

Computational Reinforcement Learning: An Introduction

Andrew Barto
Autonomous Learning Laboratory
School of Computer Science
University of Massachusetts Amherst
barto@cs.umass.edu

Artificial Intelligence

Psychology

**Control Theory and
Operations Research**

**Computational
Reinforcement
Learning (RL)**

Neuroscience

The Plan

Part I

- What is computational reinforcement learning?
 - Ancient history
 - RL and supervised learning
- Agent-Environment interaction
 - Markov Decision Processes (MDPs)
 - Some standard simple examples
- Understanding the degree of abstraction
- Value functions
- Bellman Equations
- Semi-Markov Decision Processes (SMDPs)
- Partially Observable MDPs (POMDPs)
- Major challenges for RL

The Overall Plan

Part 2

- Understanding Temporal Difference prediction
- Temporal difference control
 - Sarsa
 - Q-Learning
 - Actor-Critic architecture
- Function approximation
- Direct policy search
- Hierarchical RL
- Intrinsically motivated RL
- Summing up

Suggested Readings

Reinforcement Learning: An Introduction

*R. S. Sutton and A. G. Barto, MIT Press, 1998
Chapters 1, 3, 6 ...*

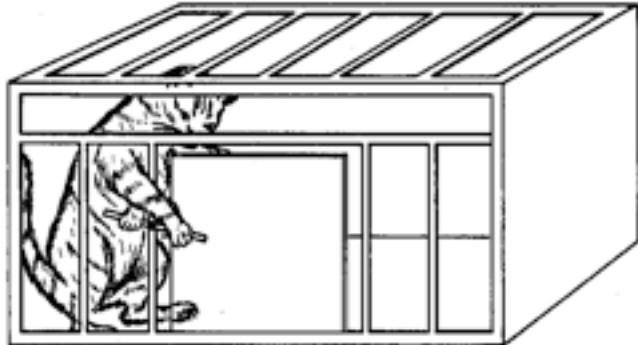
Temporal Difference Learning

*A. G. Barto, Scholarpedia, 2(11):1604,
2007*

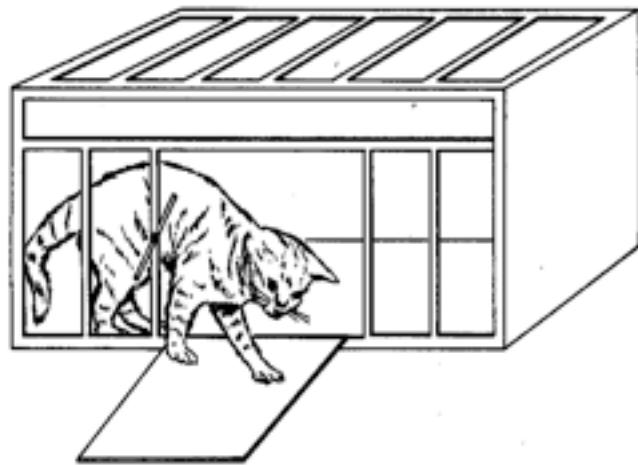
Reinforcement Learning in Robotics: A Survey

*J. Kober, J. Andrew (Drew) Bagnell, and J. Peters
International Journal of Robotics Research, July, 2013.*

Edward L. Thorndike (1874 – 1949)



(a)



puzzle box



Learning by “Trial-and-Error”
Instrumental Conditioning

Law-of-Effect

“Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that when it recurs, they will be less likely to occur.”

