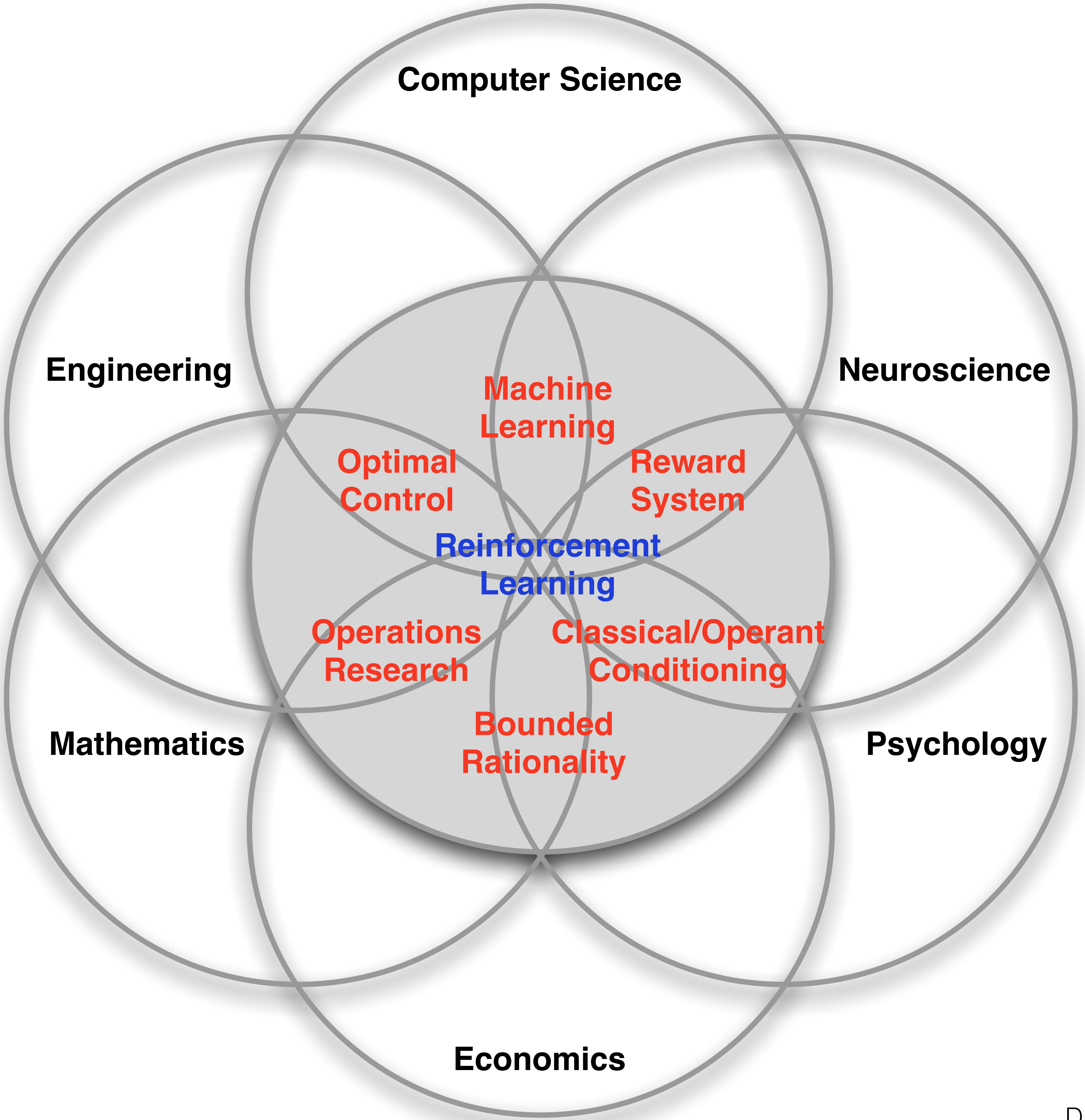
Introduction to Reinforcement Learning with Function Approximation

Rich Sutton
Reinforcement Learning and Artificial Intelligence Laboratory
Alberta Centre for Machine Learning
Department of Computing Science
University of Alberta
Canada

(with thanks to David Silver and Michael Littman for some slides and ideas)

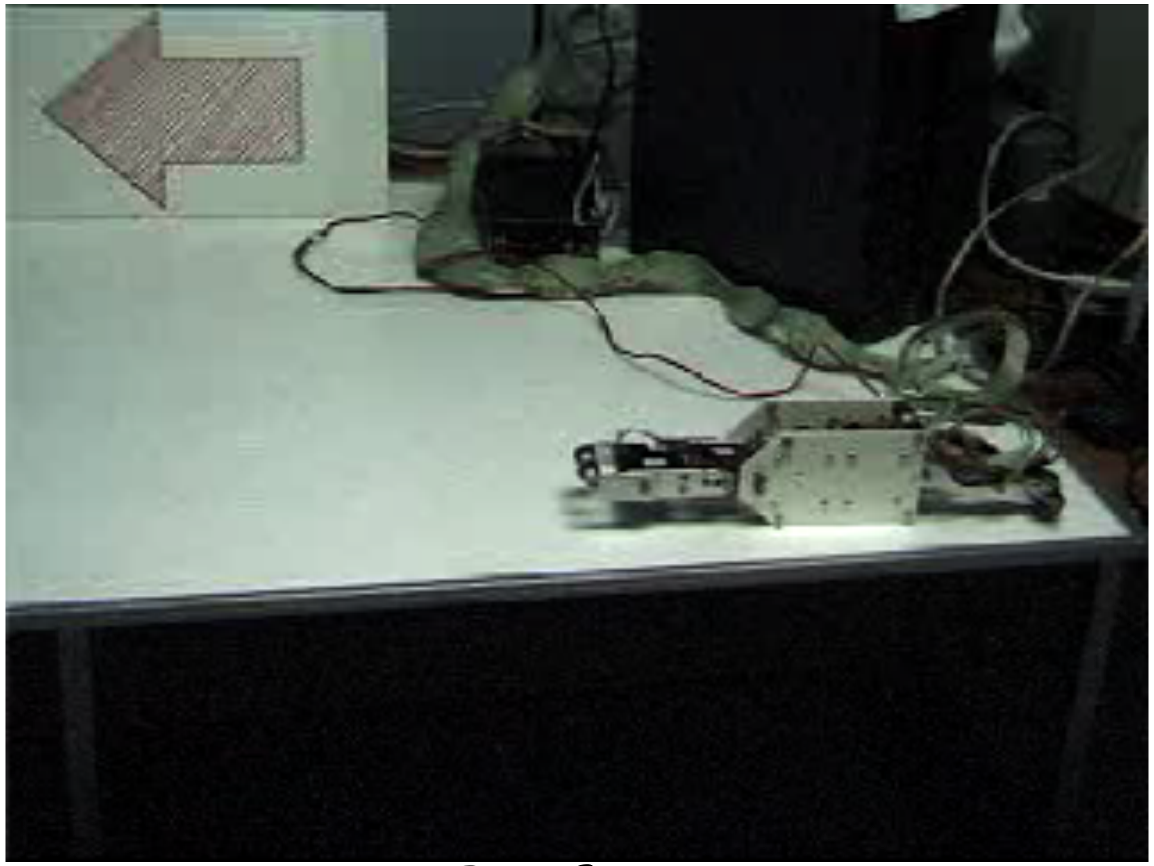
What is Reinforcement Learning?

- Agent-oriented learning—learning by interacting with an environment to achieve a goal
  - more realistic and ambitious than other kinds of machine learning
- Learning by trial and error, with only delayed evaluative feedback (reward)
  - the kind of machine learning most like natural learning
  - learning that can tell for itself when it is right or wrong
- The beginnings of a *science of mind* that is neither natural science nor applications technology



