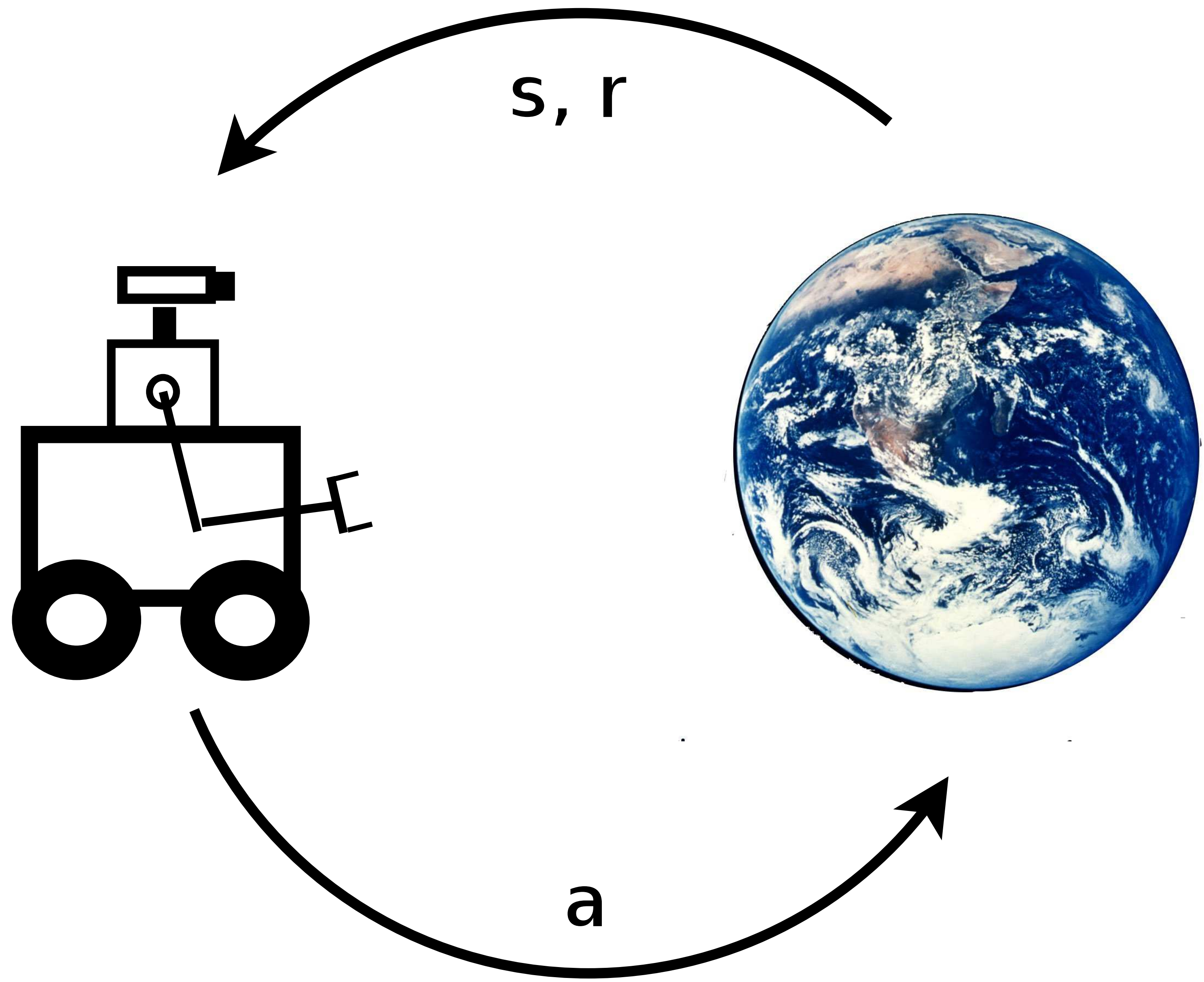
Reinforcement Learning II

George Konidaris
gdk@cs.brown.edu

Fall 2021

Reinforcement Learning



$$\pi : S \rightarrow A$$

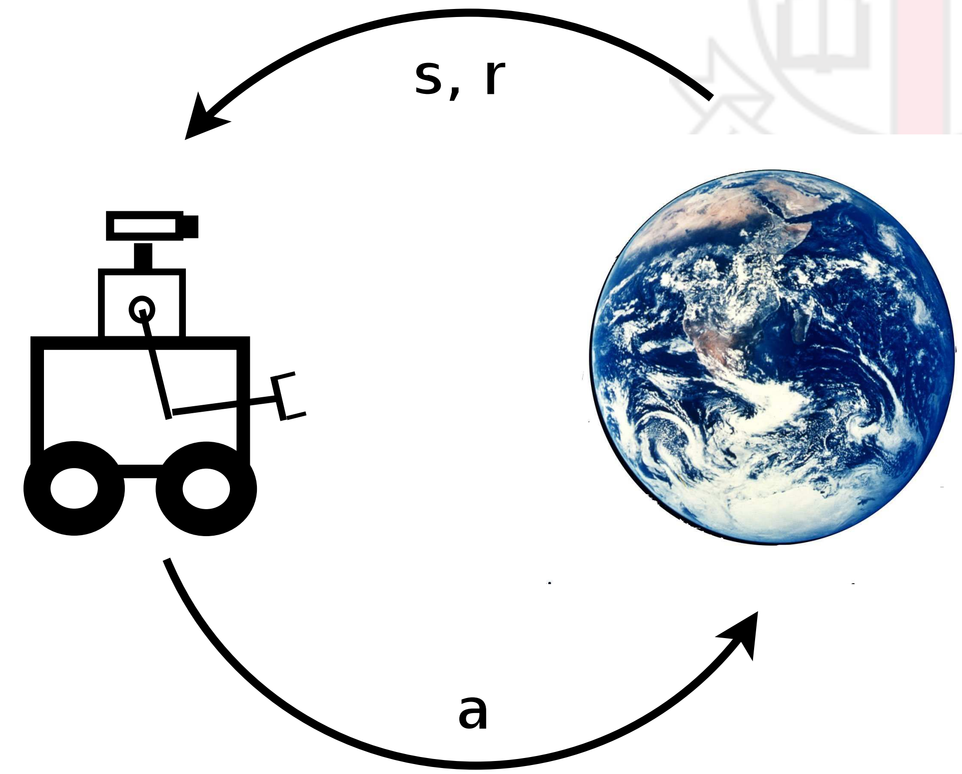
$$\max_{\pi} R = \sum_{t=0}^{\infty} \gamma^t r_t$$

MDPs

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):