Thorndike, 1911

Trial-and-Error \neq Error-Correction

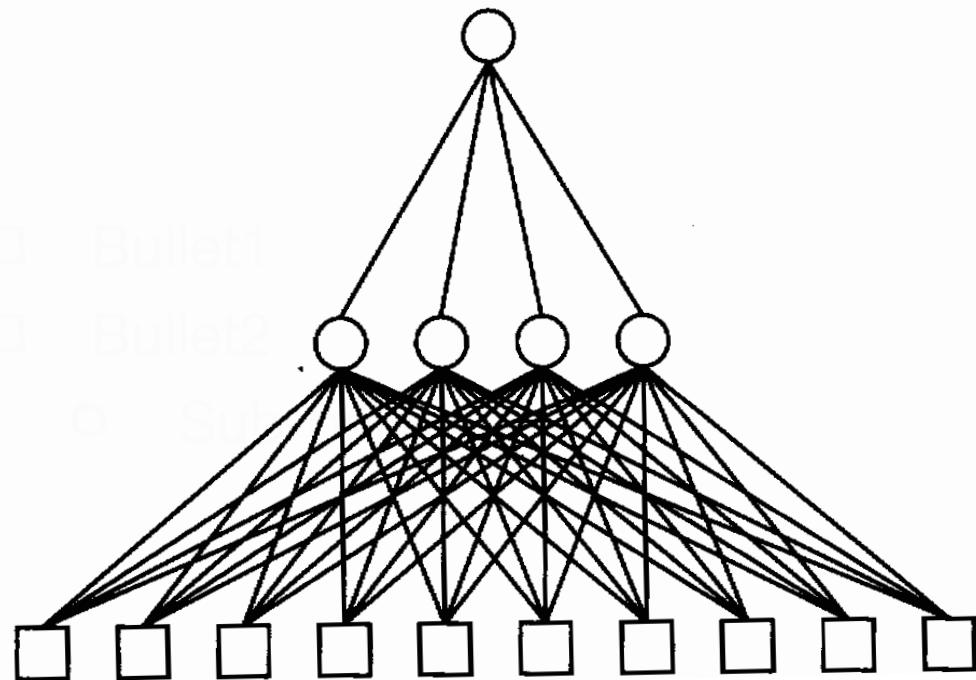
Output units O_i

$W_{j,i}$

Hidden units a_j

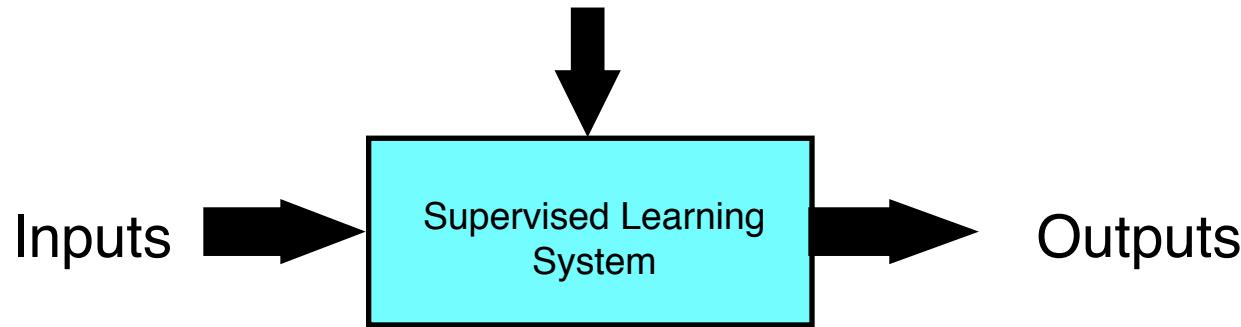
$W_{k,j}$

Input units I_k



Supervised Learning

Training Info = desired (target) outputs



Error = (target output – actual output)

Regression

Classification

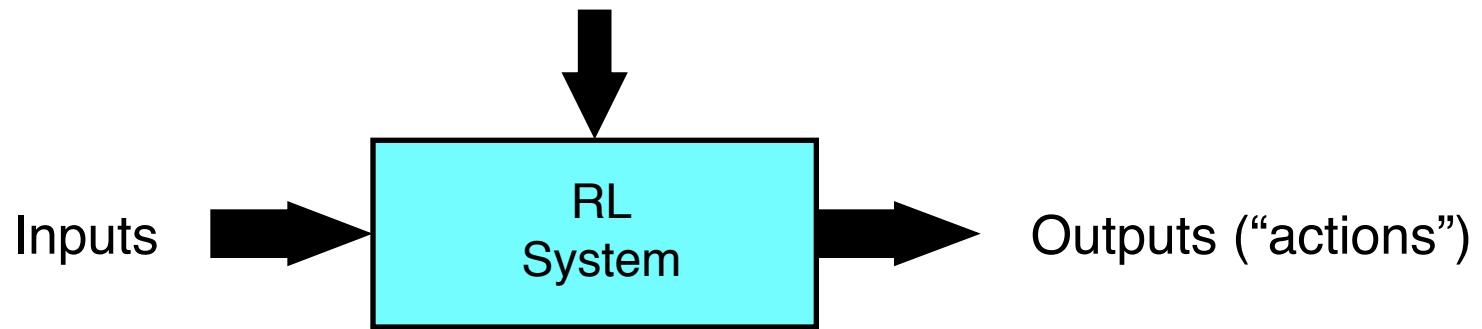
Imitation Learning

Prediction

System Identification (model learning)