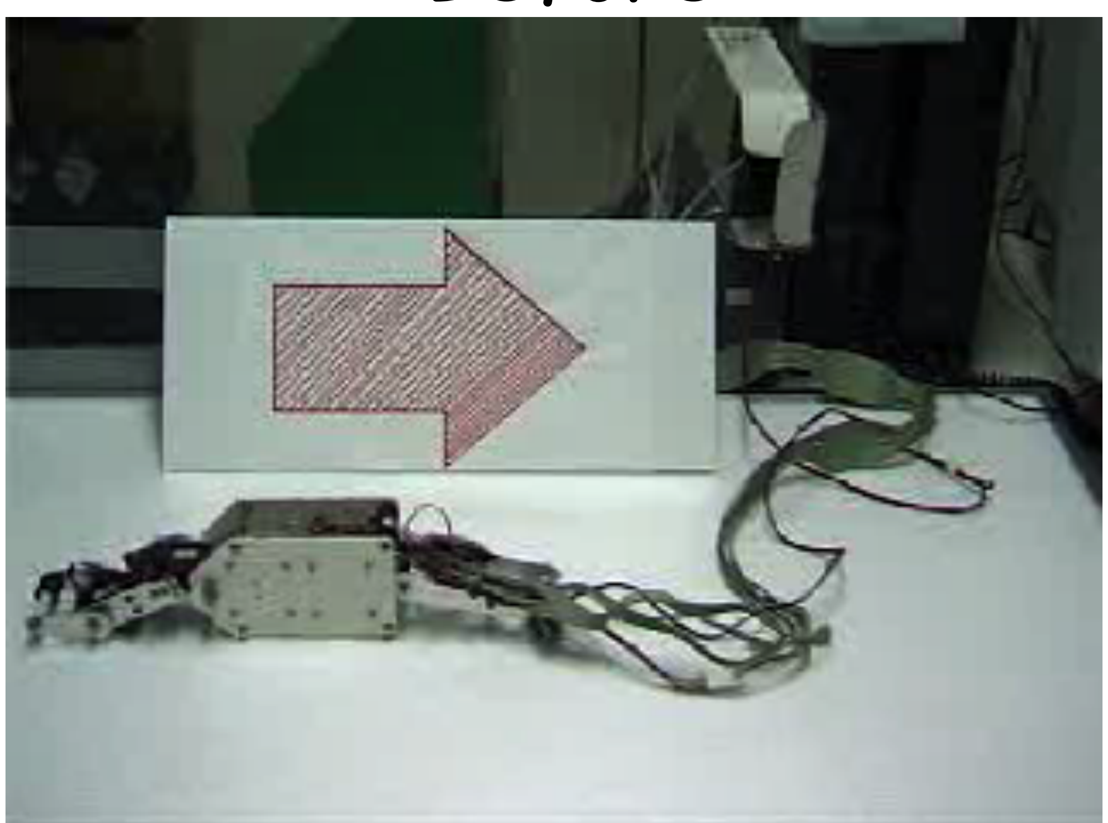
David Silver 2015

Example: Hajime Kimura's RL Robots

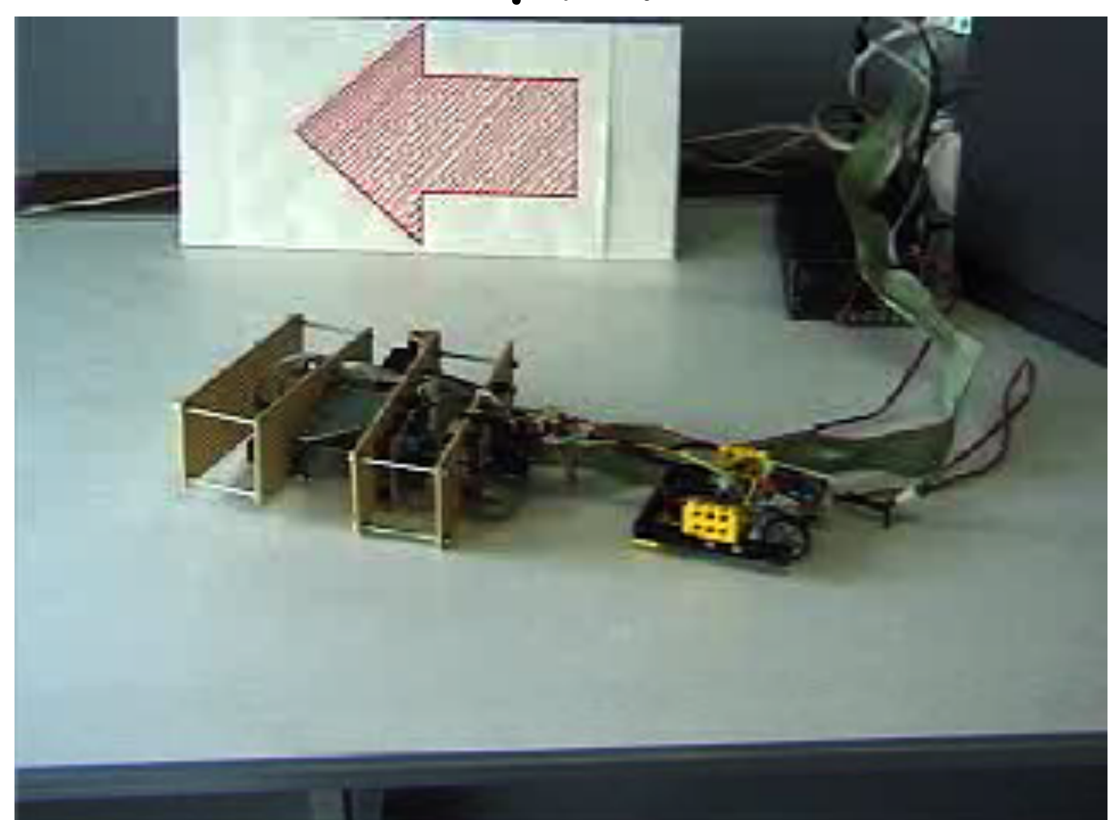


Before

After

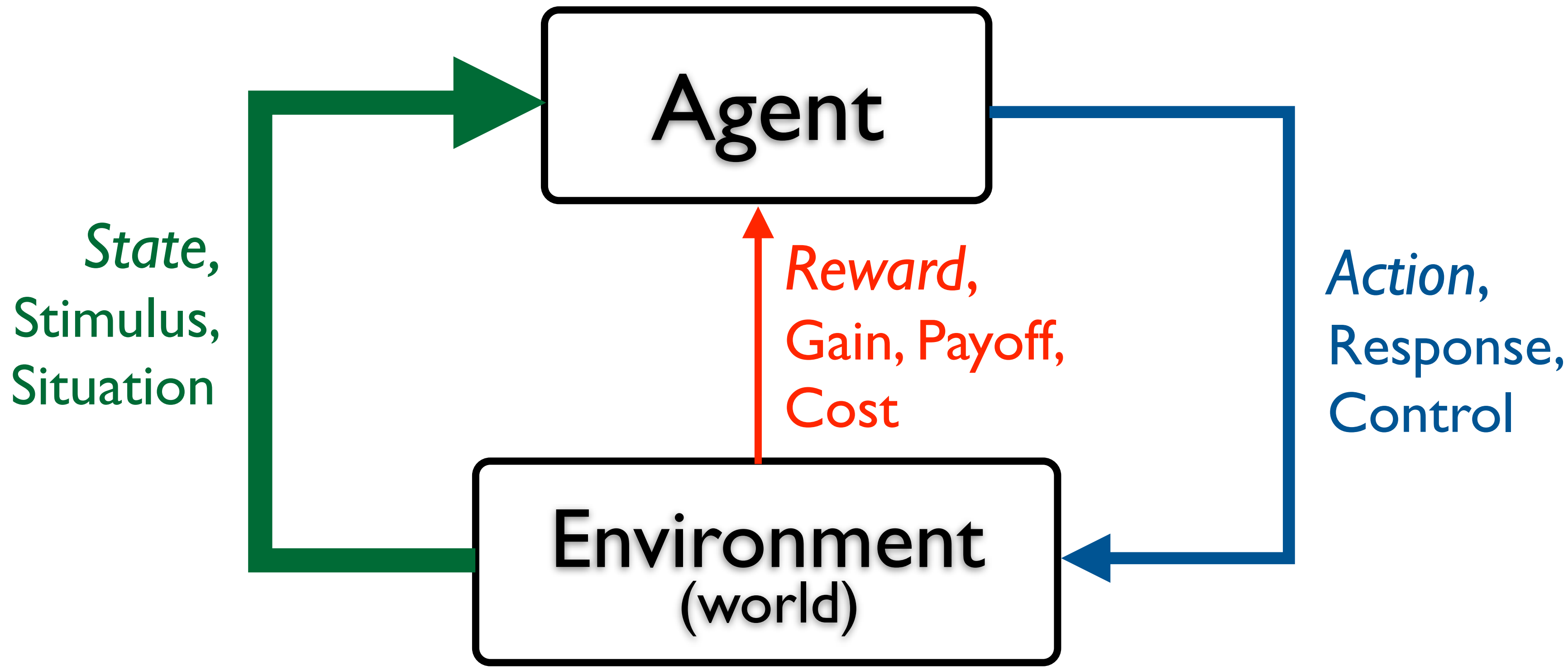


Backward



New Robot, Same algorithm

The RL Interface



- Environment may be unknown, nonlinear, stochastic and complex
- Agent learns a policy mapping states to actions
  - Seeking to maximize its cumulative reward in the long run

Signature challenges of RL

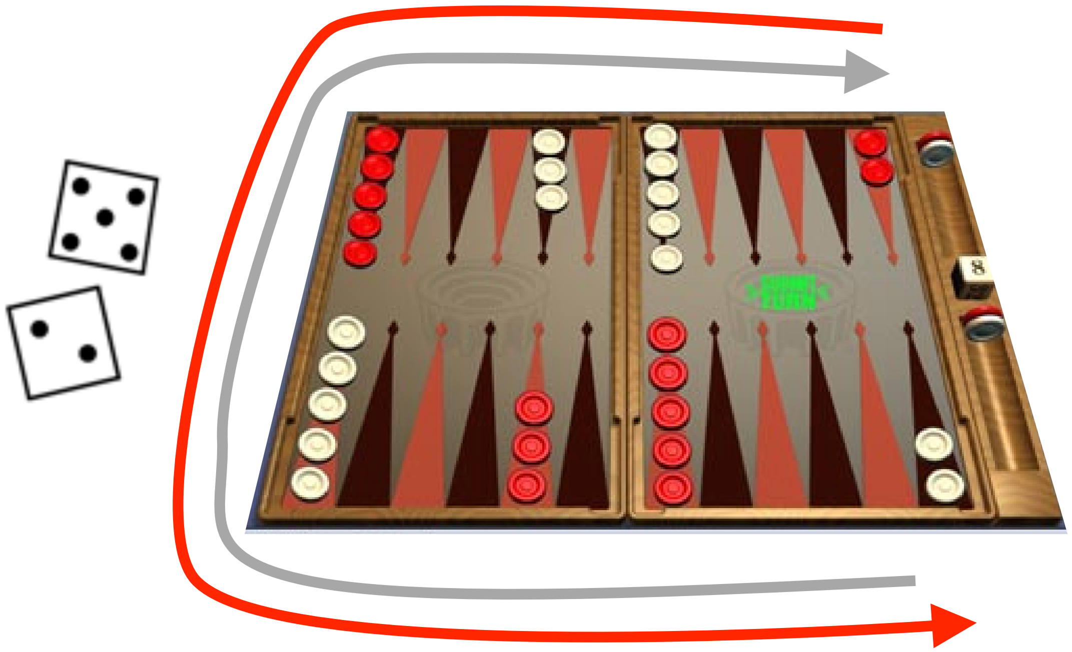
- Evaluative feedback (reward)
- Sequentiality, delayed consequences
- Need for trial and error, to explore as well as exploit
- Non-stationarity
- The fleeting nature of time and online data

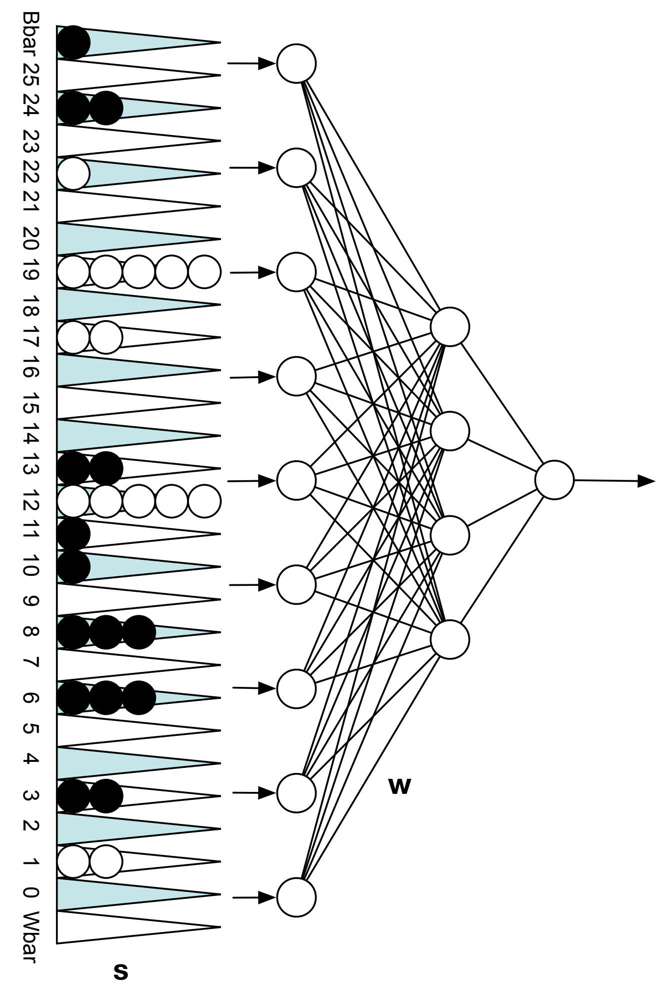
Some RL Successes

- Learned the world's best player of Backgammon (Tesauro 1995)
- Learned acrobatic helicopter autopilots (Ng, Abbeel, Coates et al 2006+)
- Widely used in the placement and selection of advertisements and pages on the web (e.g., A-B tests)
- Used to make strategic decisions in *Jeopardy!* (IBM's Watson 2011)
- Achieved human-level performance on Atari games from pixel-level visual input, in conjunction with deep learning (Google Deepmind 2015)
- In all these cases, performance was better than could be obtained by any other method, and was obtained without human instruction

Example: TD-Gammon

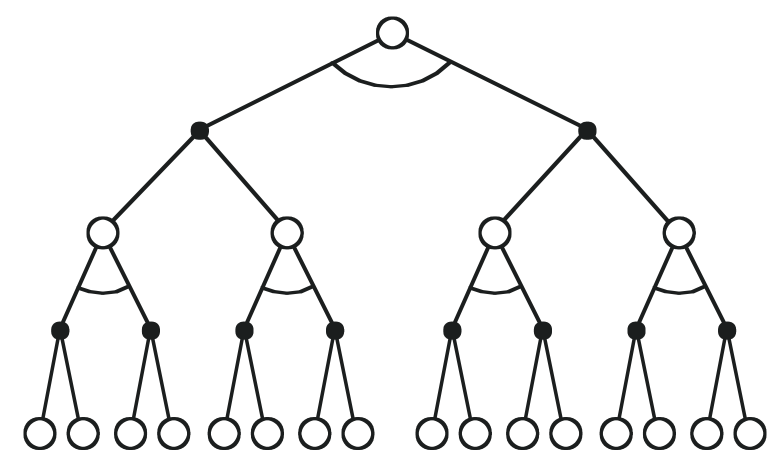
Tesauro, 1992-1995





estimated state value
(≈ prob of winning)

Action selection
by a shallow search



Start with a random Network

Play millions of games against itself

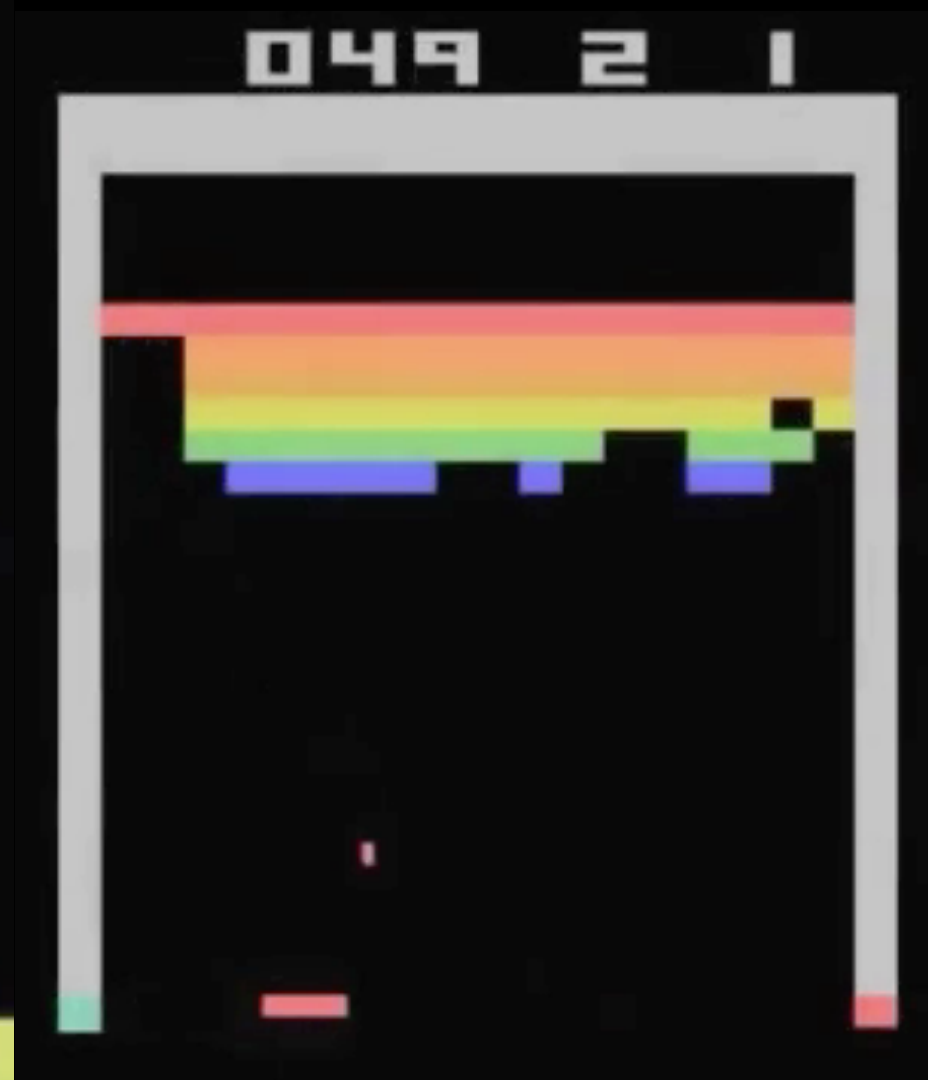
Learn a value function from this simulated experience

Six weeks later it's the best player of backgammon in the world

Originally used expert handcrafted features, later repeated with raw board positions