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

Markov Decision Processes

S : set of *states*
A : set of actions
γ : discount factor

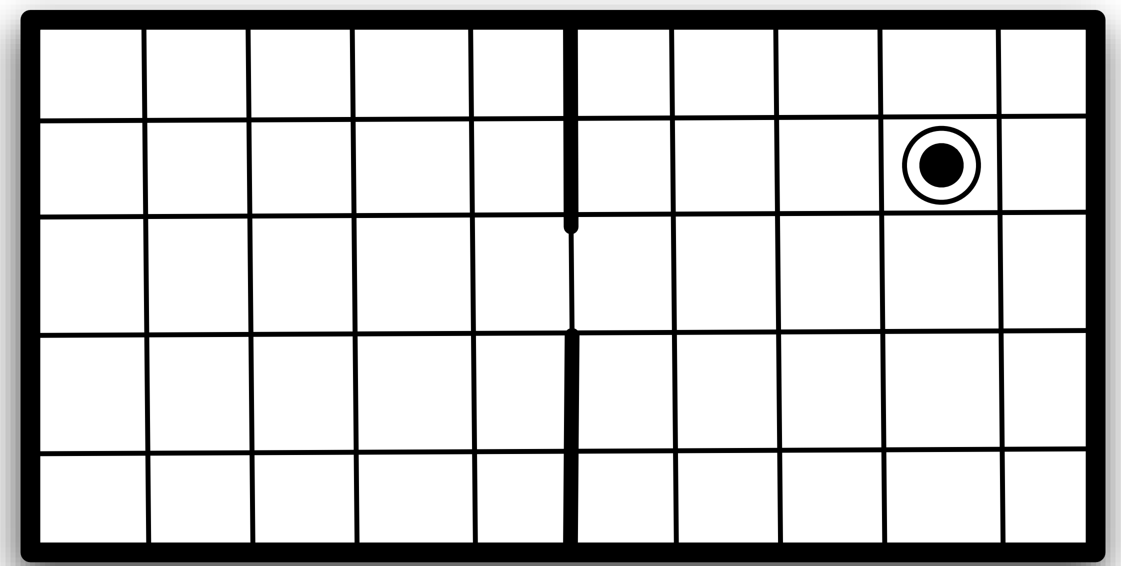
$$< S, A, \gamma, R, T >$$

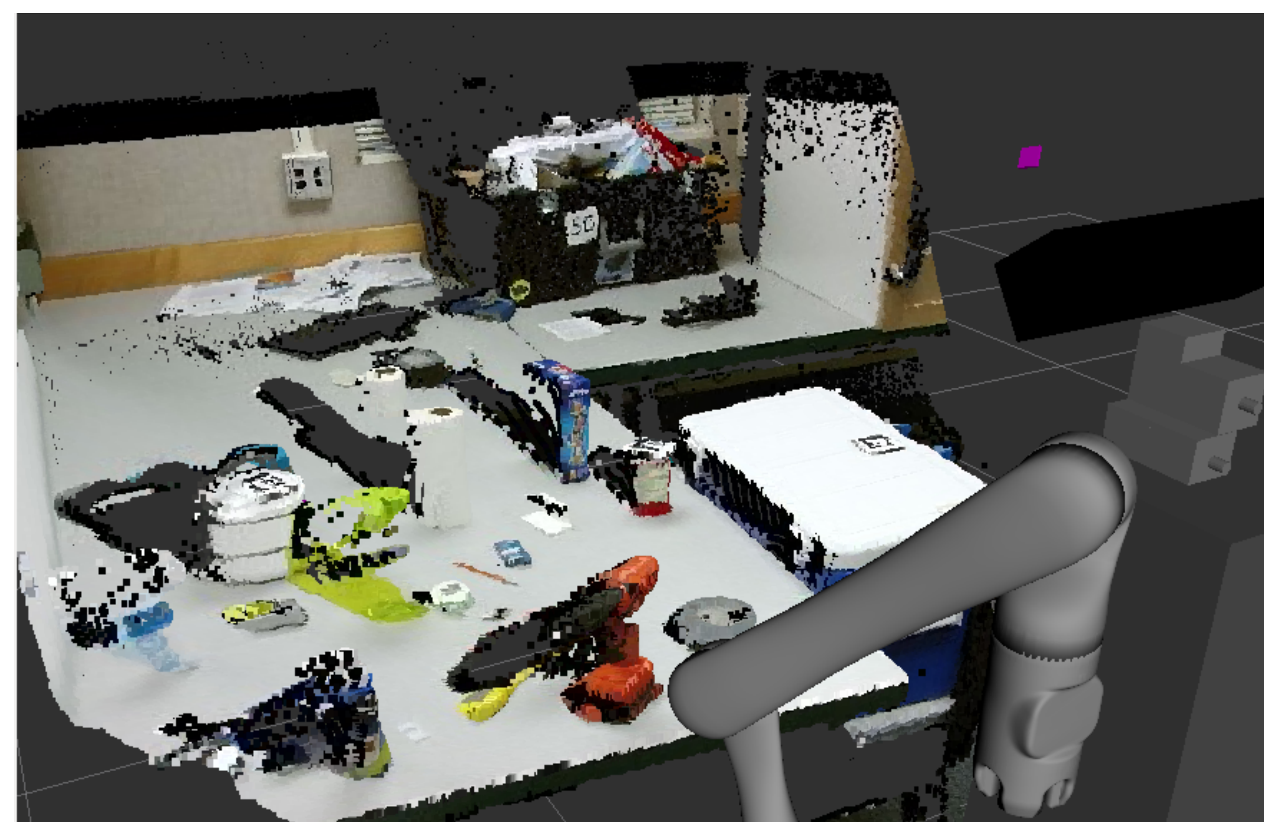
R : reward function
$R(s, a, s')$ is the reward received taking action a from state s and transitioning to state s'.

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

RL: *one or both of T, R unknown.*

The World

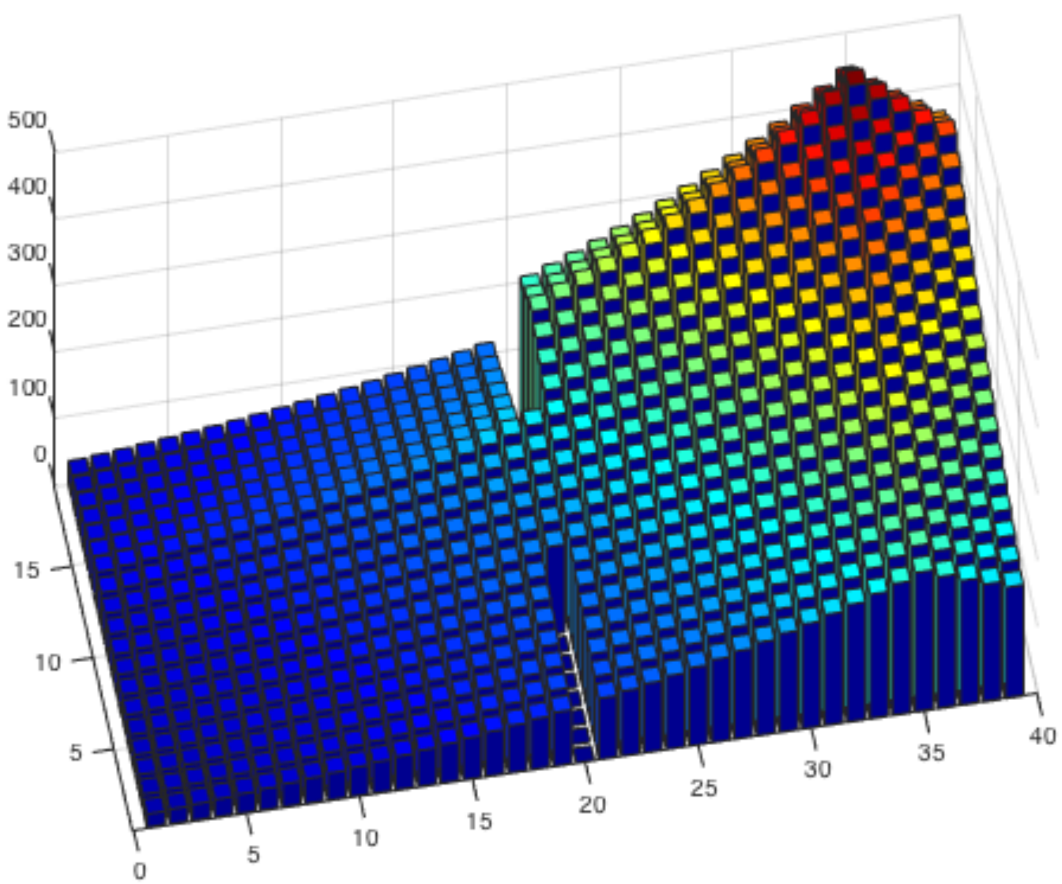




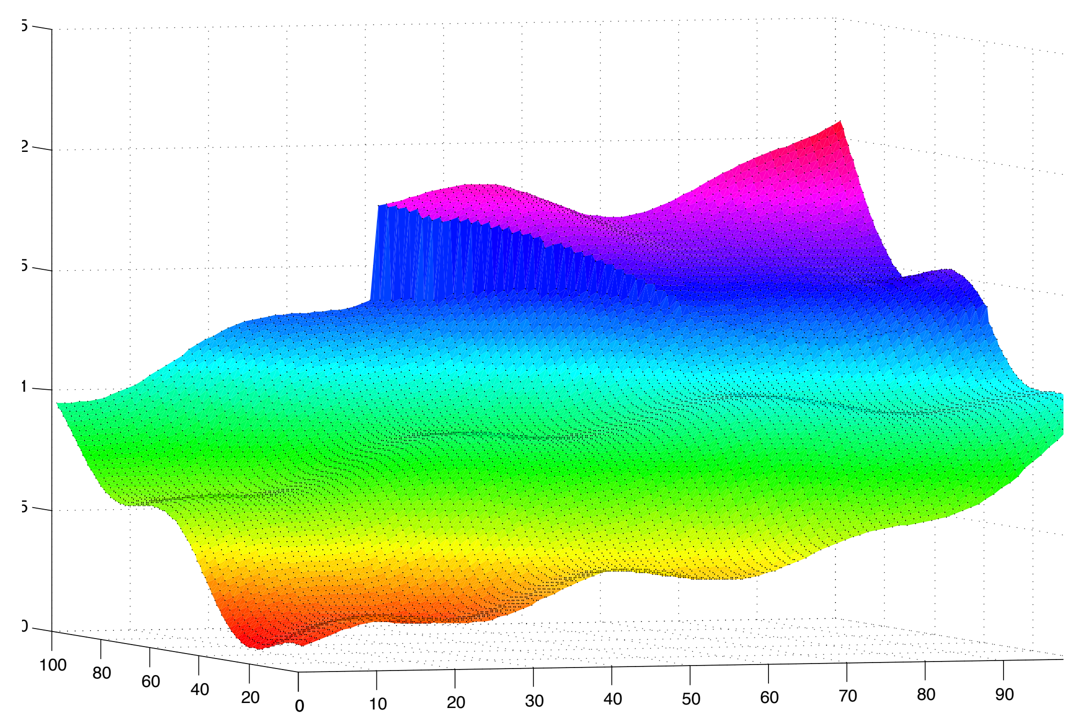
Real-Valued States

What if the states are real-valued?

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

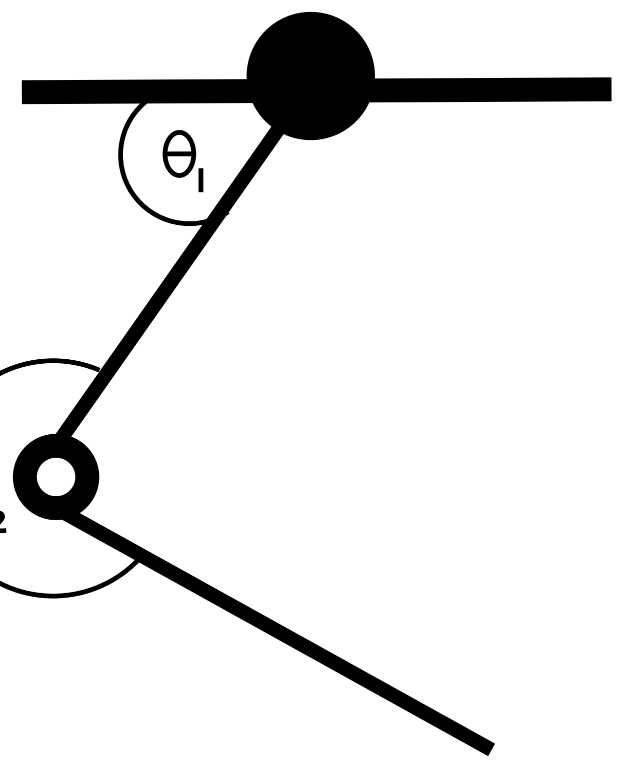


vs



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

Value Function Approximation

Represent Q function:

$$Q(s, a, w) : \mathbb{R}^n \rightarrow \mathbb{R}$$

parameter vector

Samples of form:

$$(s_i, a_i, r_i, s_{i+1}, a_{i+1})$$

Minimize summed squared TD error:

$$\min_w \sum_{i=0}^{n} \left(r_i + \gamma Q(s_{i+1}, a_{i+1}, w) - Q(s_i, a_i, w)\right)^2$$

Value Function Approximation

Given a function approximator, compute the gradient and descend it.

Which function approximator to use?