Reinforcement Learning

Training Info = evaluations (“rewards” / “penalties”)



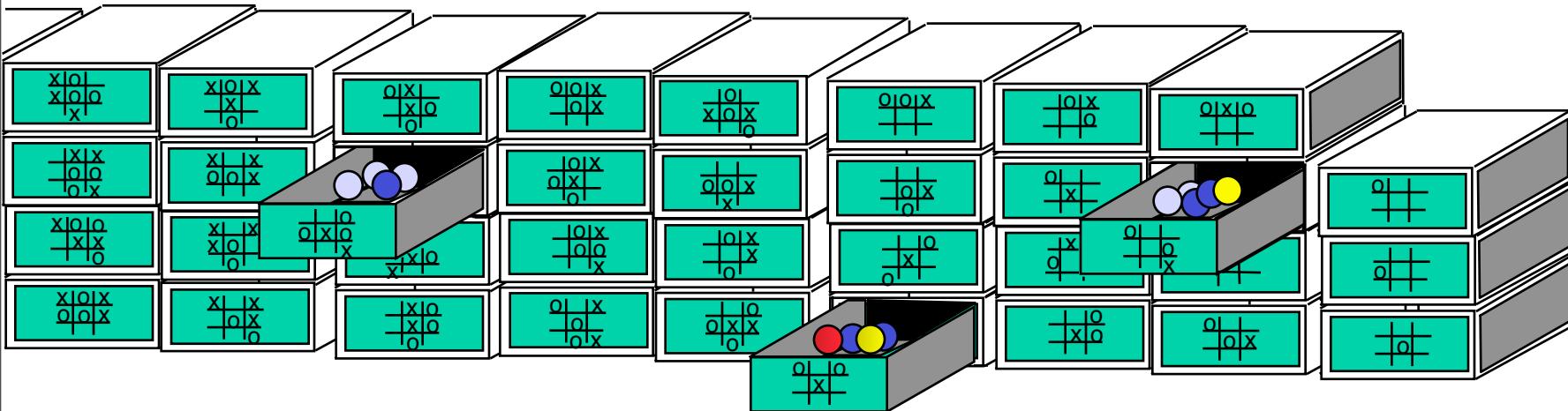
Objective: get as much reward as possible

RL = Control + Search + Memory

- ❑ **Control**: influence future inputs
- ❑ **Search**: Trial-and-Error, Generate-and-Test, Variation-and-Selection
- ❑ **Memory**: remember what worked best for each situation and start from there next time

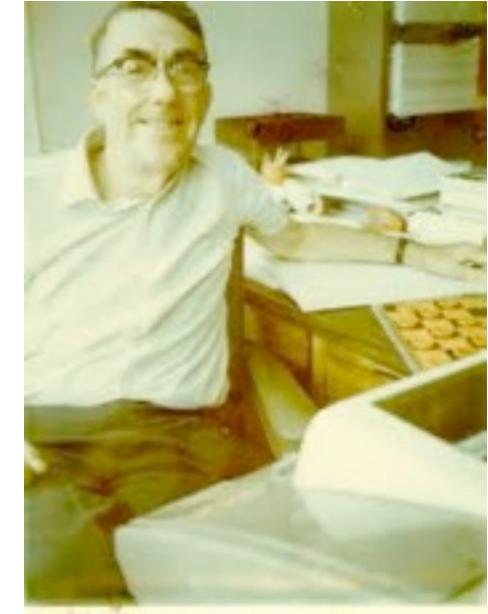
MENACE (Michie 1961)

“Matchbox Educable Noughts and Crosses Engine”