RL + Deep Learing Performance on Atari Games

Space Invaders

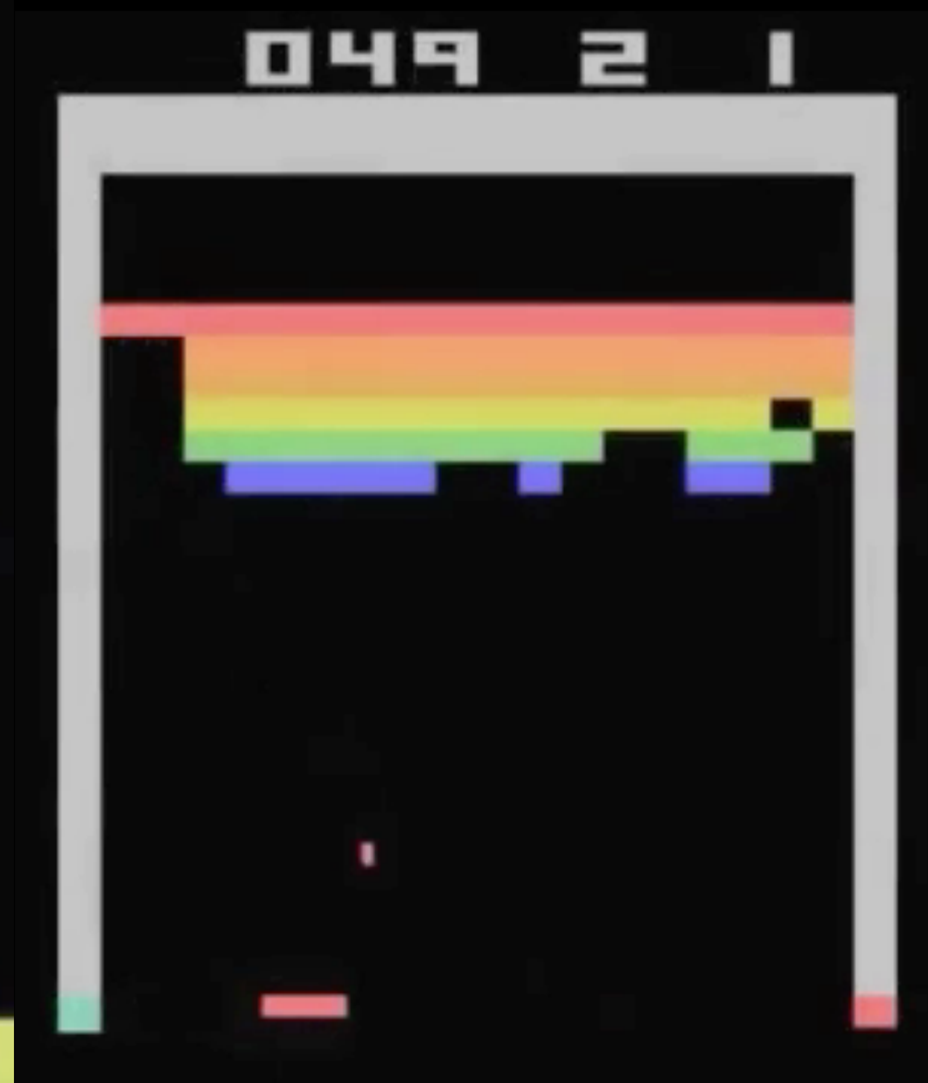


Breakout

Enduro

RL + Deep Learing Performance on Atari Games

Space Invaders

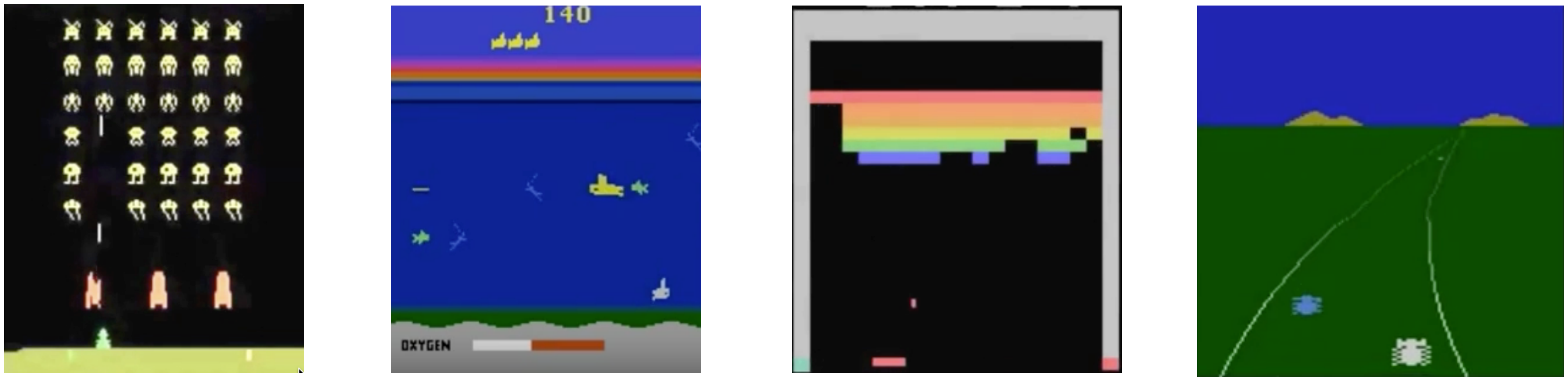


Breakout

Enduro

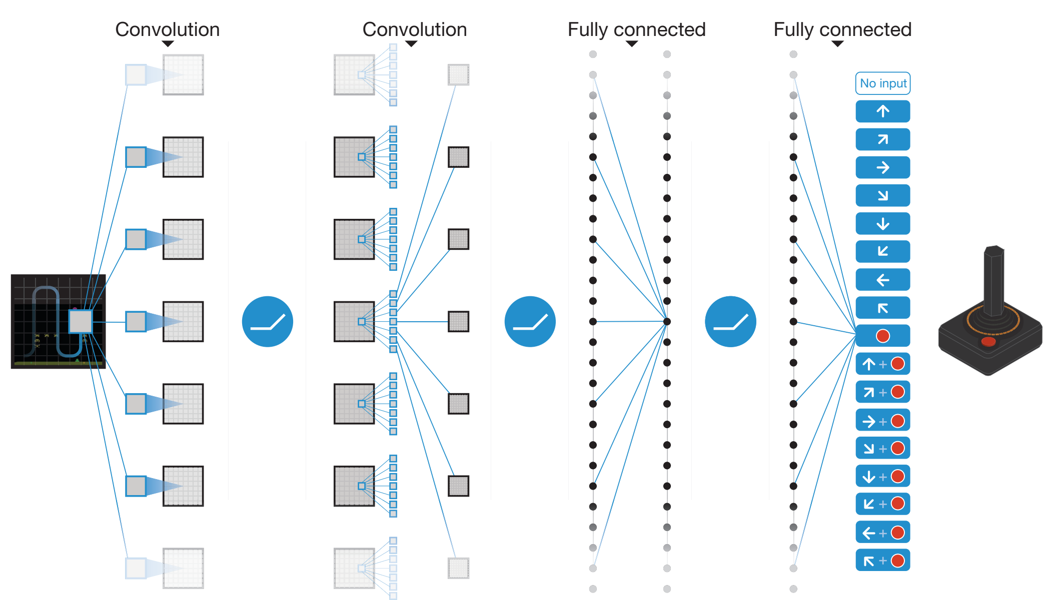
RL + Deep Learning, applied to Classic Atari Games

Google Deepmind 2015, Bowling et al. 2012



- Learned to play 49 games for the Atari 2600 game console, without labels or human input, from self-play and the score alone

mapping raw screen pixels



to predictions of final score for each of 18 joystick actions

- Learned to play better than all previous algorithms and at human level for more than half the games

Same learning algorithm applied to all 49 games! w/o human tuning

Outline

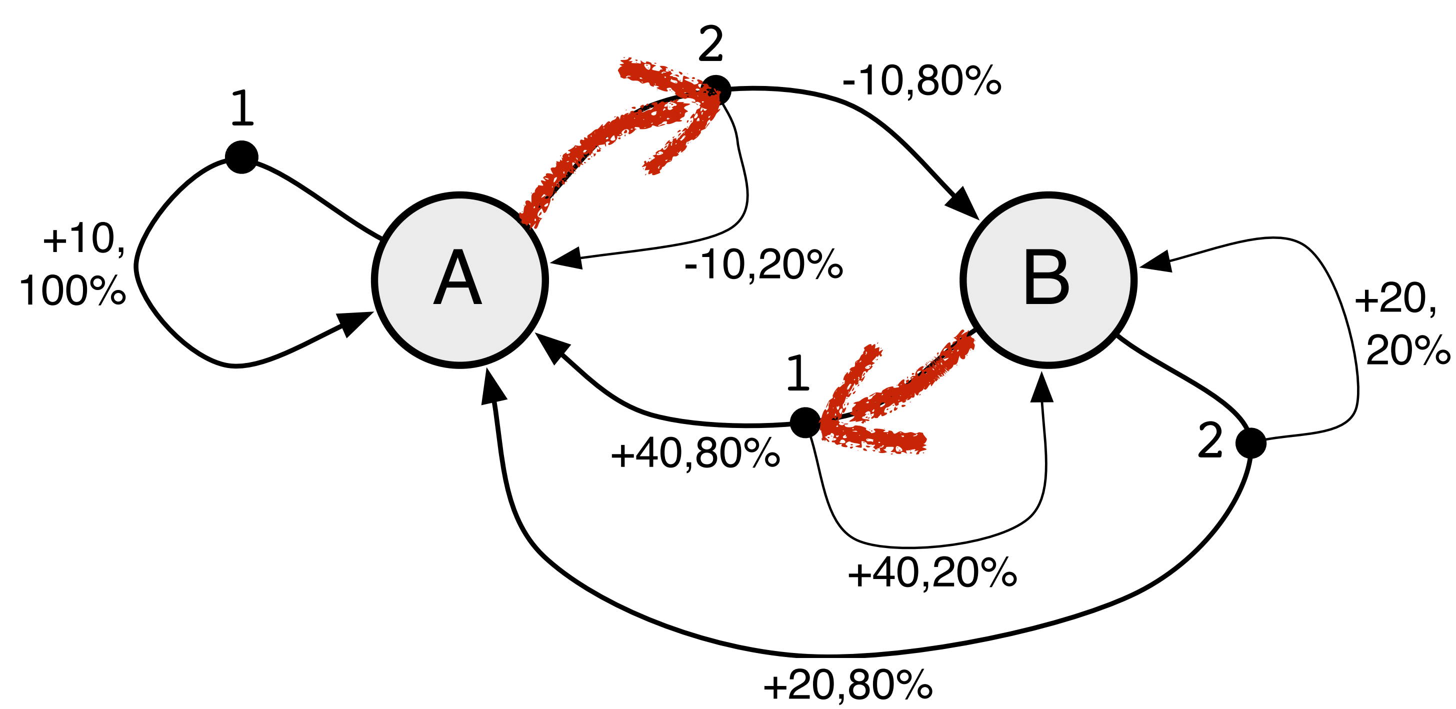
- Introduction to RL successes and challenges
- The formal problem: Finite Markov decision processes
- Part I: Exact solution methods and core theoretical ideas
- Part II: Approximate solution methods
  - Semi-gradient methods
  - On-policy and off-policy methods
  - The deadly triad; how to evade or survive it
- Miscellany and closing remarks

You are the reinforcement learner! (interactive demo)

Optimal policy (deterministic)

State	Action
A ⟶	2
B ⟶	1

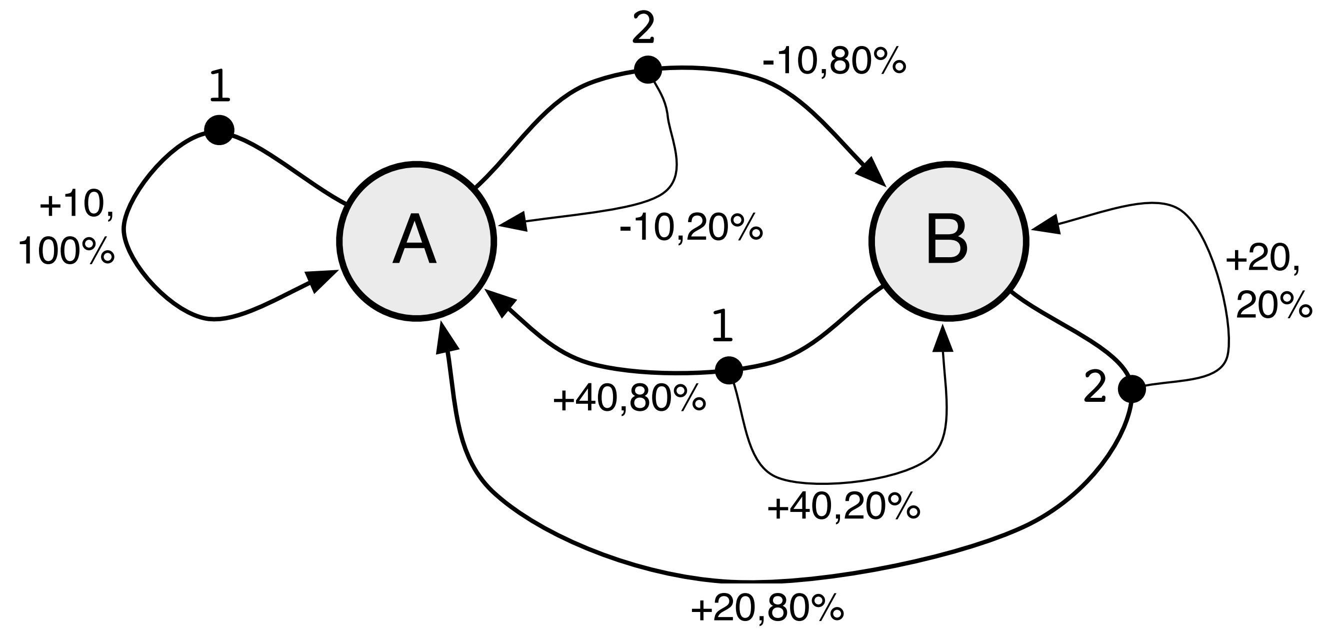
True model of the world



Howard, 1960

The Environment: A Finite Markov Decision Process (MDP)

- Discrete time $t = 1, 2, 3, \ldots$
- A finite set of states
- A finite set of actions
- A finite set of rewards



- Life is a trajectory:

$$\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots$$

- With arbitrary Markov (stochastic, state-dependent) dynamics:

$$p(r, s'|s, a) = Prob\Big[R_{t+1} = r, S_{t+1} = s' \;\Big|\; S_t = s, A_t = a\Big]$$

Policies

e.g.