Simplest thing you can do:

- *Linear value function approximation*.
- Use set of basis functions $\phi_1, ..., \phi_n$
- Q is a linear function of them:

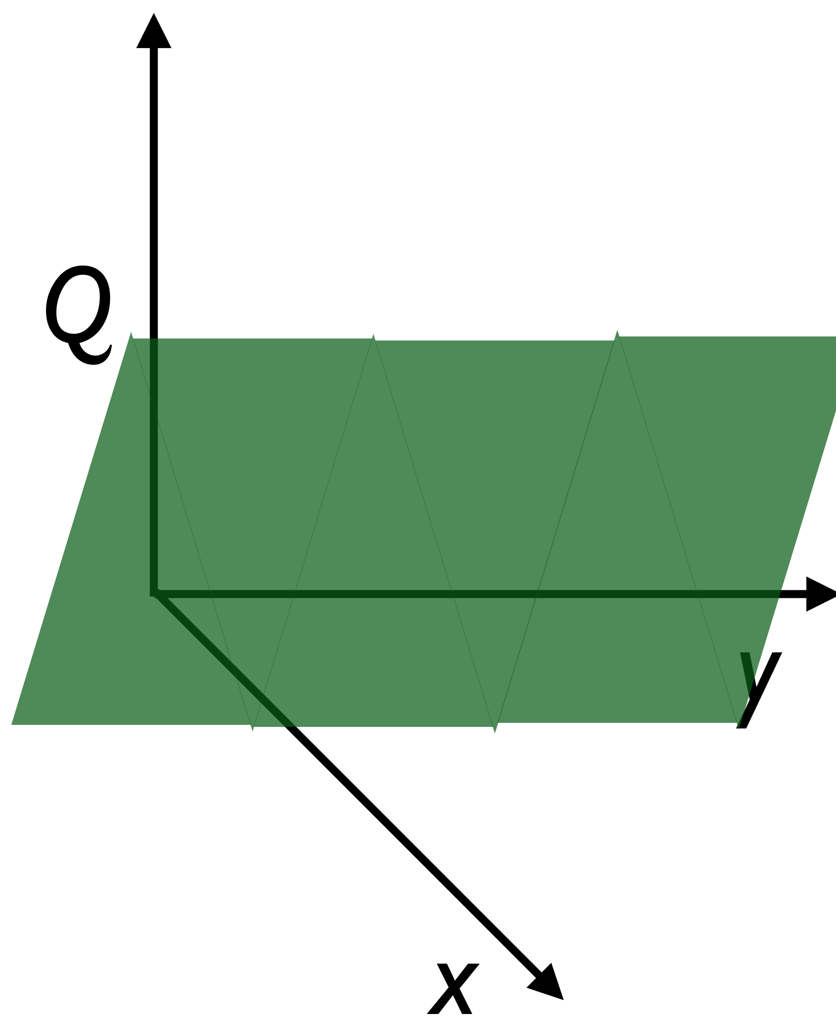
$$\hat{Q}(s,a) = w \cdot \Phi(s,a) = \sum_{j=1}^{n} w_j \phi_j(s,a)$$

Function Approximation

One choice of basis functions:

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

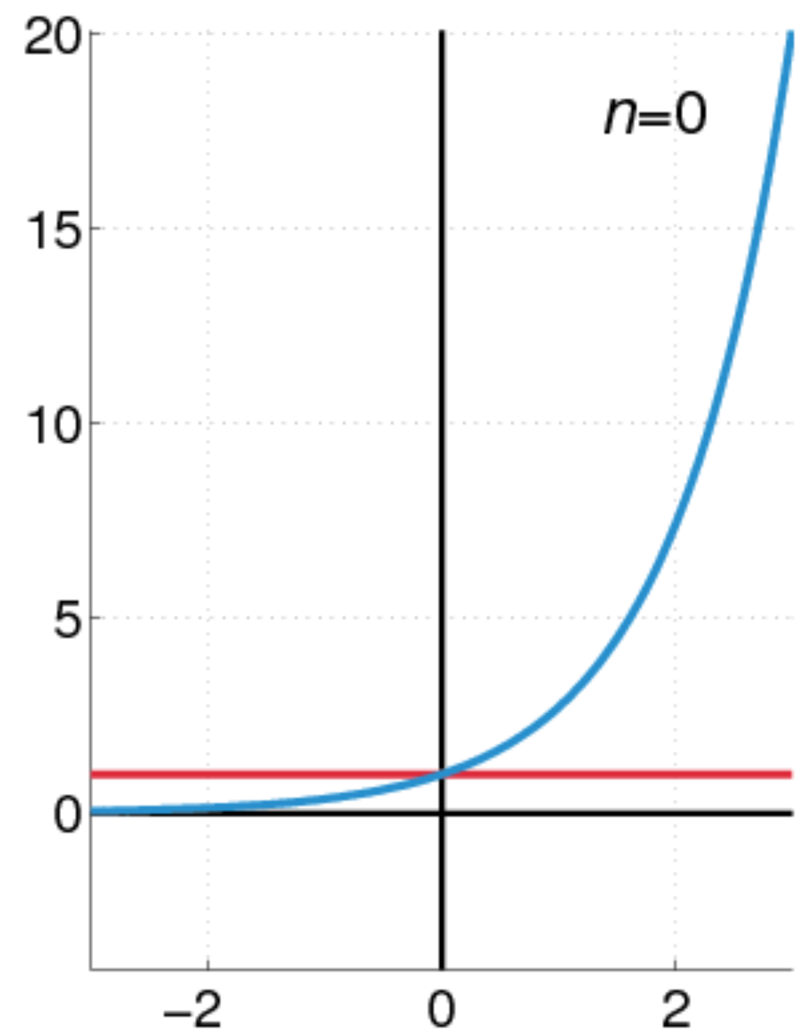
What can be represented this way?



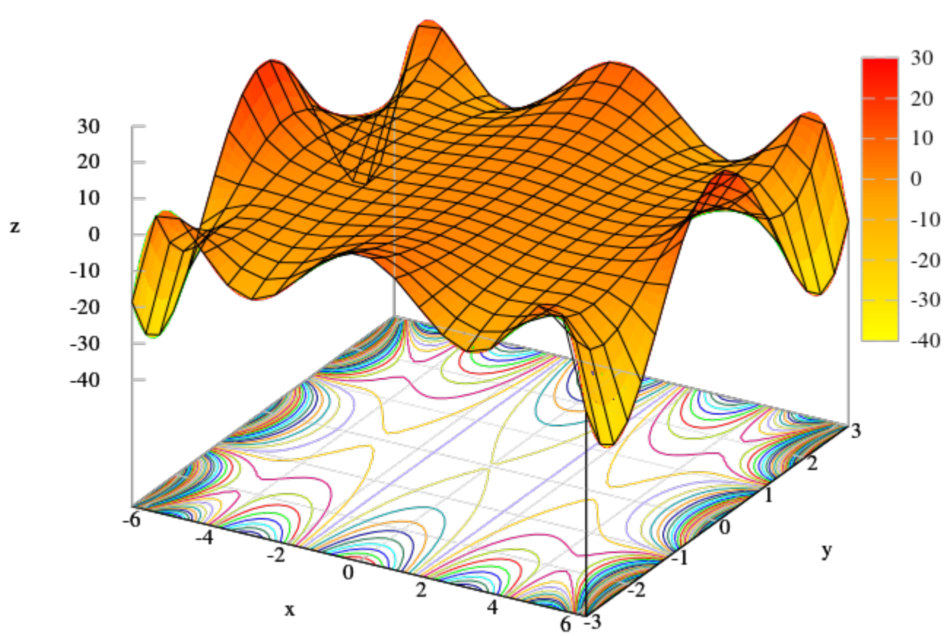
Polynomial Basis

More powerful:

- Polynomials in state variables.
  - 1st order: $[1, x, y, xy]$
  - 2nd order: $[1, x, y, xy, x^2, y^2, x^2y, y^2x, x^2y^2]$
- This is like a Taylor expansion.



What can be represented?



Function Approximation

How to get the terms of the Taylor series?

Each term has an exponent:

$c_i \in [0, ..., n]$

$$\phi_c(x, y, z) = x^{c_1} y^{c_2} z^{c_3}$$

all combinations generates basis

$$\phi_c(x, y, z) = x = x^1 y^0 z^0 \qquad c = [1, 0, 0]$$

$$\phi_c(x, y, z) = xy^2 = x^1 y^2 z^0 \qquad c = [1, 2, 0]$$

$$\phi_c(x, y, z) = x^2 z^4 = x^2 y^0 z^4 \qquad c = [2, 0, 4]$$

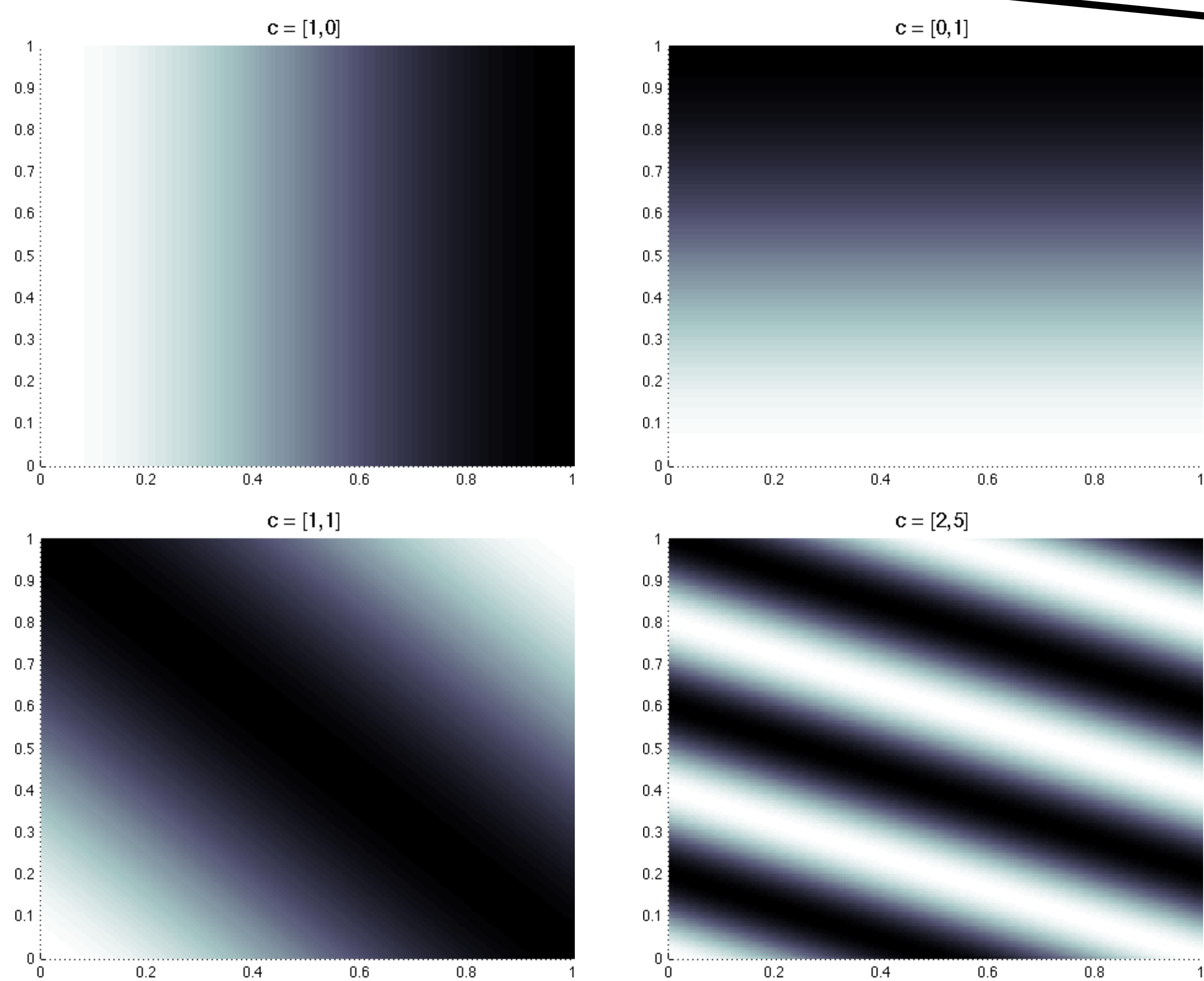
$$\phi_c(x, y, z) = y^3 z^1 = x^0 y^3 z^1 \qquad c = [0, 3, 1]$$

Function Approximation

Another:

- Fourier terms on state variables.
  - $[1, cos(\pi x), cos(\pi y), \cos(\pi[x+y])]$
  - $cos(\pi c \cdot [x, y, z])$

coefficient vector



Objective Function Minimization

First, let's do *stochastic gradient descent.*

As each data point (transition) comes in

- compute gradient of objective w.r.t. data point
- descend gradient a little bit

$$\hat{Q}(s,a) = w \cdot \Phi(s,a)$$

$$\min_w \sum_{i=0}^{n} \left(r_i + \gamma w \cdot \phi(s_{i+1}, a_{i+1}) - w \cdot \phi(s_i, a_i)\right)^2$$

Gradient

For each weight w_j:

$$\frac{\partial}{\partial w_j} \sum_{i=0}^{n} \left(r_i + \gamma w \cdot \phi(s_{i+1}, a_{i+1}) - w \cdot \phi(s_i, a_i)\right)^2$$

$$= -2 \sum_{i=0}^{n} \left(r_i + \gamma w \cdot \phi(s_{i+1}, a_{i+1}) - w \cdot \phi(s_i, a_i)\right) \phi_j(s_i, a_i)$$

so for time *i* the contribution for weight w_j is:

TD error

$$\left(r_i + \gamma w \cdot \phi(s_{i+1}, a_{i+1}) - w \cdot \phi(s_i, a_i)\right) \phi_j(s_i, a_i)$$

make a step:

$$w_{j,i+1} = w_{j,i} + \alpha \left(r_i + \gamma w \cdot \phi(s_{i+1}, a_{i+1}) - w \cdot \phi(s_i, a_i)\right) \phi_j(s_i, a_i)$$

$$w_{i+1} = w_i + \alpha \delta \phi(s_i, a_i)$$ vector

λ-Gradient

The same logic applies when using eligibility traces.

$$w_{i+1} = w_i + \alpha\delta\phi(s_i, a_i)$$

becomes

$$w_{i+1} = w_i + \alpha\delta e$$

where

vectors

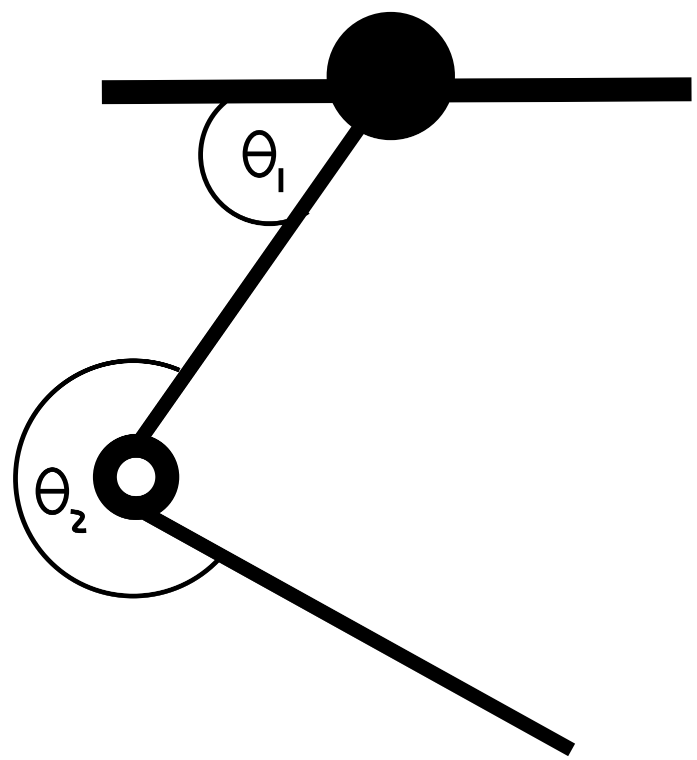
$$e_t = \gamma\lambda e_{t-1} + \phi(s_t, a_t)$$

$$e_0 = \bar{0}$$

[Sutton and Barto, 1998]

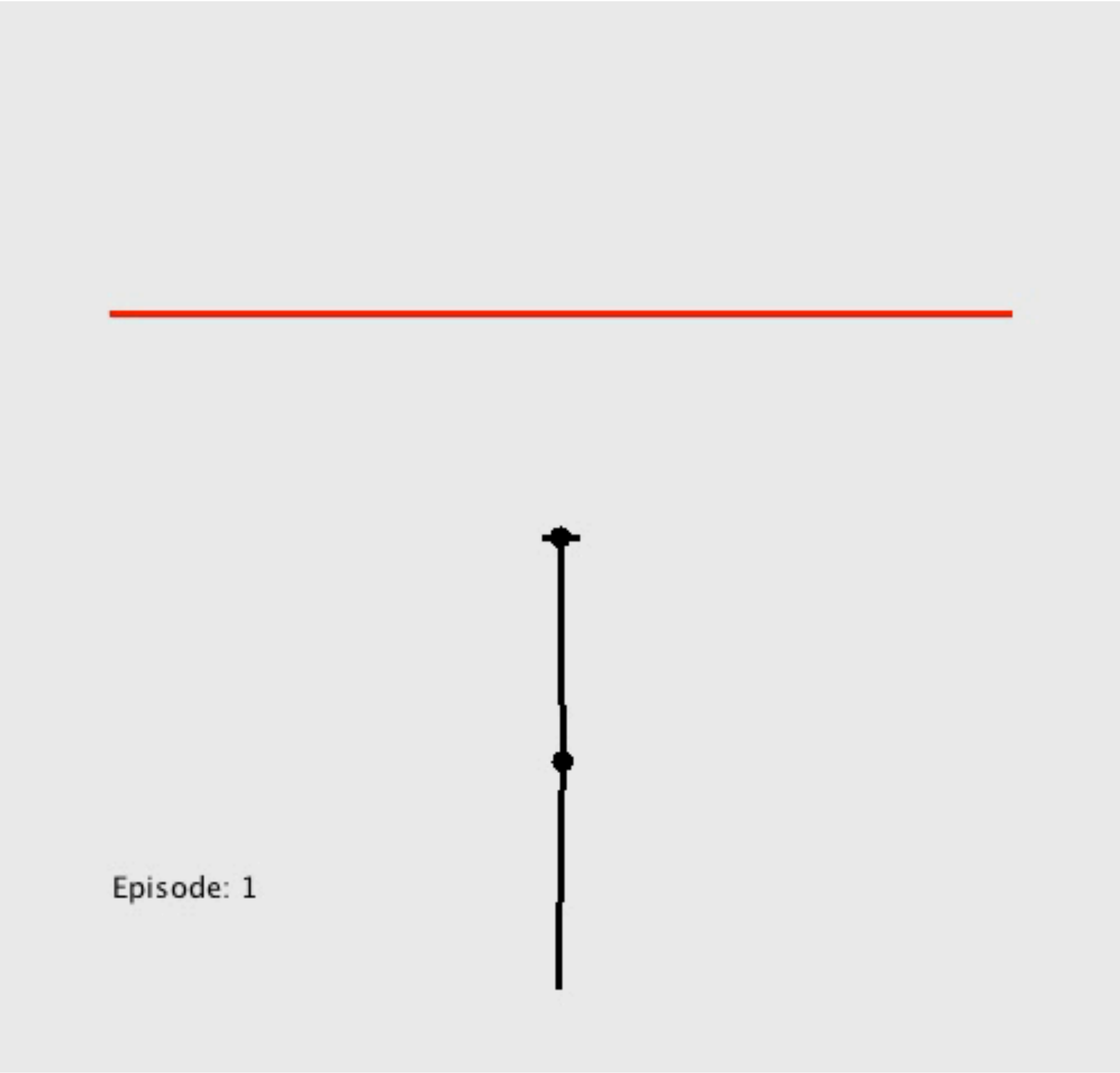
RL

Example:

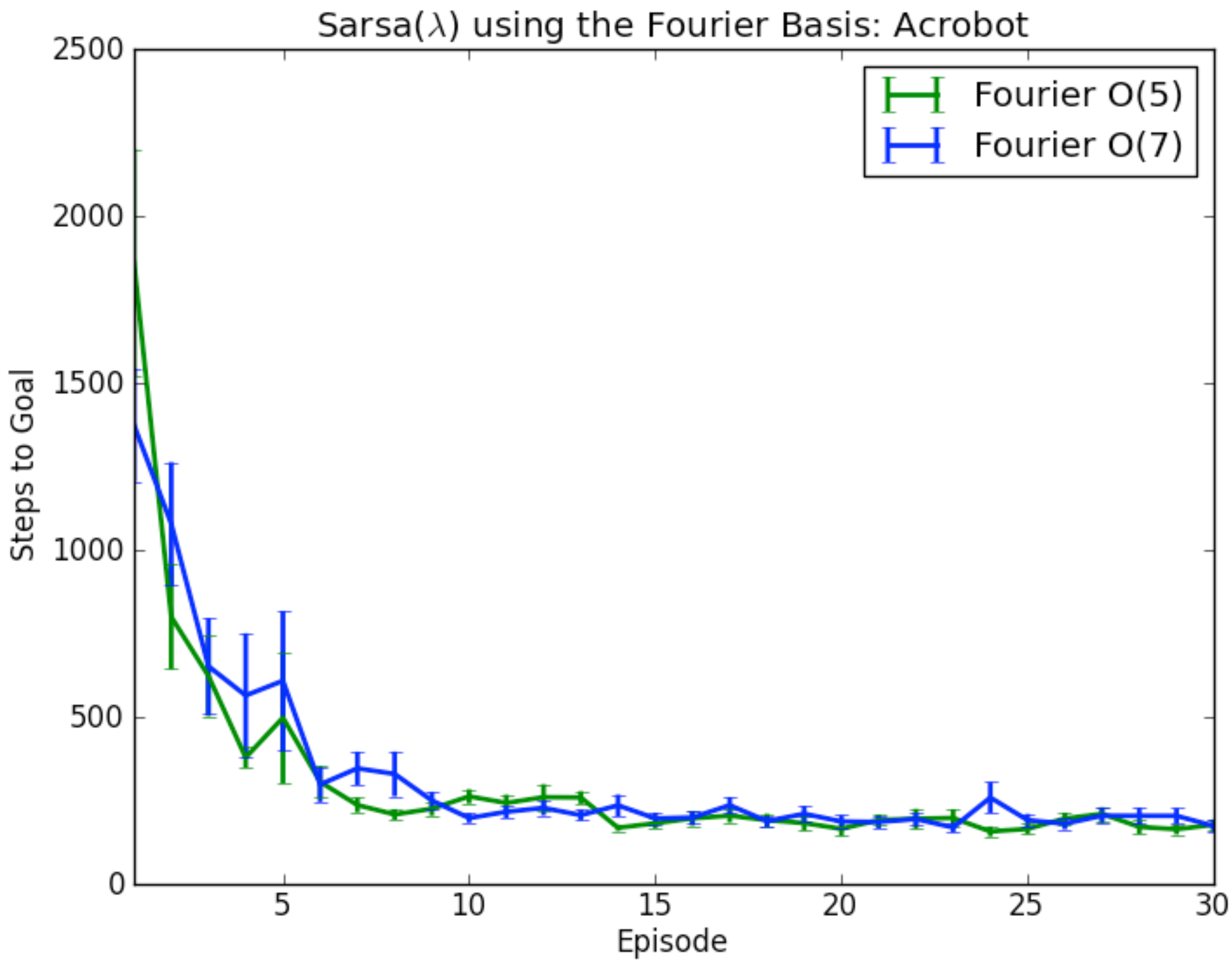


States: $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ (real-valued vector)
Actions: +1, -1, 0 units of torque added to elbow
Transition function: physics!
Reward function: -1 for every step

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

Least-Squares TD

Minimize:

$$\min_w \sum_{i=0}^{n} (r_i + \gamma w \cdot \phi(s_{i+1}, a_{i+1}) - w \cdot \phi(s_i, a_i))^2$$

Error function has a bowl shape, so *unique minimum*. Just go right there!

Least-Squares TD

Derivative set to zero:

$$\sum_{i=1}^{n} \left(w \cdot \phi(s_i, a_i) - r_i - \gamma w \cdot \phi(s_{i+1}, a_{i+1})\right) \phi(s_i, a_i)^T = 0$$

$$w^T \sum_{i=1}^{n} \left(w \cdot \phi(s_i, a_i) - \gamma w \cdot \phi(s_{i+1}, a_{i+1})\right) \phi^T(s_i, a_i) = \sum_{i=1}^{n} r_i \phi^T(s_i, a_i)$$

$$w = A^{-1} b$$

$$A = \sum_{i=1}^{n} \left(\phi(s_i, a_i) - \gamma \phi(s_{i+1}, a_{i+1})\right) \phi^T(s_i, a_i)$$

$$b = \sum_{i=1}^{n} r_i \phi^T(s_i, a_i)$$

[Bradtke and Barto, 1996]

LSTD(λ)

Can derive the least-squares version of LSTD(λ) in this way. Try it at home!

- Write down the objective function …
- Sample r_i replaced by complex reward estimate.

- You will get a trace vector if you do some clever algebra.
- Trace vector is the same size as w.

[Boyan, 1999]

LSTD(λ)


One inversion solves for *w*!

But:

- Computationally expensive.
- *A* may not be invert-able.
- Least-squares behavior sometimes unstable outside of data.

- LSPI: Least Squares Policy Iteration
- Requires recomputing *A* over historical data.
  - a_{i+1} changes with the policy

[Lagoudakis and Parr, 2003]

Linear Methods Don't Scale

Why not?

- They're complete.
- They have nice properties (bowl-shaped error).
- They are easy to use!

How many basis functions in a complete *n*th order Taylor series of *d* variables?

$$(n+1)^d$$

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.

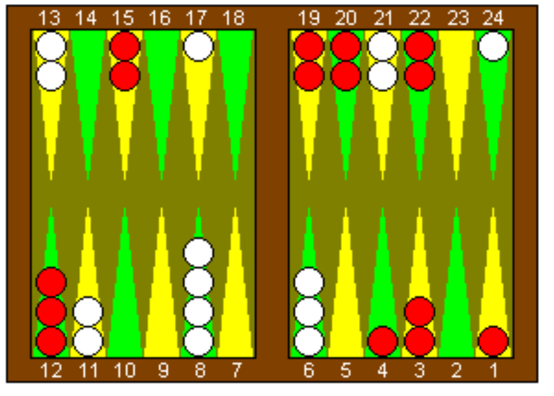
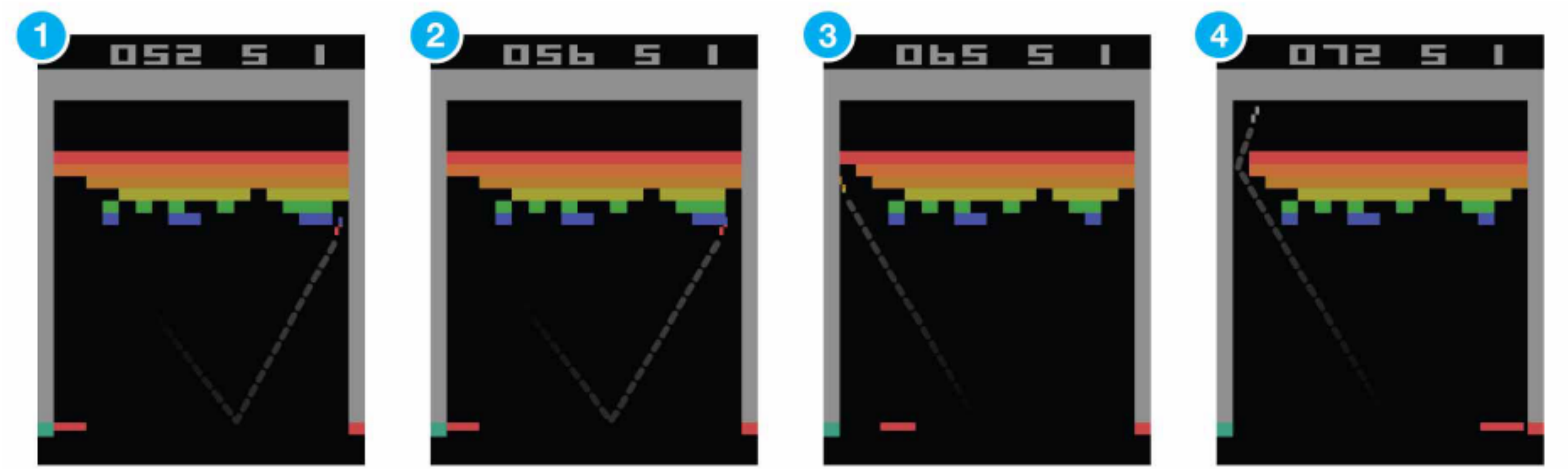
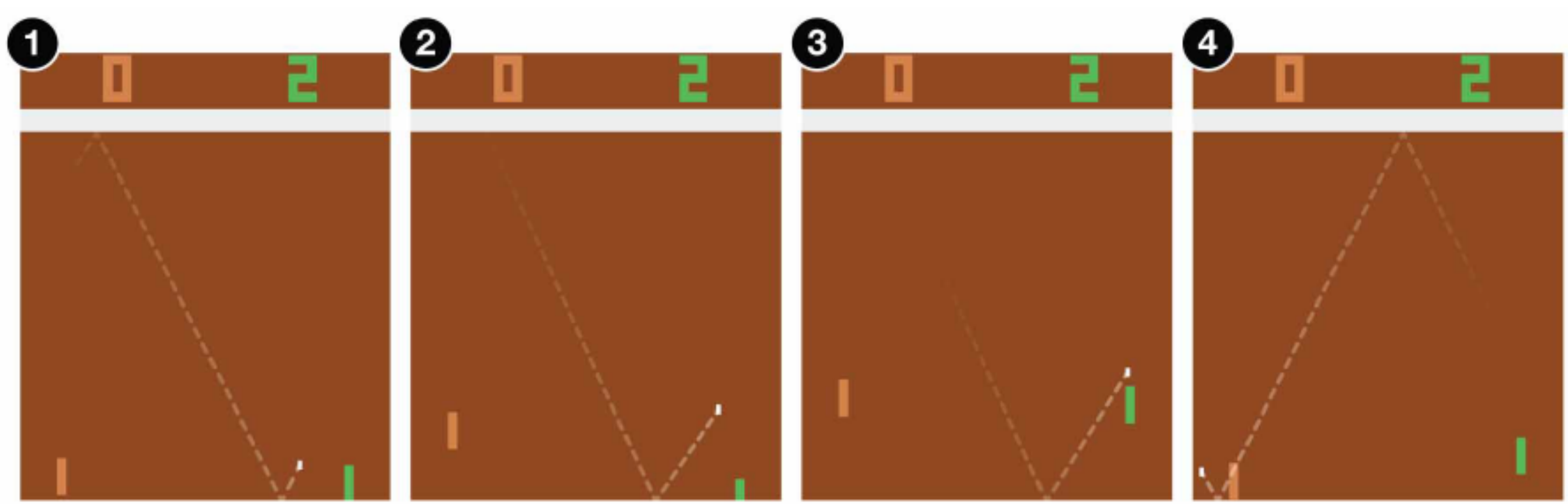