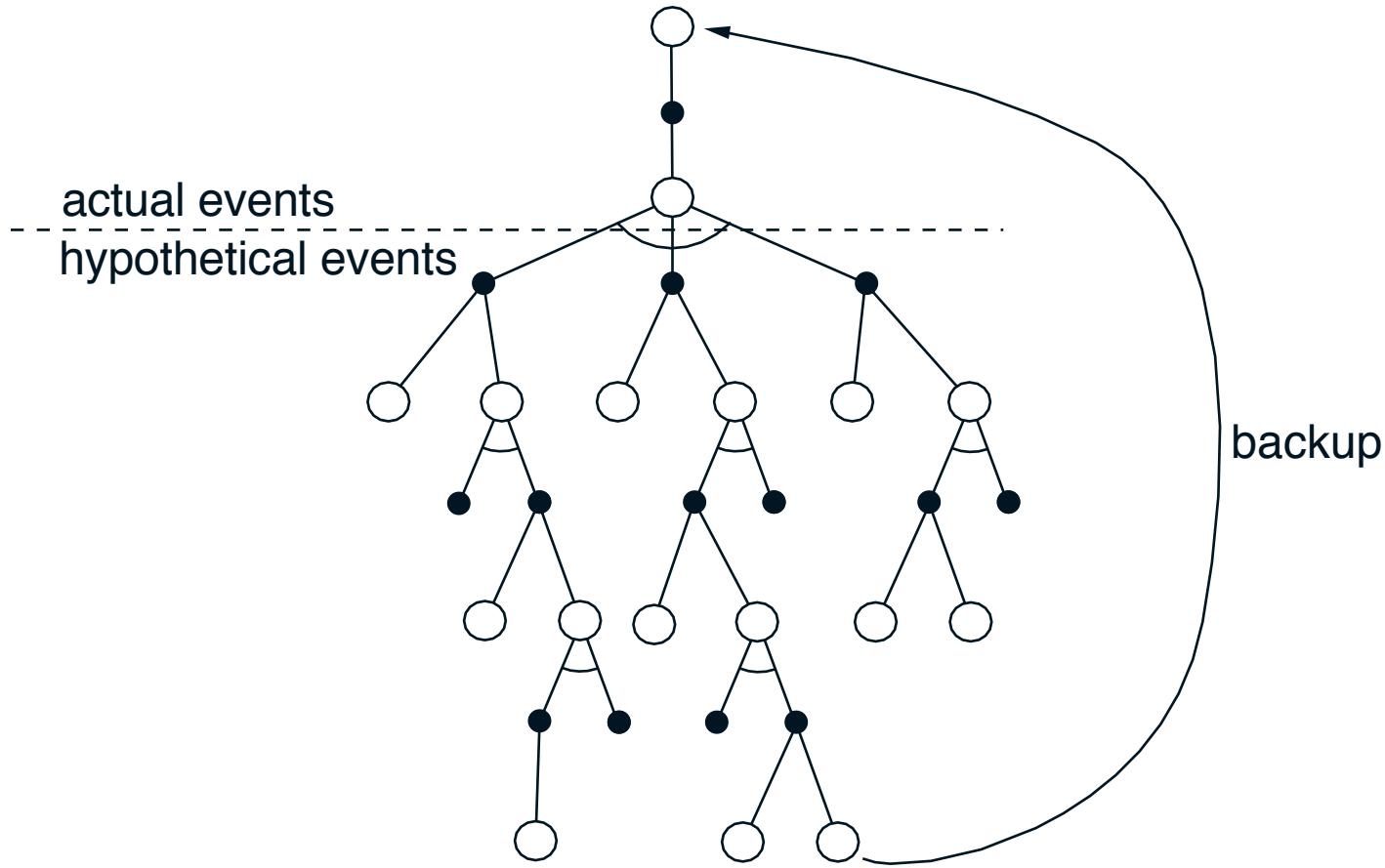
Samuel's Checkers Player

Arthur Samuel 1959, 1967



- Score board configurations by a “**scoring polynomial**” (after Shannon, 1950)
- **Minimax** to determine “backed-up score” of a position
- **Alpha-beta** cutoffs
- **Rote learning**: save each board config encountered together with backed-up score
 - needed a “sense of direction”: like discounting
- **Learning by generalization**: similar to TD algorithm

Samuel's Backups



Samuel's Basic Idea

“. . . we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board positions of the chain of moves which most probably occur during actual play.”