State	Action
A $\longrightarrow$	2
B $\longrightarrow$	1

- Deterministic policy

$$a = \pi(s)$$

- An agent following a policy

$$A_t = \pi(S_t)$$

The number of deterministic policies is *exponential* in the *number of states*

- Informally the agent's goal is to choose each action so as to maximize the discounted sum of future rewards,

to choose each A_t to maximize $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$

- We are searching for a policy

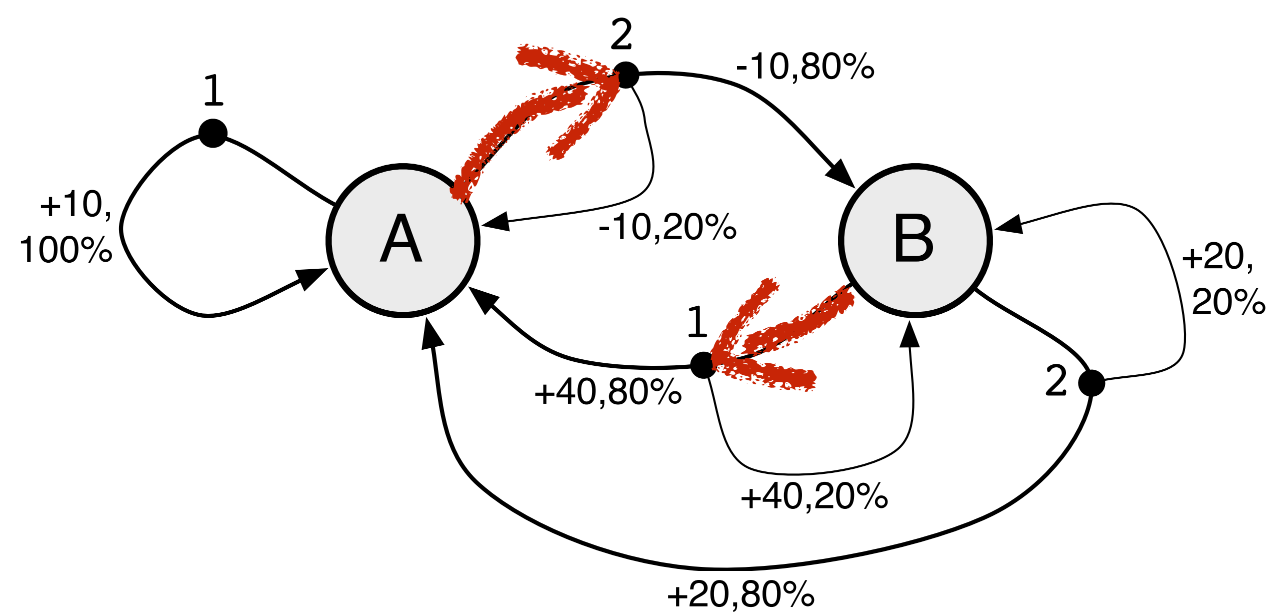
Action-value functions

- An action-value function says how good it is to be in a state, take an action, and thereafter follow a policy:

$$q_\pi(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \;\Big|\; S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\Big]$$

Action-value function for the optimal policy and γ=0.9

State	Action	Value
A	1	130.39
A	2	133.77
B	1	166.23
B	2	146.23



Optimal policies

- A policy π_* is optimal if it maximizes the action-value function:

$$q_{\pi_*}(s, a) = \max_{\pi} q_{\pi}(s, a) = q_*(s, a)$$

- Thus all optimal policies share the same optimal value function

- Given the optimal value function, it is easy to act optimally:

$$\pi_*(s) = \arg\max_{a} q_*(s, a)$$

"greedification"

- We say that the optimal policy is greedy with respect to the optimal value function

- There is always at least one deterministic optimal policy

Part I

Exact Solution Methods
(tabular methods)

Q-learning, the simplest RL algorithm

1. Initialize an array $Q(s, a)$ arbitrarily

2. Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

3. On each time step, change one element of the array:

$$\Delta Q(S_t, A_t) = \alpha\Big(\underbrace{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)}_{\text{target}} - Q(S_t, A_t)\Big)$$

4. If desired, reduce the step-size parameter α over time

- Theorem: For appropriate choice of 4, Q converges to q_*, and its greedy policy to an optimal policy π_* (Watkins & Dayan 1992)

- This is kind of amazing — learning long-term optimal behavior without any model of the environment, for arbitrary MDPs!

Demo

Off-policy learning gridworld

Policy improvement theorem

- Given the value function for *any policy* π:

$$q_\pi(s,a) \qquad \text{for all } s,a$$

- It can always be greedified to obtain a *better policy*:

$$\pi'(s) = \arg\max_a q_\pi(s,a)$$

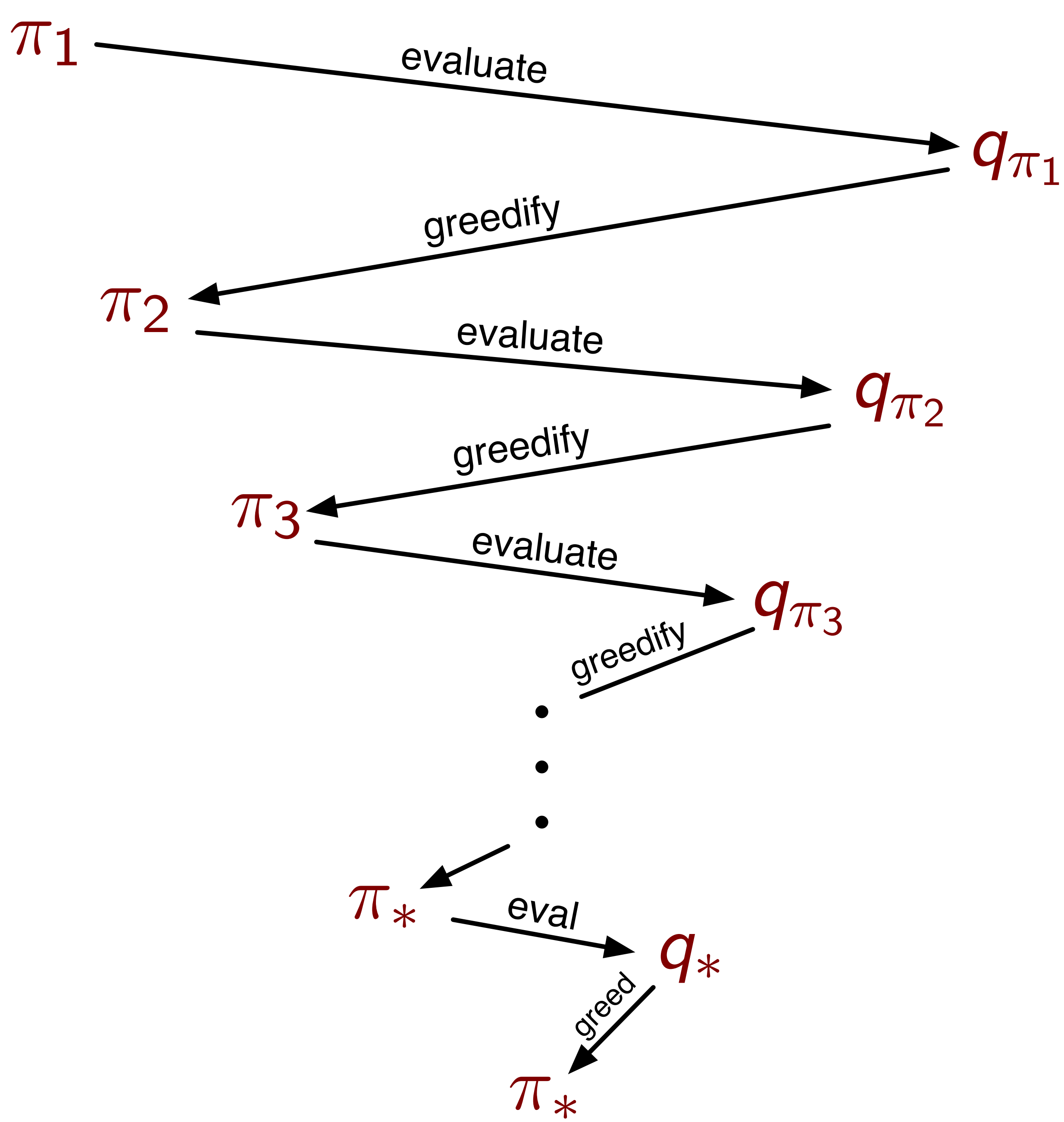
(π' is not unique)

- where better means:

$$q_{\pi'}(s,a) \geq q_\pi(s,a) \quad \text{for all } s,a$$

- with equality only if <u>both policies are optimal</u>

The dance of policy and value (Policy Iteration)



Any policy evaluates to a unique value function (soon we will see how to learn it)

which can be greedified to produce a better policy

That in turn evaluates to a value function

which can in turn be greedified…

Each policy is *strictly better* than the previous, until *eventually both are optimal*

There are *no local optima*

The dance converges in a finite number of steps, usually very few

The dance is very robust

- to initial conditions
- to delayed and asynchronous updating, as in parallel and distributed implementations
- to incomplete evaluation and greedification
  - updating only some states but not others
  - updating only part of the way
- to randomization and noise
- in particular, it works if only a single state is updated at a time by a random amount that is only correct in expectation

The Explore/Exploit dilemma

- You can't do the action that you think is best all the time
  - because you will miss out big—forever—if you are wrong
  - to find the real best action, you must explore them all…an infinite number of times!
- You also can't explore all the time
  - because then you would never get any advantage of your learning
- Thus you must both explore and exploit, but neither to excess. What is the right balance?

How did Q-learning escape the dilemma?

Q-learning, the simplest RL algorithm

1. Initialize an array $Q(s,a)$ arbitrarily

2. Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

3. On each time step, change one element of the array:

$$\Delta Q(S_t, A_t) = \alpha\Big(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big)$$

4. If desired, reduce the step-size parameter α over time

Bootstrapping

- The key idea underlying both dynamic programming (DP) and all temporal-difference (TD) learning

- Updating an estimate from an estimate, a guess from a guess

- Based on the Bellman expectation equation:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \,\middle|\, S_t = s, A_t = a, A_{t+1:\infty} \sim \pi \right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \,\middle|\, S_t = s, A_t = a, A_{t+1} \sim \pi \right] \end{aligned}$$

- or the Bellman optimality equation:

$$q_*(s,a) = \mathbb{E}\Big[\underbrace{R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')}_{\text{Q-learning's target for } Q(S_t, A_t)} \,\Big|\, S_t = s, A_t = a \Big]$$

Q-learning is off-policy learning

- Off-policy learning is learning about the value of a policy other than the policy being used to generate the trajectory
- Q-learning learns about the value of its deterministic greedy policy—which gradually become optimal—from data while behaving in a more exploratory manner
  - thus Q-learning is off-policy
  - and this is essential to its strategy for escaping the explore/exploit dilemma
- Some terminology
  - the target policy is the policy being learned about
  - the behavior policy is the policy generating the trajectory data
  - on-policy learning is when the two policies are the same

Part II

Approximate Solution Methods
(function approximation)

- So, RL finds optimal policies for arbitrary environments, if the value functions and policies can be exactly represented in tables
- But the real world is too large and complex for tables
- Will RL work with approximations?
- Will RL work with function approximators?

Function approximation

- Represent the action-value function by a parameterized function approximator with parameter $\boldsymbol{\theta}$

$$q(s, a, \boldsymbol{\theta}) \approx q_*(s, a) \quad \text{or} \approx q_\pi(s, a)$$

- The approximator could be a deep neural network, with the weights being the parameter
  - or simply a linear weighting of features (the most pressing theoretical problems are all best addressed in this setting)
- Function approximation is a powerful concept, e.g., subsuming much of the problem of hidden state
- For large applications, it is important that all computations scale linearly with the number of parameters

Does Q-learning work with function approximation?

- Yes, there is a obvious generalization of Q-learning to function approximation (Watkins 1989)
- Often, it works well
- But there are counterexamples
  - simple examples where the parameters diverge to infinity
  - even for linear function approximation
- We could get by, but something is not right, there is something, probably many things, that we are not understanding

Semi-gradient Q-learning (Watkins 1989)

- Consider the following objective function, based on the Bellman optimality equation:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}\left[\left(\underbrace{R_{t+1} + \gamma \max_{a} \hat{q}(S_{t+1}, a, \boldsymbol{\theta})}_{\text{target}} - \hat{q}(S_t, A_t, \boldsymbol{\theta})\right)^2\right]$$

- The target here depends on the parameter, but if we ignore that dependence when taking the derivative, then we get a semi-gradient Q-learning update:

$$\Delta\boldsymbol{\theta}_t = \alpha\left(R_{t+1} + \gamma \max_{a} \hat{q}(S_{t+1}, a, \boldsymbol{\theta}_t) - \hat{q}(S_t, A_t, \boldsymbol{\theta}_t)\right)\frac{\partial \hat{q}(S_t, A_t, \boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t}$$

Semi-gradient Sarsa (Rummery 1994, Sutton 1988)

- Consider instead an objective function based on the Bellman *expectation* equation:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}\left[\left(\underbrace{R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{\theta})}_{\text{target}} - \hat{q}(S_t, A_t, \boldsymbol{\theta})\right)^2\right]$$

- Again the target depends on the parameter, and again we ignore that dependence when taking the derivative, this time to get the semi-gradient Sarsa update:

$$\Delta\boldsymbol{\theta}_t = \alpha\Big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{\theta}_t) - \hat{q}(S_t, A_t, \boldsymbol{\theta}_t)\Big)\frac{\partial\hat{q}(S_t, A_t, \boldsymbol{\theta}_t)}{\partial\boldsymbol{\theta}_t}$$

- This is an on-policy algorithm: it approximates q_π not q_* ; thus π should be near greedy, typically it is ε-greedy

Why is it called Sarsa?

- It is the only learning update that uses exactly these five things from the trajectory:

$$\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots$$

- Sarsa is equivalent to the TD(0) algorithm (Sutton 1988) when applied to state-action pairs rather than to states

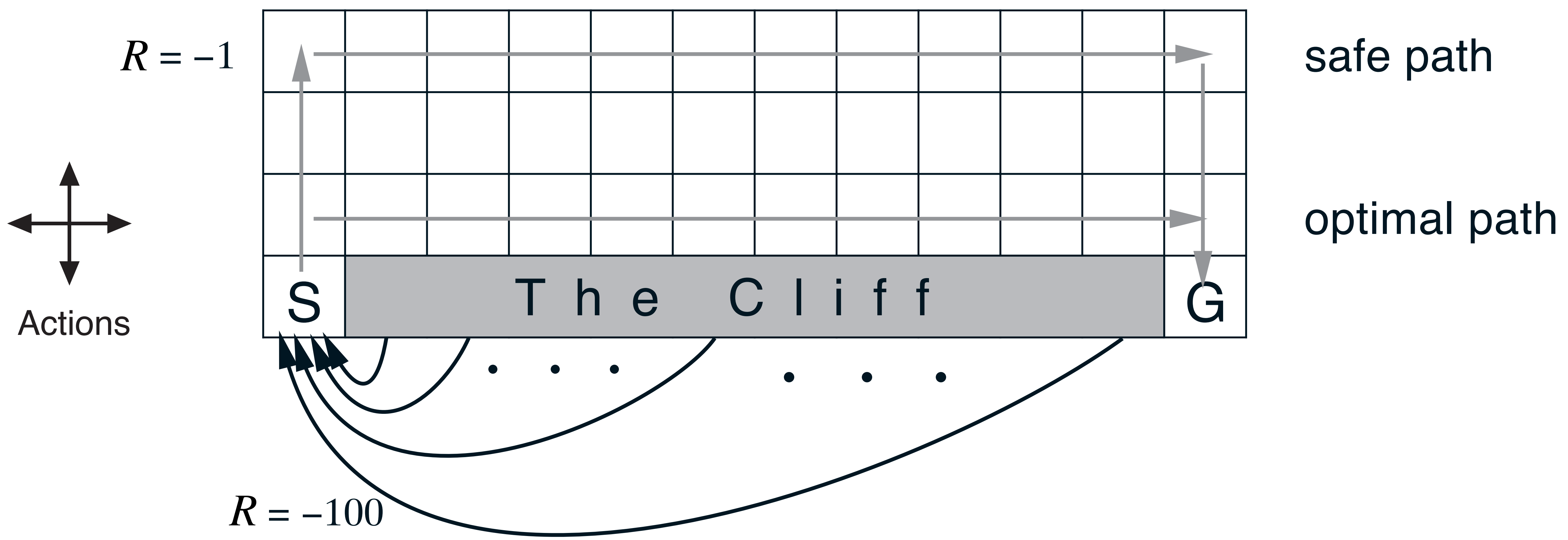
What is an ε-greedy policy?