Figure 3. A complex situation where TD-Gammon's positional judgment is apparently superior to traditional expert thinking. White is to play 4-4. The obvious human play is 8-4*, 8-4, 11-7, 11-7. (The asterisk denotes that an opponent checker has been hit.) However, TD-Gammon's choice is the surprising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

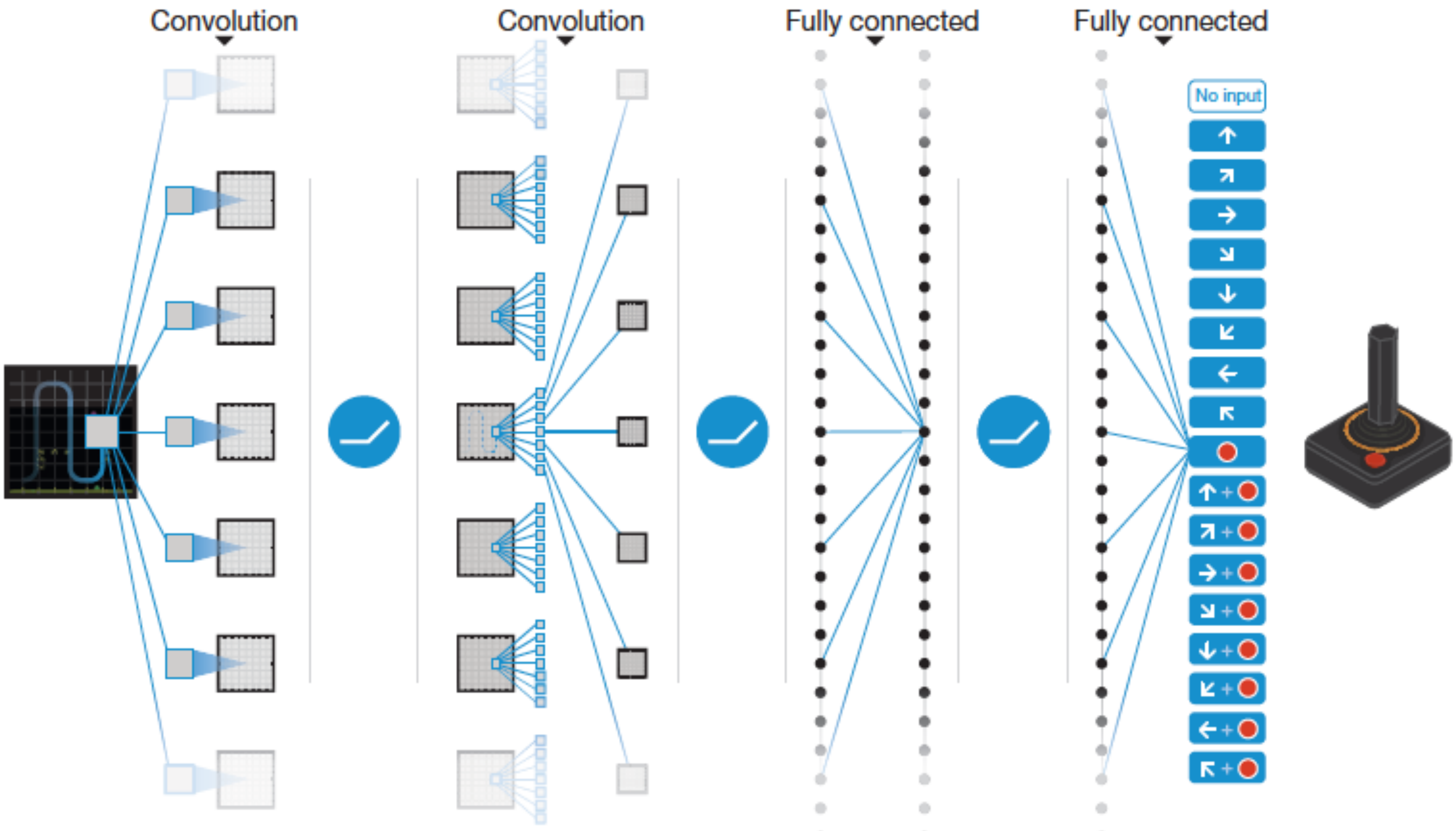
Arcade Learning Environment





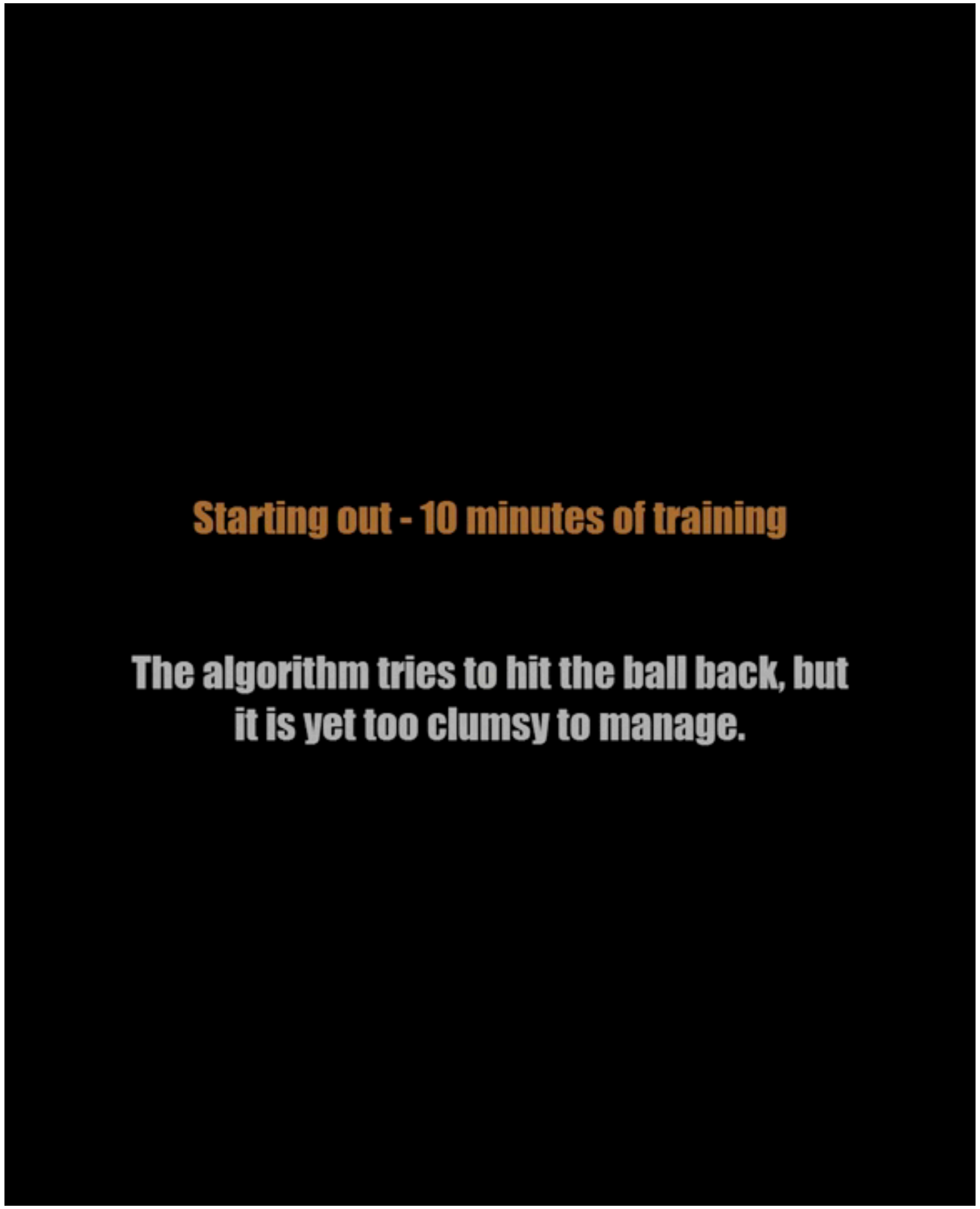
[Bellemare 2013]

Deep Q-Networks



[Mnih et al., 2015]