A. L. Samuel

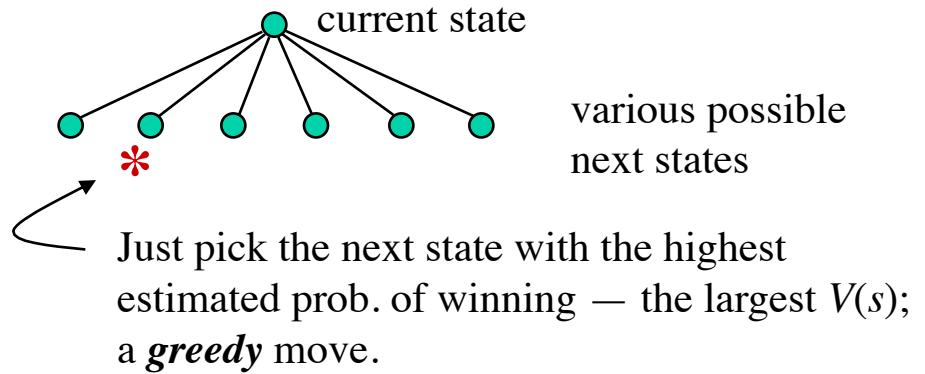
*Some Studies in Machine Learning
Using the Game of Checkers, 1959*

Another Tic-Tac-Toe Learner

1. Make a table with one entry per state:

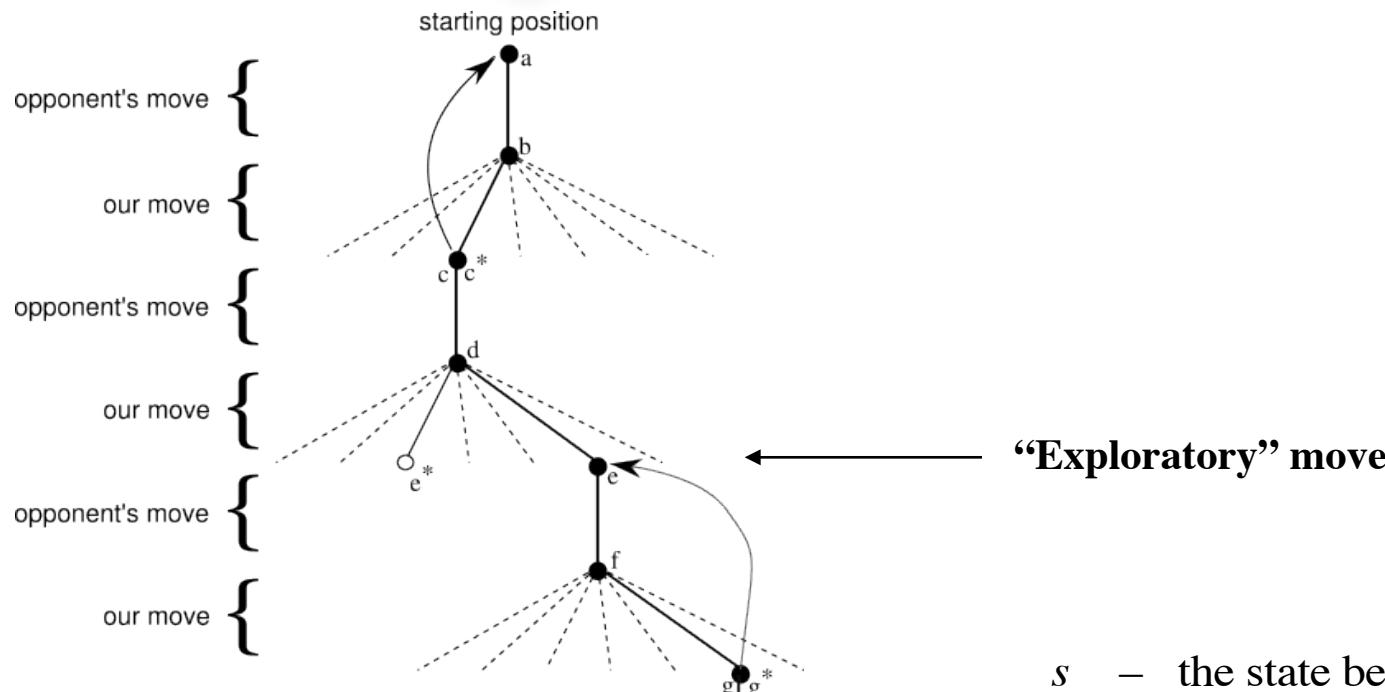
State	$V(s)$ – estimated probability of winning	
#	.5	?
#	.5	?
.	:	
.	:	
x x x o o o o o	1	win
.	:	
.	:	
x x o x o o o	0	loss
.	:	
.	:	
o x o o x x x o o	0	draw

2. Now play lots of games.
To pick our moves,
look ahead one step:



But 10% of the time pick a move at random;
an *exploratory move*.