- An ε-greedy policy is a *stochastic* policy that is *usually greedy*, but with small probability ε instead selects an action *at random*

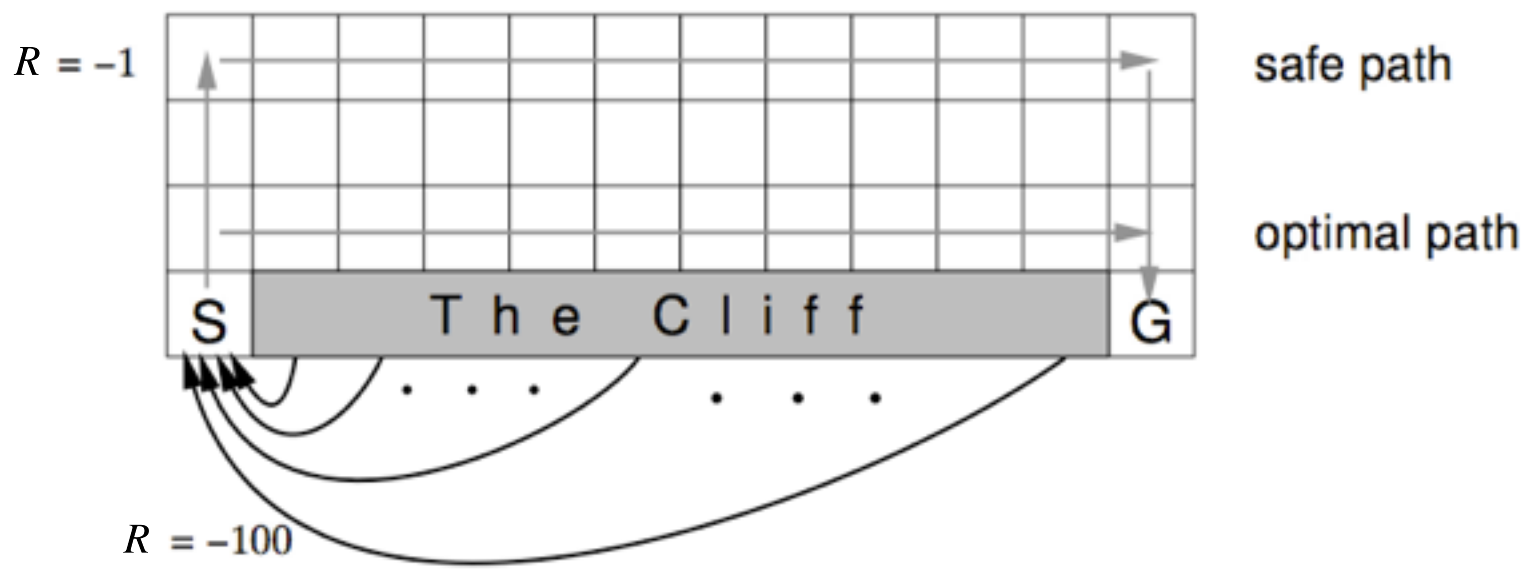
As an on-policy method, Semi-gradient Sarsa has good convergence properties

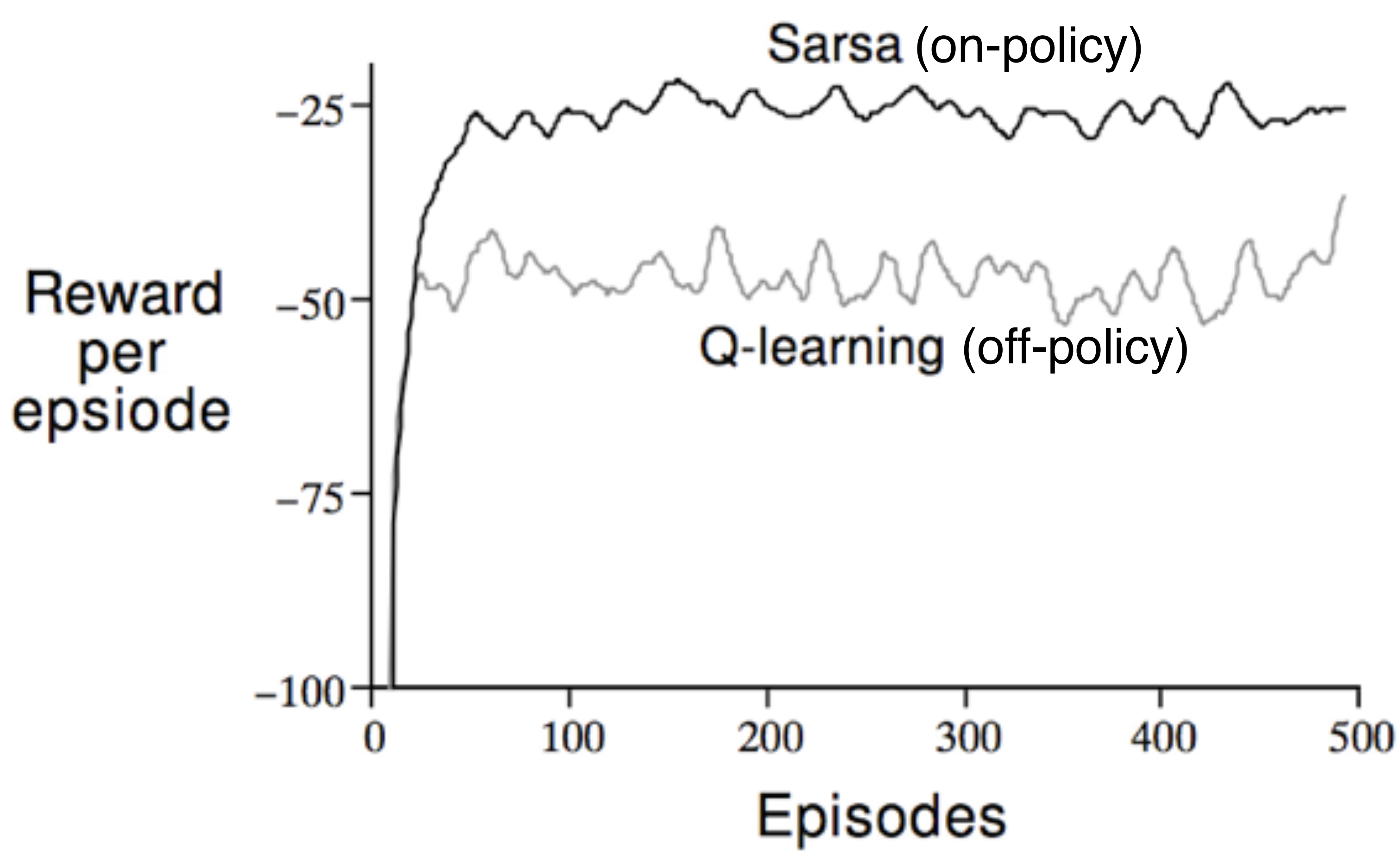
- If the function approximator is linear, it is
  - guaranteed convergent for prediction (fixed target policy) (Sutton 1988, Dayan 1992, Tsitsiklis & Van Roy 1997)
  - guaranteed non-divergent for control, with bounded error (though may “chatter” –Gordon 1995)
- For general non-linear function approximation, there is one known counterexample, but it is very artificial and contrived
- On-policy methods typically perform better than off-policy methods, but find poorer policies

Cliff-walking example (on-policy vs off-policy)



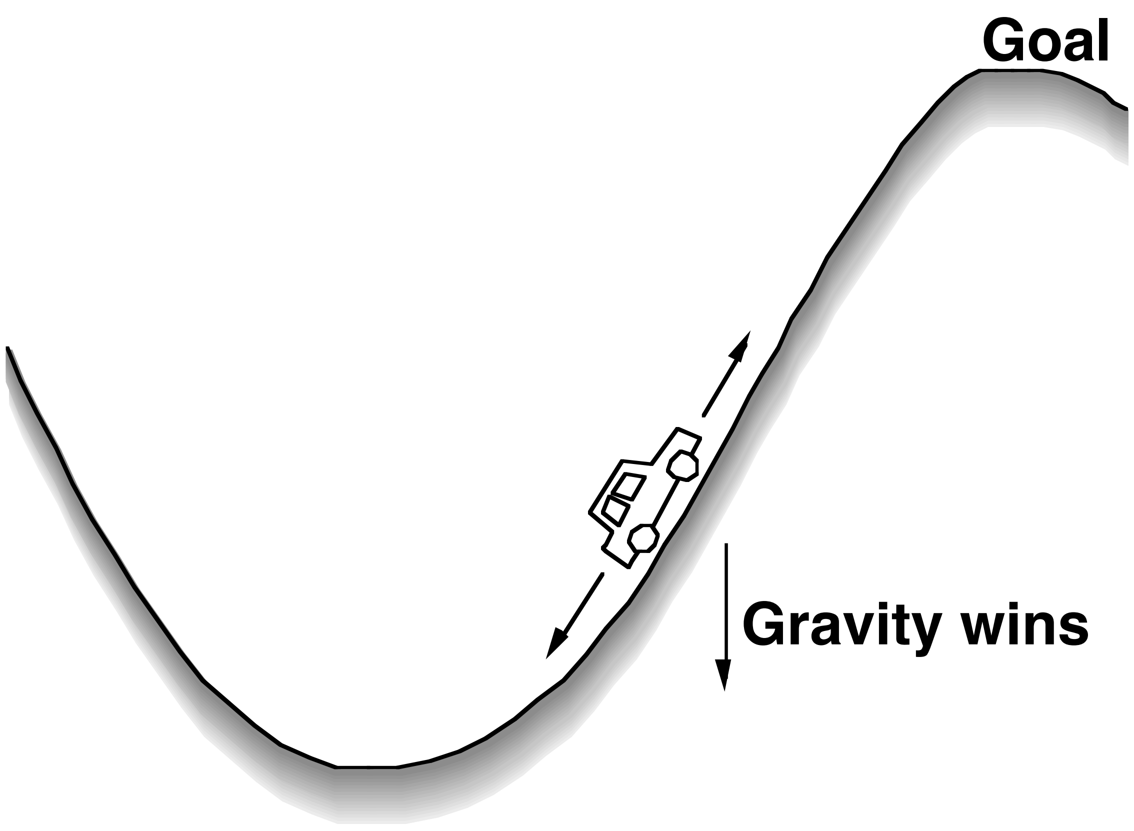
Cliff-walking example (on-policy vs off-policy)