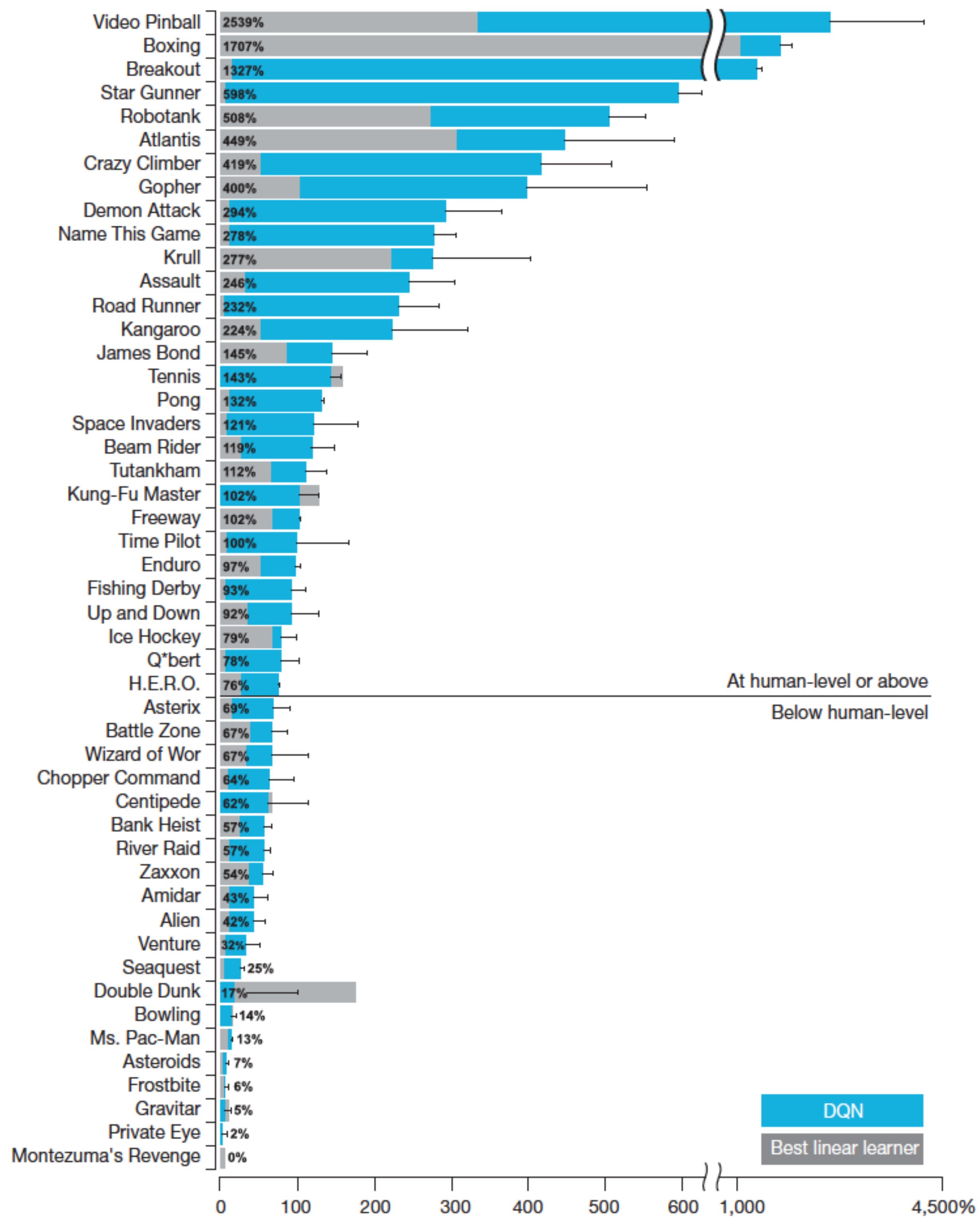
Atari



[Mnih et al., 2015]

video: Two Minute Papers

Atari



[Mnih et al., 2015]

POLICY SEARCH

Policy Search

Represent policy directly:

$$\pi(s, a, \theta) : \mathbb{R}^n, \mathbb{R}^m \rightarrow [0, 1]$$

parameter vector

Objective function:

$$\max_{\theta} \mathbb{E}\left[R = \sum_{i=0}^{\infty} \gamma^i r_i\right]$$

Why?

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.

Hill Climbing

What if you can't differentiate π?

Sample-based optimization:

- Sample some θ values near your current best θ.
- Adjust your current best to the highest value θ.

Aibo Gait Optimization

from Kohl and Stone, ICRA 2004.

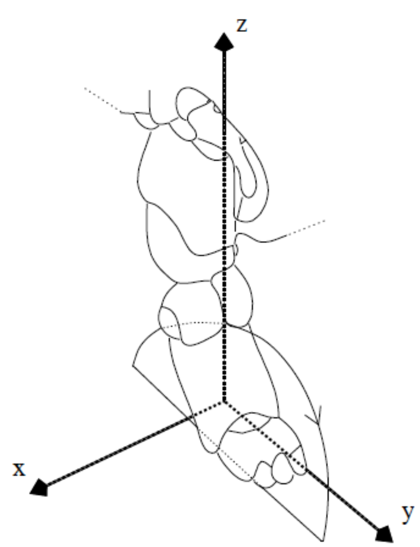


Fig. 2. The elliptical locus of the Aibo's foot. The half-ellipse is defined by length, height, and position in the x-y plane.

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

PoWER and PI2

More recently, two closely related algorithms:

- Generate some sample θ values.
- Next θ is sum of prior samples weighted by reward.

(Theodorou and Schaal 2010, Kober and Peters 2011)

Policy Search

What if we can differentiate π with respect to θ?

Policy gradient methods.

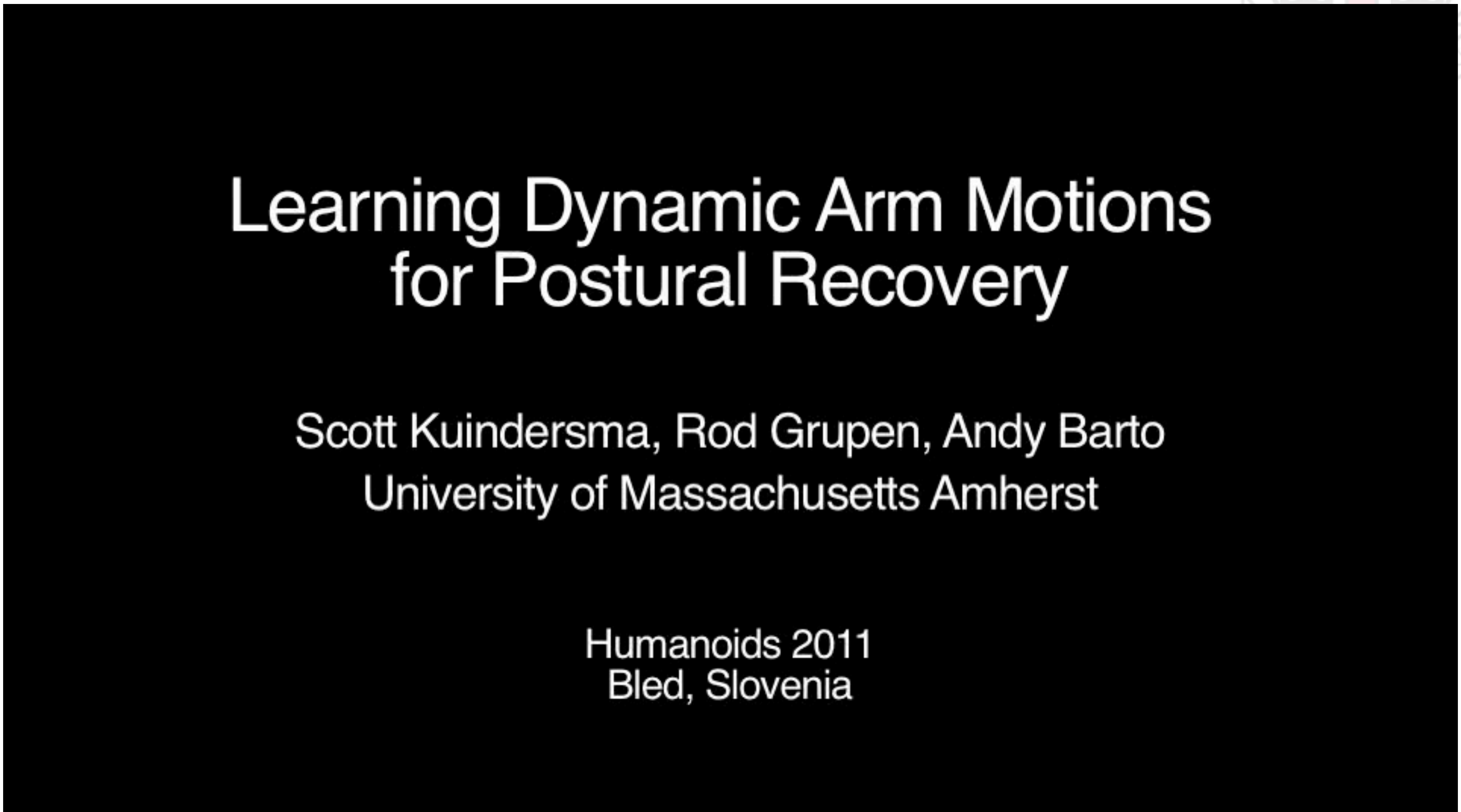
- Compute and ascend $\partial R/\partial\theta$
- This is the gradient of return w.r.t policy parameters

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.