RL Learning Rule for Tic-Tac-Toe



s – the state before our greedy move
 s' – the state after our greedy move

We increment each $V(s)$ toward $V(s')$ – a **backup**:

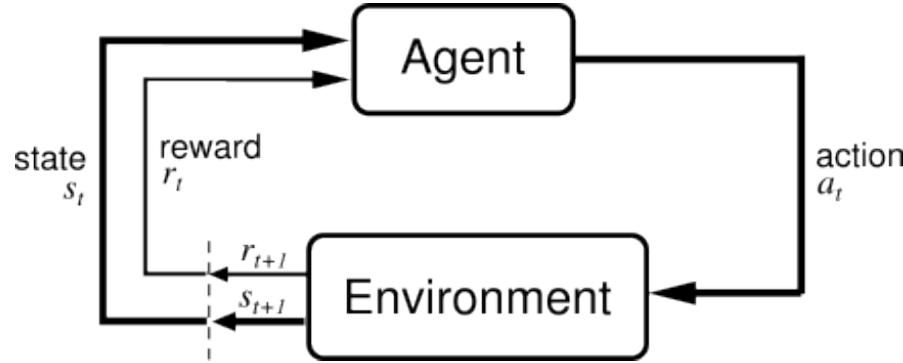
$$V(s) \leftarrow V(s) + \alpha [V(s') - V(s)]$$

a small positive fraction, e.g., $\alpha = .1$
the **step-size parameter**

But Tic-Tac-Toe is WAY too easy ...

- Finite, small number of states
- Finite, small number of actions
- One-step look-ahead is always possible
- State completely observable
- Easily specified goal
- . . .

Agent-Environment Interaction



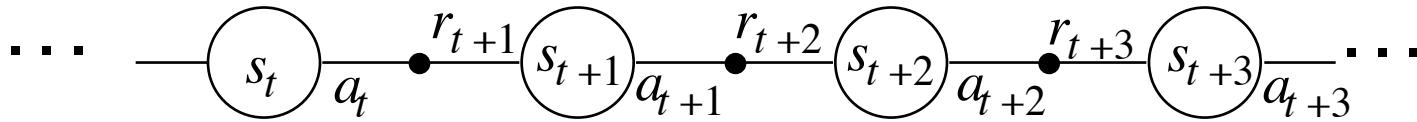
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \dots$

Agent observes state at step t : $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}



Agent Learns a Policy

Policy at step t , π_t :

a mapping from states to action probabilities

$\pi_t(s, a) =$ probability that $a_t = a$ when $s_t = s$

- ❑ Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- ❑ Roughly, the agent's goal is to get as much reward as it can over the long run.

The Markov Property

- By “the state” at step t , we mean whatever information is available to the agent at step t about its environment.
- The state can include immediate “sensations,” highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all “essential” information, i.e., it should have the **Markov Property**:

$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\right\} = \\ \Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\right\}$$

for all s' , r , and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0$.

Markov Decision Processes (MDPs)

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
 - **state and action sets**
 - one-step “dynamics” defined by **transition probabilities**:
$$P_{ss'}^a = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad \text{for all } s, s' \in S, a \in A(s).$$
 - **reward expectations**:
$$R_{ss'}^a = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \quad \text{for all } s, s' \in S, a \in A(s).$$

Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \dots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t .

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \dots + r_T,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma, 0 \leq \gamma \leq 1$, is the **discount rate**.

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

Average reward:

$$\rho^\pi(s) = \lim_{N \rightarrow \infty} \frac{1}{N} E \left[\sum_{t=1}^N r_t \right]$$

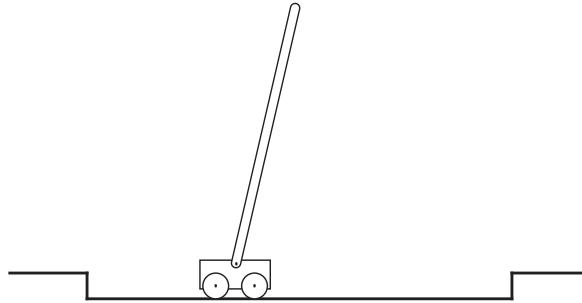
Rewards

What Sutton & Barto wrote in 1998:

“... the reward signal is not the place to impart to the agent prior knowledge about *how to achieve what we want it to do*. For example, a chess-playing agent should be rewarded only for actually winning, not for achieving subgoals such taking its opponent's pieces or gaining control of the center of the board. If achieving these sorts of subgoals were rewarded, then the agent might find a way to achieve them without achieving the real goal. For example, it might find a way to take the opponent's pieces even at the cost of losing the game. The reward signal is your way of communicating to the robot *what you want it to achieve, not how you want it achieved*.”