both algorithms are ε−greedy
ε = 0.1

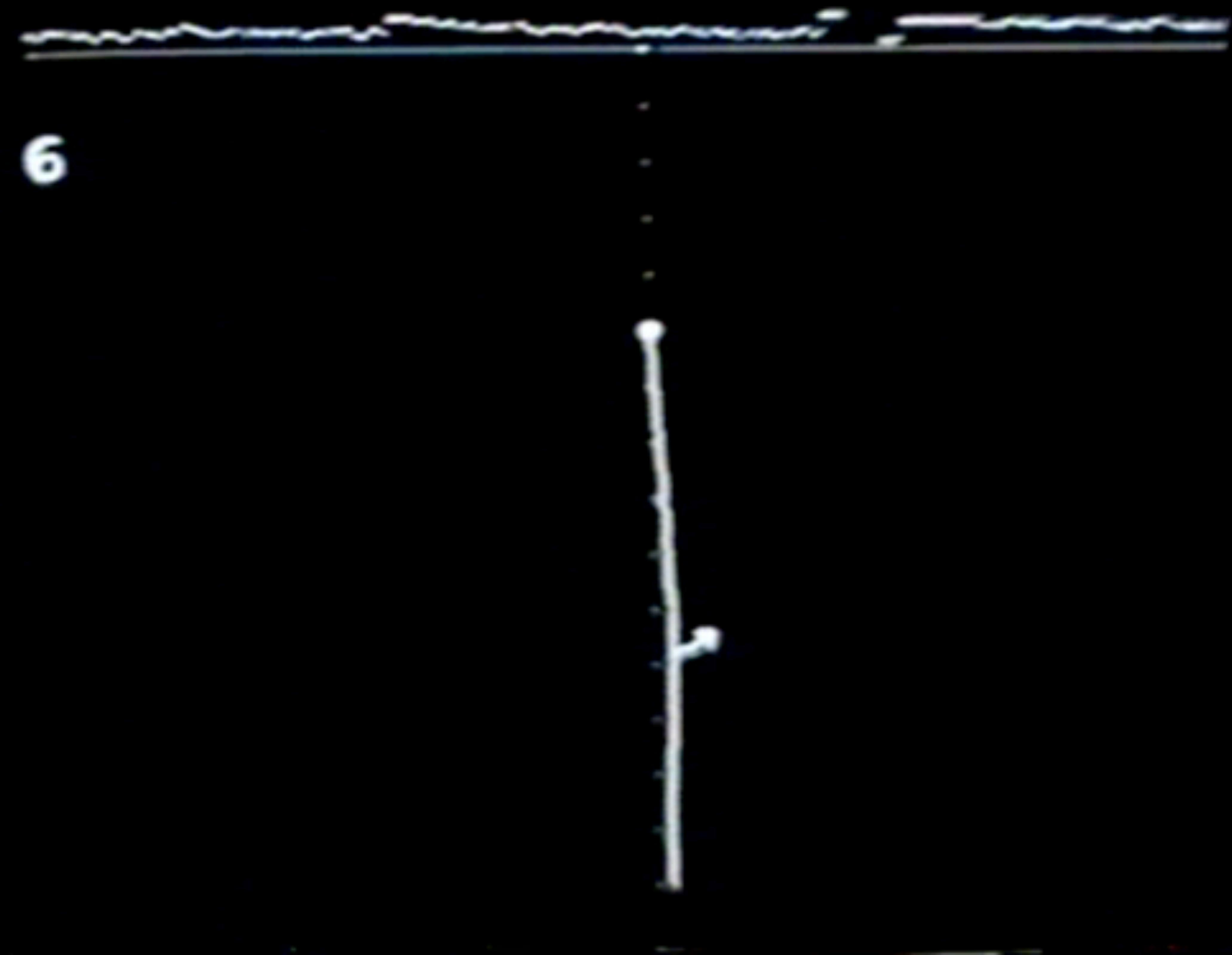
Mountain Car Demo



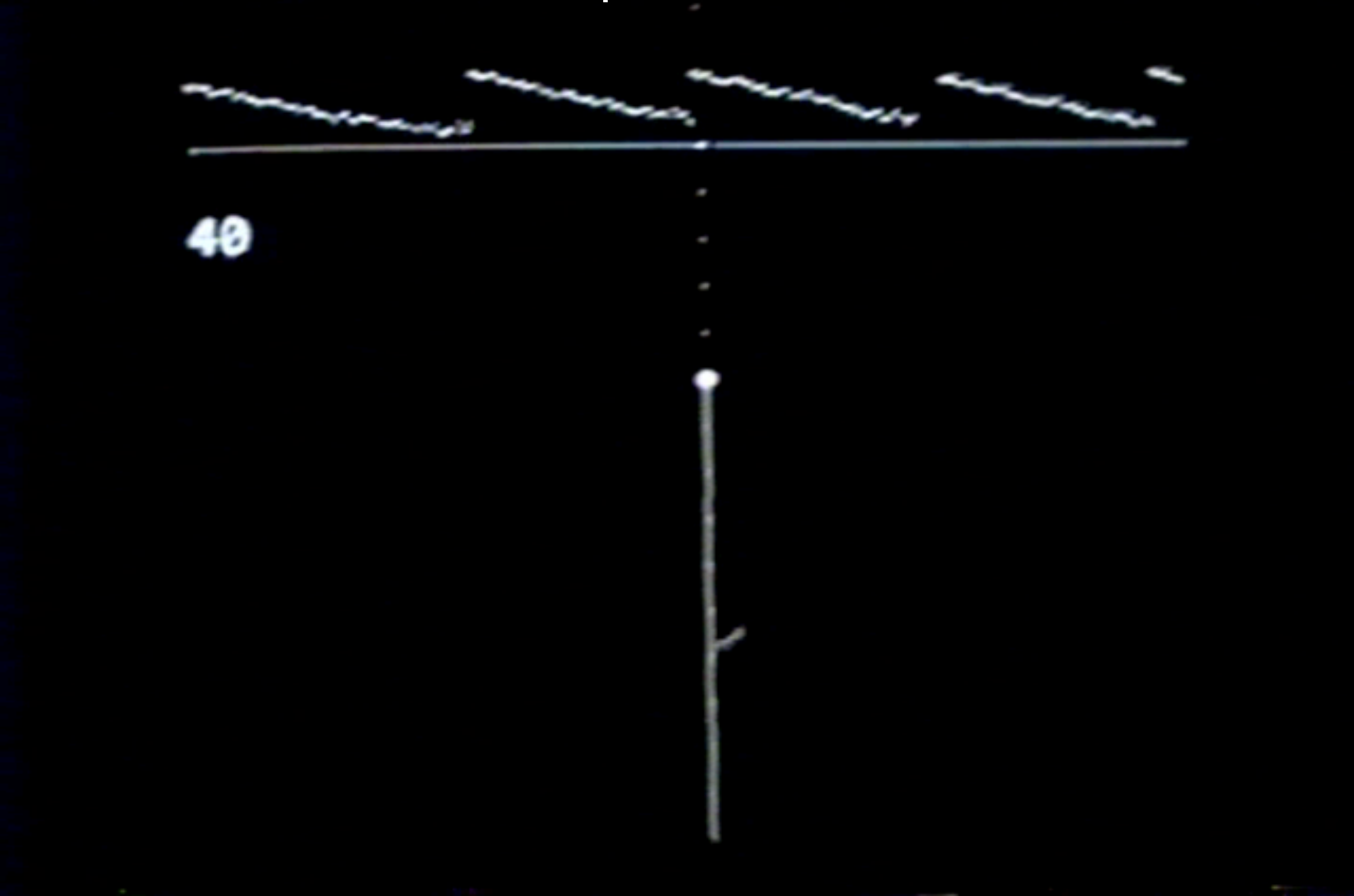
- Linear function approximation
- Coarse-coded features of state (tile coding, CMAC)