Postural Recovery



Deep Policy Search

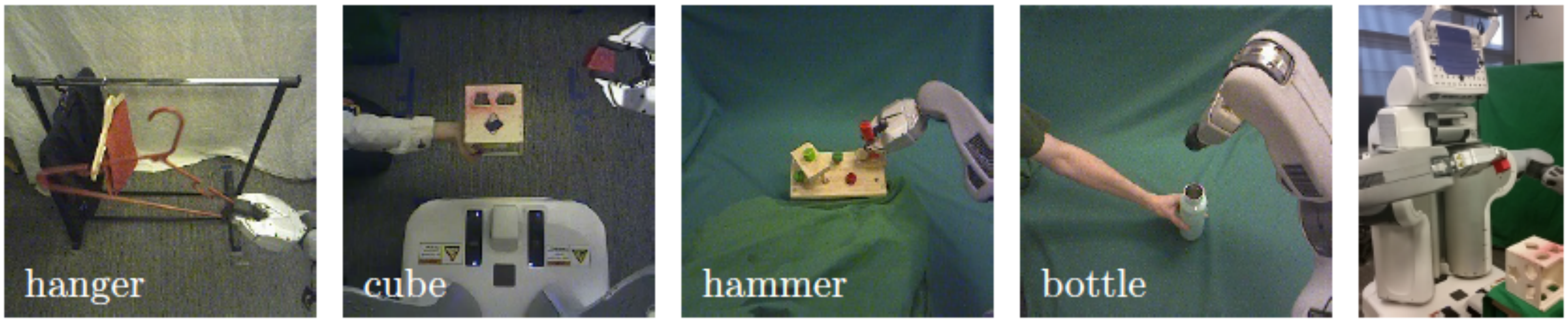
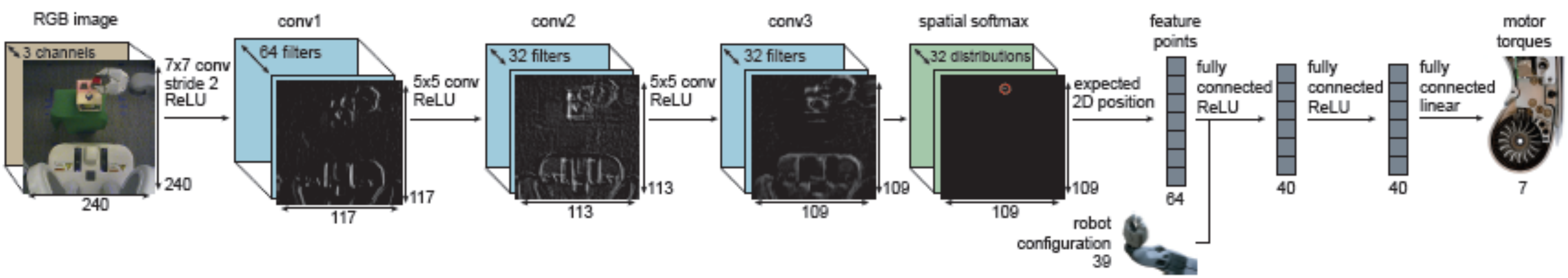
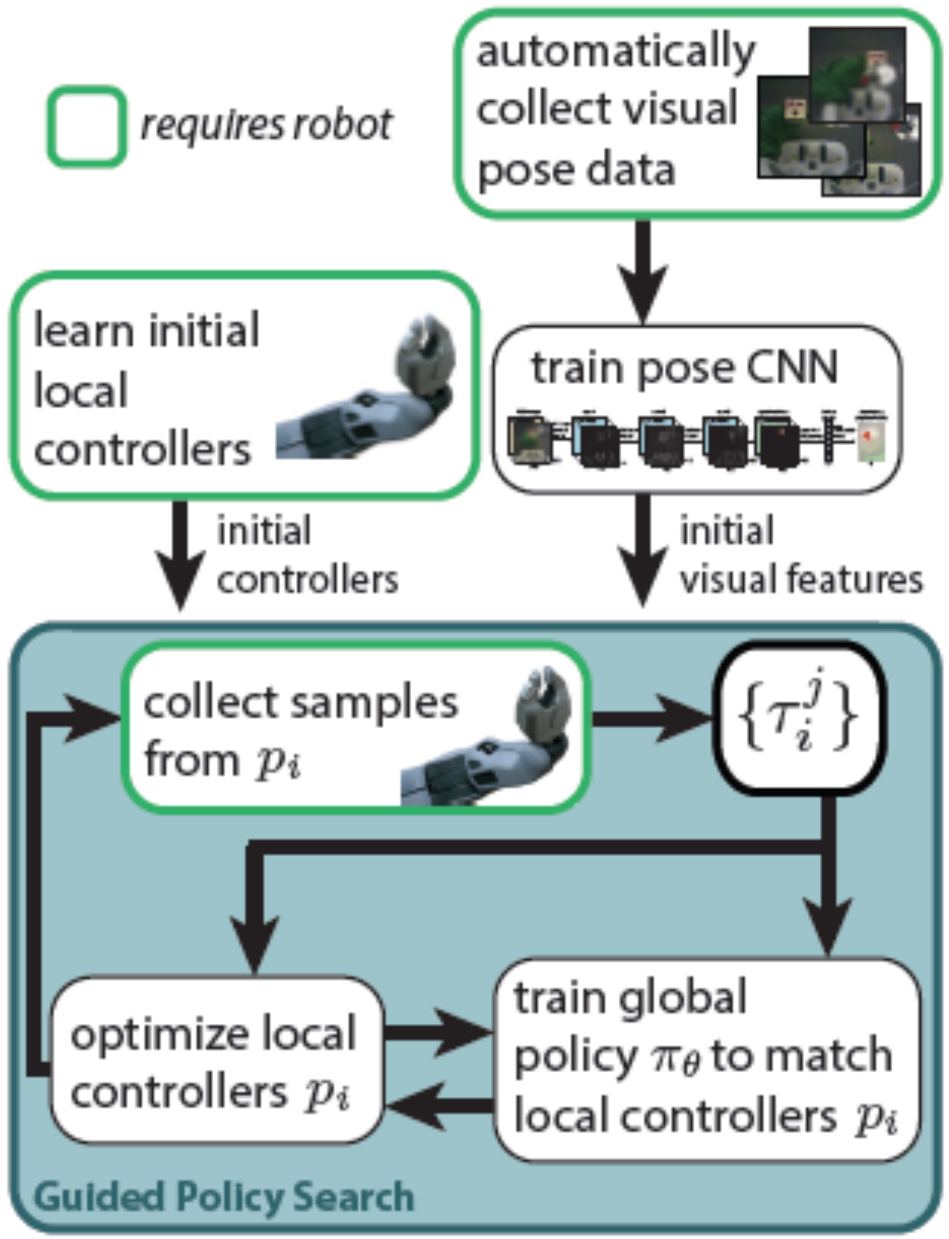


Figure 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).



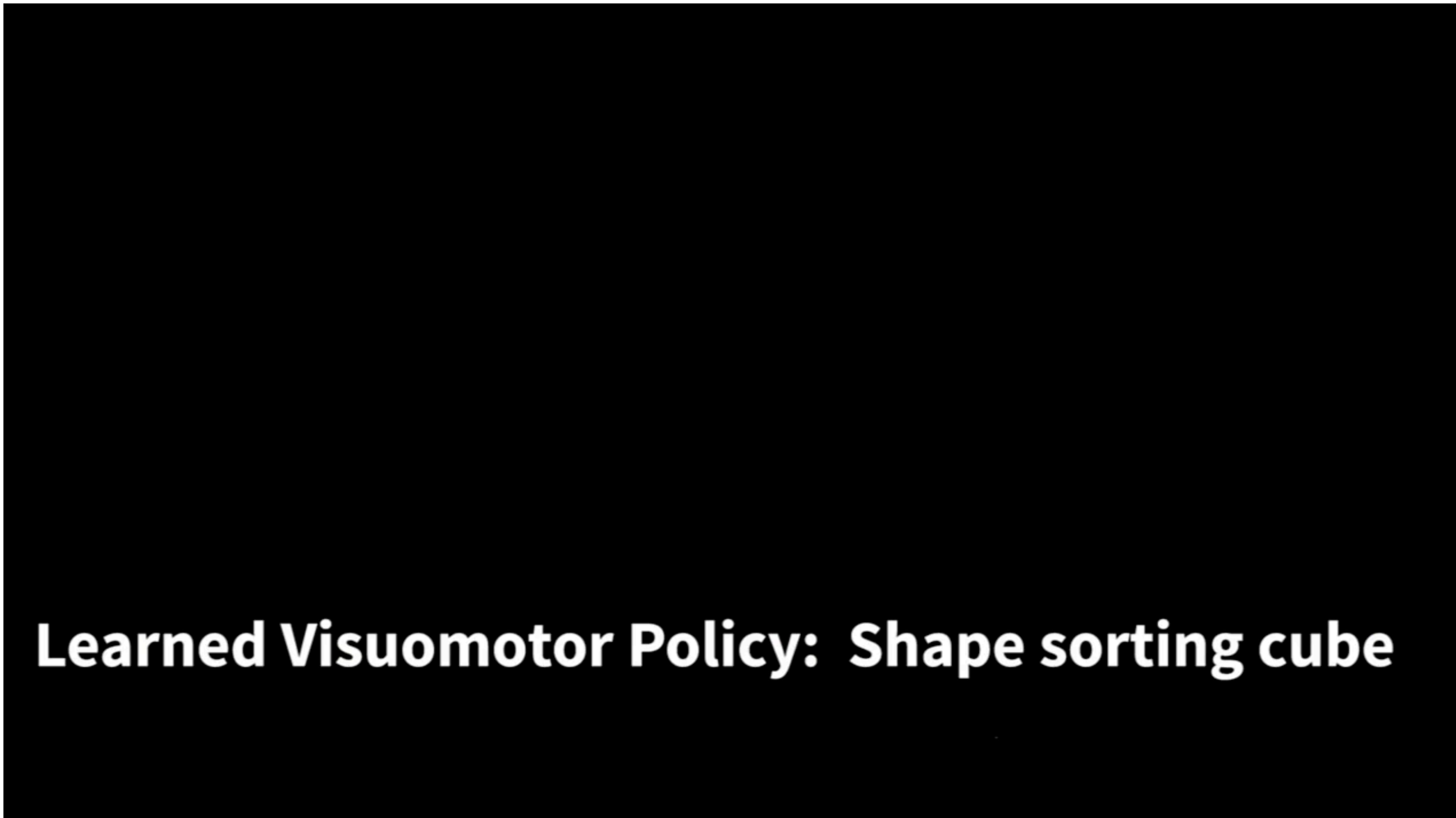
[Levine et al., 2016]

Deep Policy Search



[Levine et al., 2016]

Robotics



[Levine et al., 2016]

Reinforcement Learning

Very active area of current research, applications in:

- Robotics
- Operations Research
- Computer Games
- Theoretical Neuroscience

AI

- The primary function of the brain is control.