Moore 1990, Sutton 1996

Acrobot Demo, Sarsa(λ=0.9)
Episode 6

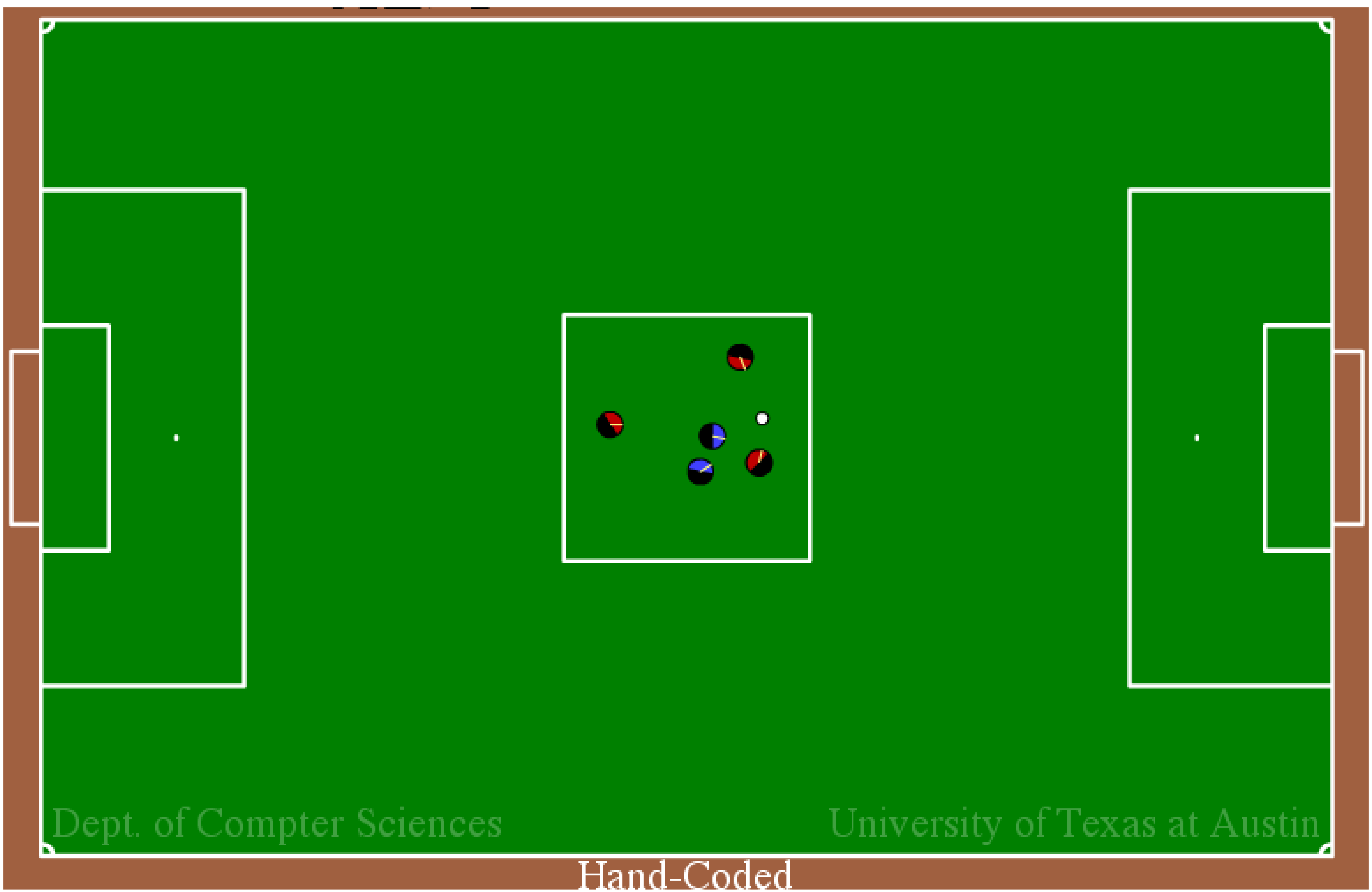


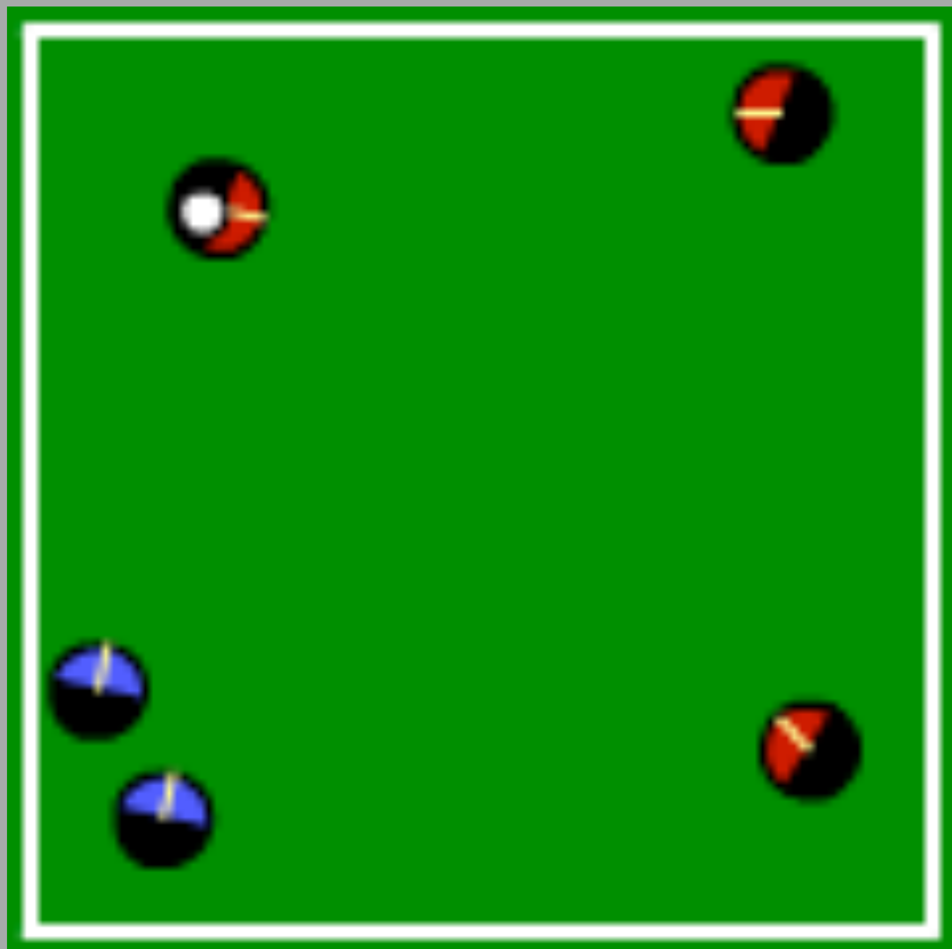
Acrobot Demo, Sarsa(λ=0.9)
Episode 40+



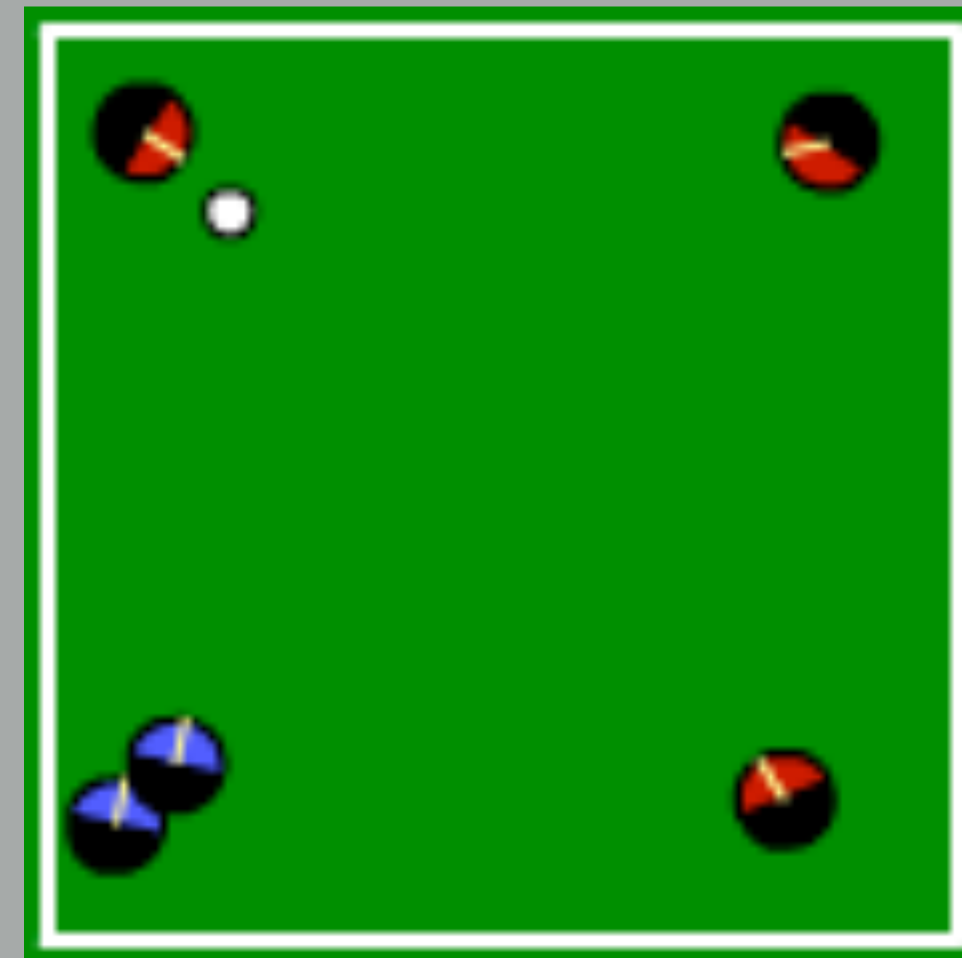
RoboCup soccer keepaway

Stone, Sutton & Kuhlmann, 2005



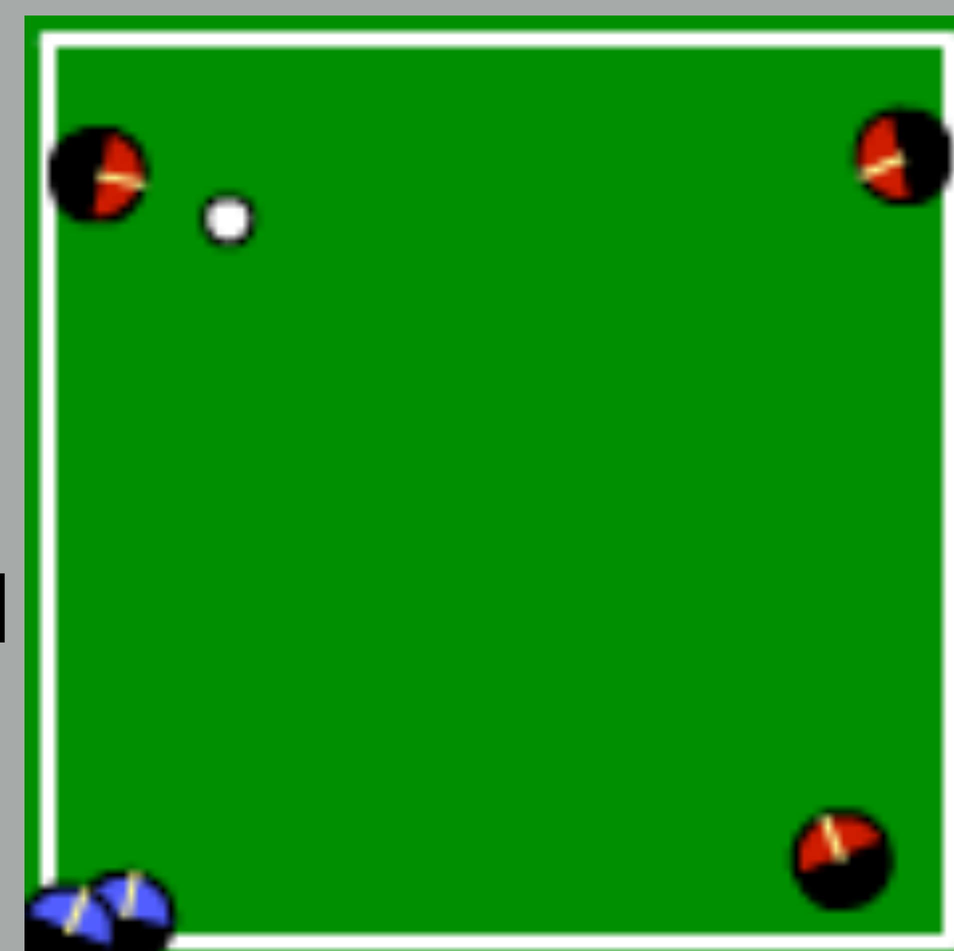


Random



Learned

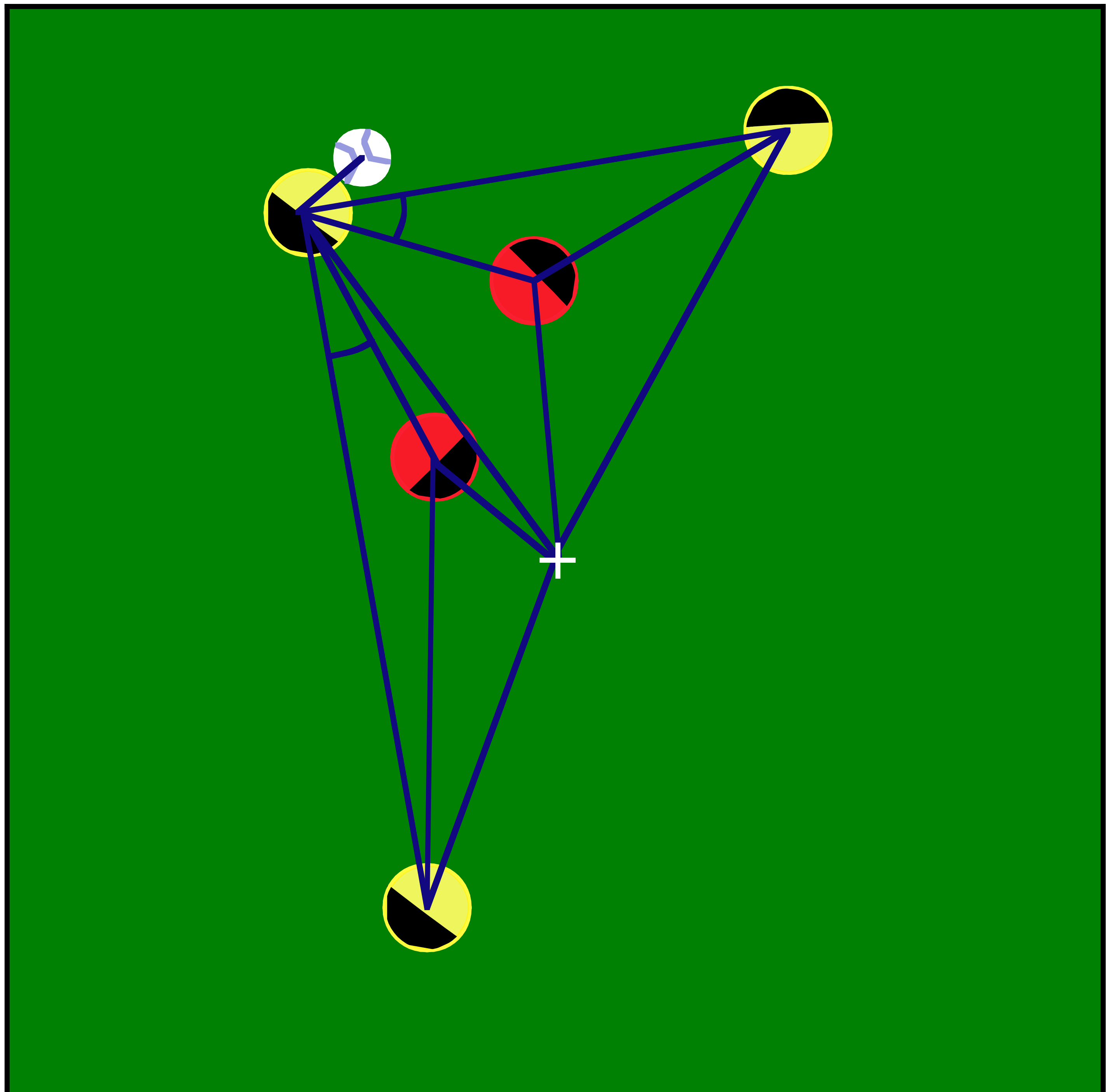
Hand-coded



Hold

Stone, Sutton & Kuhlmann, 2005

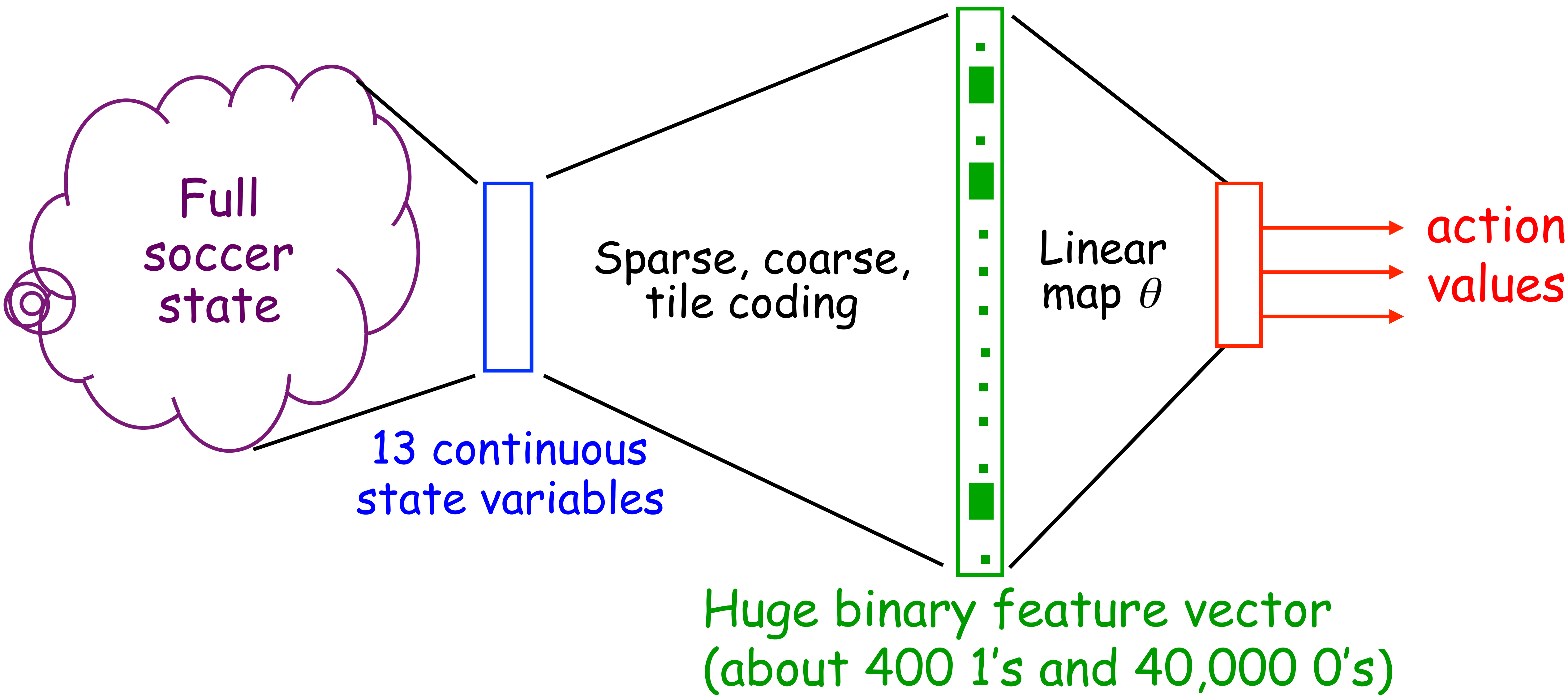
How is the state encoded?
In 13 continuous state variables



11 distances among the players, ball, and the center of the field

2 angles to takers along passing lanes

The Feature-Construction Pipeline



But let's return to the <u>bad news</u>, the problem of instability with semi-gradient Q-learning

What causes the problem of instability?

- It has nothing to do with learning or sampling
  - Even dynamic programming, the classical solution method for known MDPs, suffers from divergence with function approximation
- It has nothing to do with exploration, greedification, or control
  - Even policy evaluation alone can diverge
- It has nothing to do with complex non-linear approximators
  - Even simple linear approximators can produce instability

The deadly triad

- The risk of divergence arises whenever we combine three things:

  1. Function approximation

     significantly generalizing from large numbers of examples

  2. Bootstrapping

     learning value estimates from other value estimates,
     as in dynamic programming and temporal-difference learning

  3. Off-policy learning (Why is dynamic programming off-policy?)

     learning about a policy from data not due to that policy,
     as in Q-learning, where we learn about the greedy policy from
     data with a necessarily more exploratory policy

- Any two without the third is ok

Can we do without bootstrapping?

- Bootstrapping is critical to the computational efficiency of DP
- Bootstrapping is critical to the data efficiency of TD methods
- On the other hand, bootstrapping introduces bias, which harms the asymptotic performance of approximate methods
- The degree of bootstrapping can be finely controlled via the λ parameter, from λ=0 (full bootstrapping) to λ=1 (no bootstrapping)
- For the naive loss:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}\left[\left(q_\pi(S_t, A_t) - \hat{q}(S_t, A_t, \boldsymbol{\theta})\right)^2\right]$$

semi-gradient Sarsa(λ) converges to a fixpoint $\boldsymbol{\theta}_{\text{Sarsa}}$ where

$$\mathcal{L}(\boldsymbol{\theta}_{\text{Sarsa}}) \quad \leq \quad \frac{1-\gamma\lambda}{1-\gamma} \min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$

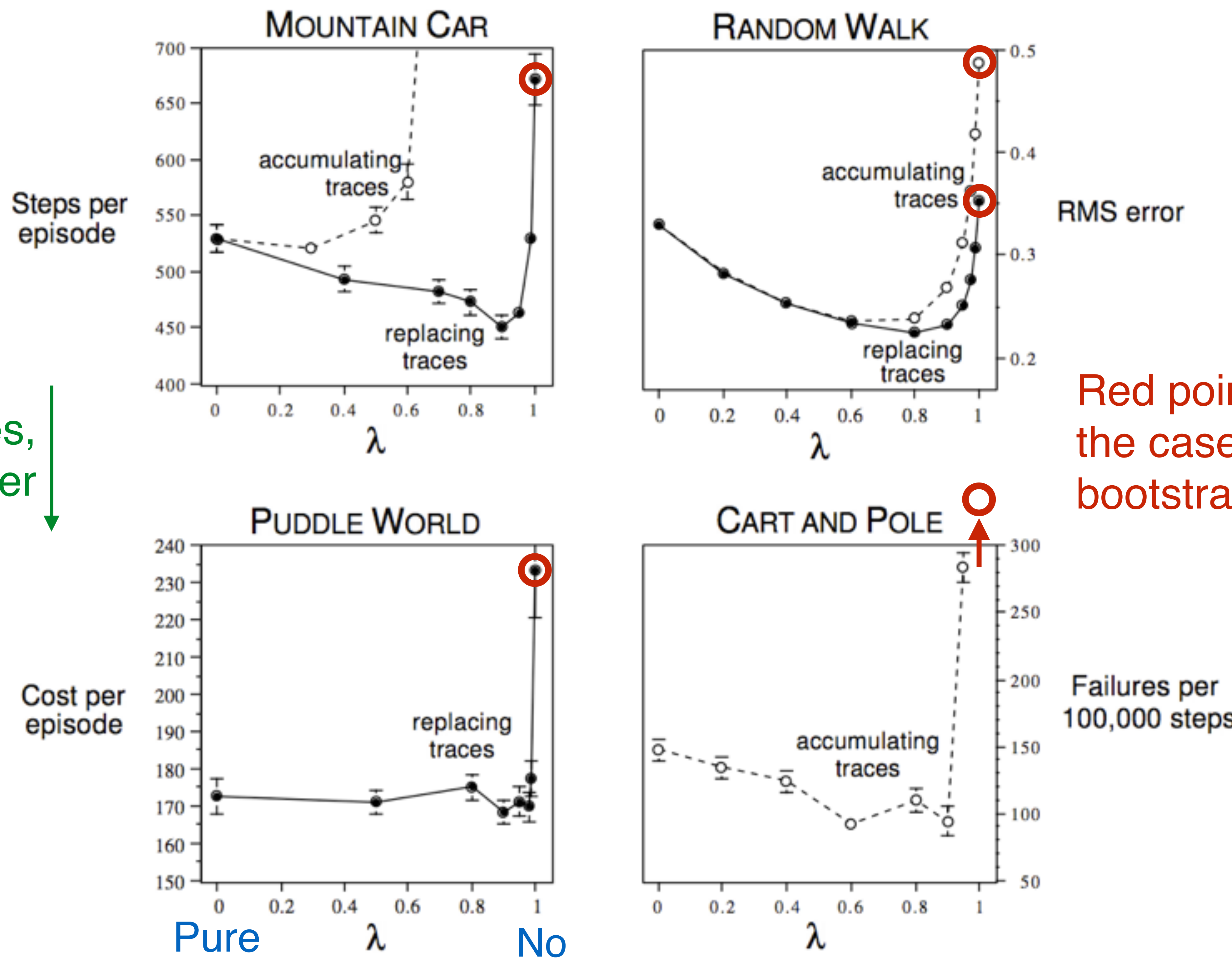
Tsitsiklis & Van Roy 1997

$\Rightarrow$ λ=1 is best!?

4 examples of the effect of bootstrapping

suggest that λ=1 (no bootstrapping) is a very poor choice

In all cases, lower is better



Red points are the cases of <u>no</u> bootstrapping

Pure bootstrapping

No bootstrapping

Other ways to survive the deadly triad

- Use high λ. Use good features
- Recent results suggest that replay and more stable targets (e.g., Double Q-learning, van Hasselt 2010) may help, but it is too soon to be sure
- Use least-squares methods like off-policy LSTD(λ) (Yu 2010, Mahmood et al. 2015). Such methods (Bradtke & Barto 1996, Boyan 2000) easily survive the triad, but their computational costs scale with the *square* of the number of parameters
- Try the new true-gradient RL methods (Gradient-TD and proximal-gradient-TD) developed by Maei (2011) and Mahadevan (2015) et al. These seem to me to be the best attempts to make TD methods with the robust convergence properties of stochastic gradient descent. Residual gradient methods (Baird 1999) are also true gradient methods, but optimize a poor objective, or can't learn purely from data (double sampling). These and other methods based on the Bellman error/residual are not recommended
- Try the even newer Emphatic-TD methods (Sutton, White & Mahmood 2015, Yu 2015). These semi-gradient methods attain stability through an extension of the early on-policy theorems

Outline

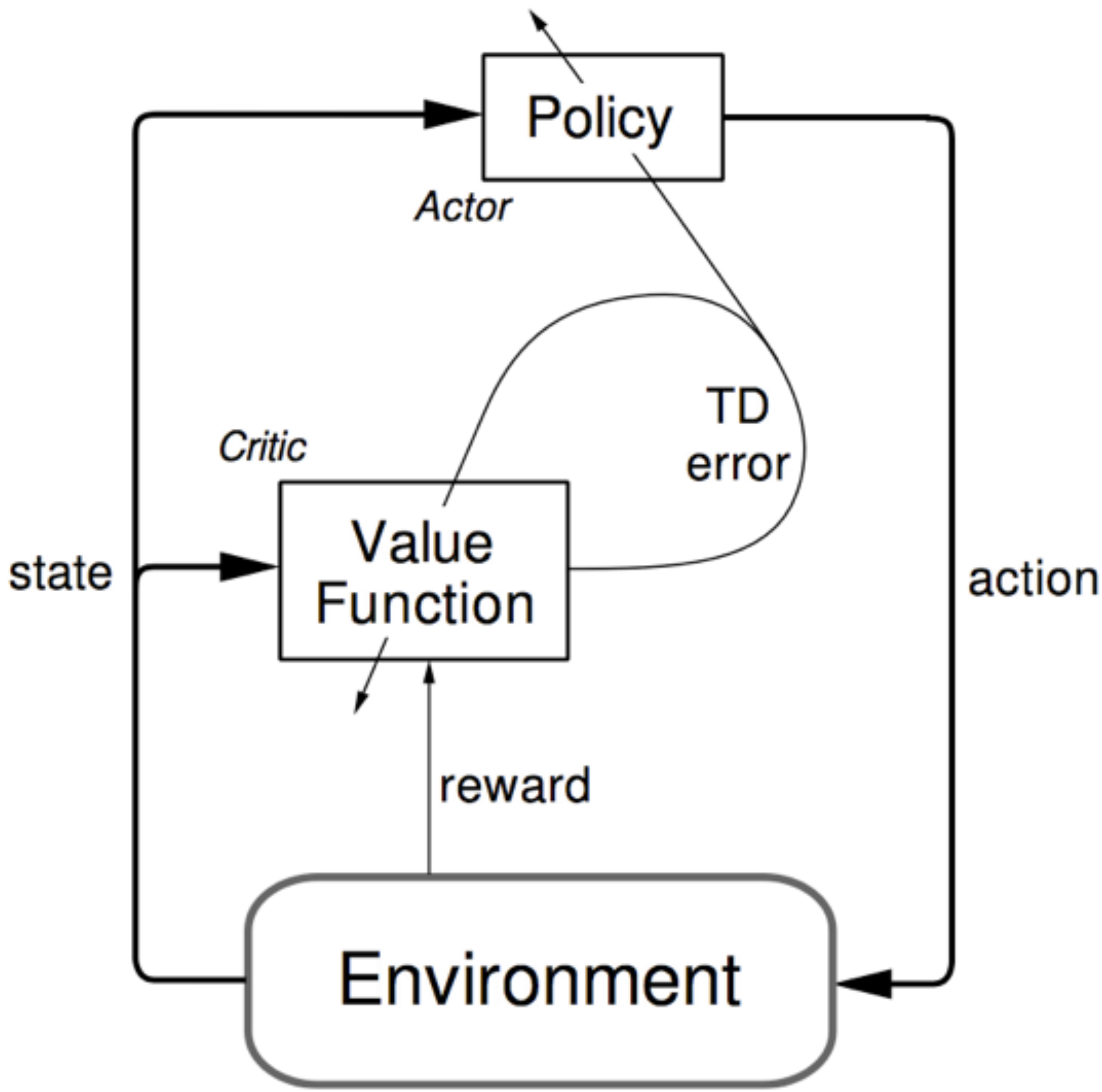
- Introduction to RL successes and challenges
- The formal problem: Finite Markov decision processes
- Part I: Exact solution methods and core theoretical ideas
- Part II: Approximate solution methods
  - Semi-gradient methods
  - On-policy and off-policy methods
  - The deadly triad; how to evade or survive it
- Miscellany and closing remarks

The many dimensions of RL

- Problems
  - prediction vs control
  - MDPs vs Bandits (one state, non-sequential)
- Methods
  - Tabular vs function approximation
  - On-policy vs off-policy
  - Bootstrapping vs Monte Carlo (unified by eligibility traces)
  - Model-based vs model-free
  - Value-based vs policy-based

And yet there is an amazing unity and convergence of methods

Policy-gradient actor-critic methods



- Policy is explicitly represented with its own parameters independent of any value function
- Policy parameters are updated by stochastic gradient ascent in a performance measure such as average reward per step
- A state-value function (critic) is optional but can significantly reduce variance
- Good convergence properties (on-policy)

Why approximate policies rather than values?

- In many problems, the policy is simpler than the value function
- In many problems, the optimal policy is stochastic
  - e.g., bluffing, POMDPs
- To enable smoother change in policies
- To avoid a search on every step (the max)
- To better relate to biology

We should never discount when optimizing approximate policies!

It breaks the definition of an optimal policy

With approximation, *the* optimal policy is no longer representable

There is no way to rank the remaining policies

Different policies will be best in different states

Instead, you must say what states you care about

Or else use average reward
(which you should probably do anyway)

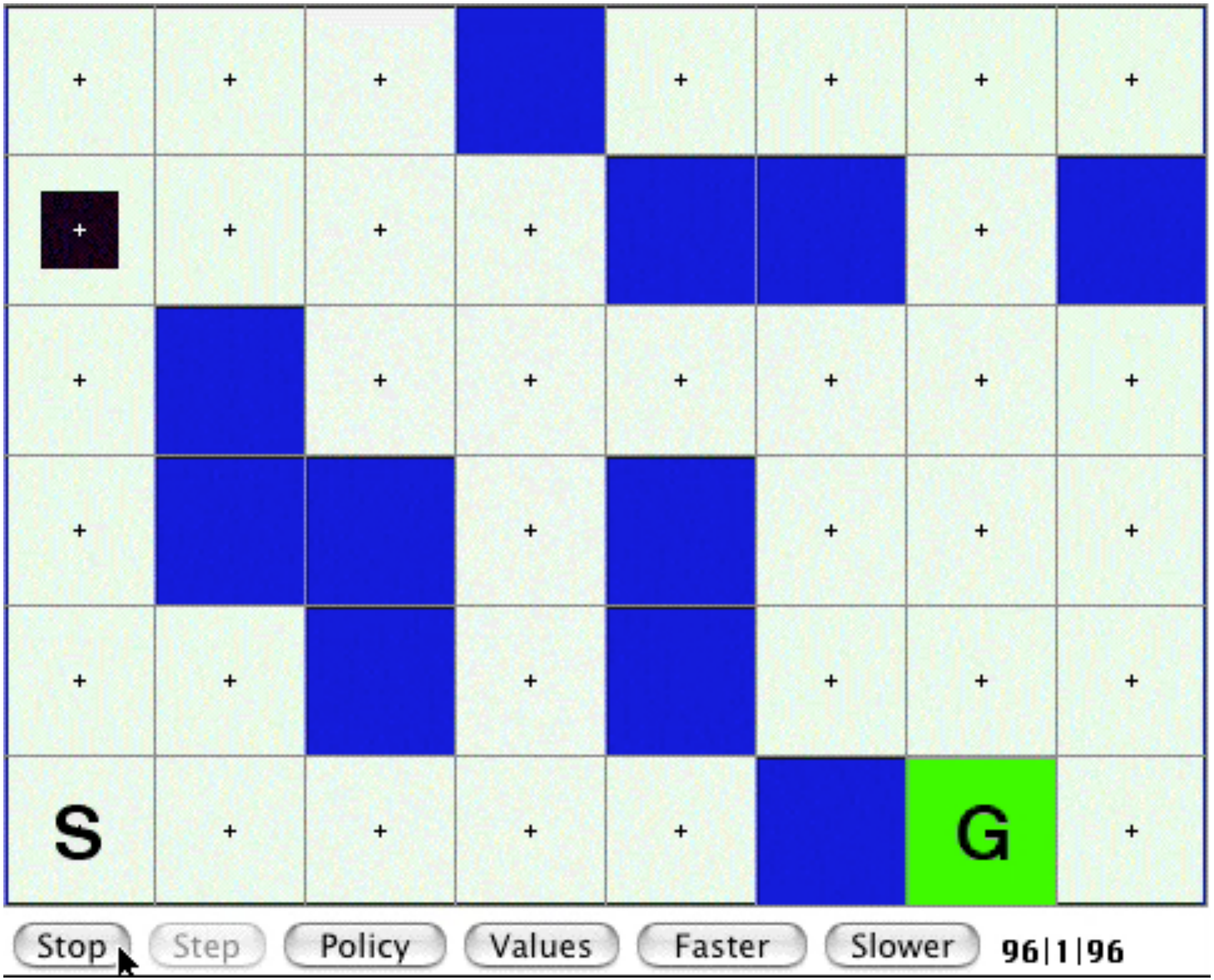
Model-based reinforcement learning

- Learn a model of the environment's transition dynamics

$$\hat{p}(r, s'|s, a) \;\approx\; p(r, s'|s, a)$$

- Use it to generate simulated trajectories
- Apply RL methods to the simulated trajectories, as if they had really happened, to learn an action-value function and policy
- Can be intermixed with direct RL

Model-based RL: GridWorld Example




Andrew Ng, Adam Coates, Pieter Abbeel, et al., Stanford University

Eligibility traces

- An elegant unification of bootstrapping and Monte Carlo (non-bootstrapping) methods
- A key algorithmic innovation that greatly reduces computational complexity in multi-step prediction learning; its most important advantages have nothing to do with bootstrapping or control
- Necessary to extend RL beyond discrete time steps that just happen to be nicely aligned with the world's causal dynamics
- One of the most algorithmically intense topics in RL
  - interacts strongly with off-policy learning via importance sampling, and concomitant struggles to reduce variance

Temporal abstraction in RL

- Function approximation abstracts over state, but we need also to abstract over time

- There are several approaches, including the framework of options—macro-actions of extended and variable duration that can nevertheless interoperate with primitive actions in DP planning methods and TD learning methods

- The problems of temporal abstraction can be divided into three classes, increasing in difficulty:
  - representing temporal abstractions (e.g., by options)
  - learning temporal abstractions (e.g., by off-policy methods)
  - discovering temporal abstractions and selecting among them

Conclusion

- Reinforcement learning is a big topic, with a long history, an elegant theoretical core, novel algorithms, many open problems, and vast unexplored territories
- RL can be viewed as a microcosm of the whole AI problem, including planning, acting, learning, perception, world modeling, even knowledge representation
- Yet, even so, it can be reduced to small steps on each of which measurable progress can be made
- RL fits well into the longest mega-trend in AI, that towards turning more of the work over to the machine
- Realistic, ambitious, pragmatic

Thank you for your attention