Not Necessarily...

An Example



Avoid **failure**: the pole falling beyond a critical angle or the cart hitting end of track.

As an **episodic task** where episode ends upon failure:

reward = +1 for each step before failure

⇒ return = number of steps before failure

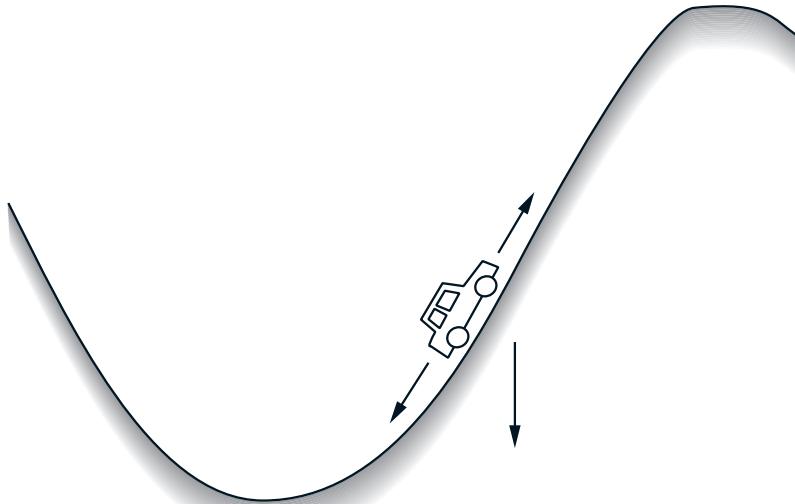
As a **continuing task** with discounted return:

reward = -1 upon failure; 0 otherwise

⇒ return is related to $- \gamma^k$, for k steps before failure

In either case, return is maximized by avoiding failure for as long as possible.

Another Example



Get to the top of the hill
as quickly as possible.

reward = -1 for each step where **not** at top of hill

\Rightarrow return = - number of steps before reaching top of hill

Return is maximized by minimizing
number of steps reach the top of the hill.

Models

- ❑ **Model**: anything the agent can use to predict how the environment will respond to its actions
- ❑ **Distribution model**: description of all possibilities and their probabilities
 - e.g., $P_{ss'}^a$ and $R_{ss'}^a$ for all s, s' , and $a \in A(s)$
- ❑ **Sample model**: produces sample experiences
 - e.g., a simulation model
- ❑ Both types of models can be used to produce **simulated experience**
- ❑ Often sample models are much easier to come by

“Credit Assignment Problem”

Marvin Minsky, 1961

Getting useful training information to the right places at the right times

Spatial

Temporal

“Credit Assignment Problem”

Marvin Minsky, 1961

Getting useful training information to the right places at the right times

Spatial



Temporal

Exploration/Exploitation Dilemma

n -Armed Bandit

- ❑ Choose repeatedly from one of n actions; each choice is called a **play**
- ❑ After each play a_t , you get a reward r_t , where

$$E\langle r_t | a_t \rangle = Q^*(a_t)$$

These are unknown **action values**

Distribution of r_t depends only on a_t

- ❑ Objective is to maximize the reward in the long term, e.g., over 1000 plays

To solve the n -armed bandit problem,
you must **explore** a variety of actions
and the **exploit** the best of them

Exploration/Exploitation Dilemma

- Suppose you form estimates

$$Q_t(a) \approx Q^*(a) \quad \text{action value estimates}$$

- The **greedy** action at t is

$$a_t^* = \arg \max_a Q_t(a)$$

$$a_t = a_t^* \Rightarrow \text{exploitation}$$

$$a_t \neq a_t^* \Rightarrow \text{exploration}$$

- You can't exploit all the time; you can't explore all the time
- You can never stop exploring; but you should always reduce exploring

Getting the Degree of Abstraction Right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), “mental” (e.g., shift in focus of attention), etc.
- States can be low-level “sensations”, or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being “surprised” or “lost”).
- An RL agent is not like a whole animal or robot, which consist of many RL agents as well as other components.
- The environment is not necessarily unknown to the agent, only incompletely controllable.
- Reward computation is in the agent’s environment because the agent cannot change it arbitrarily.

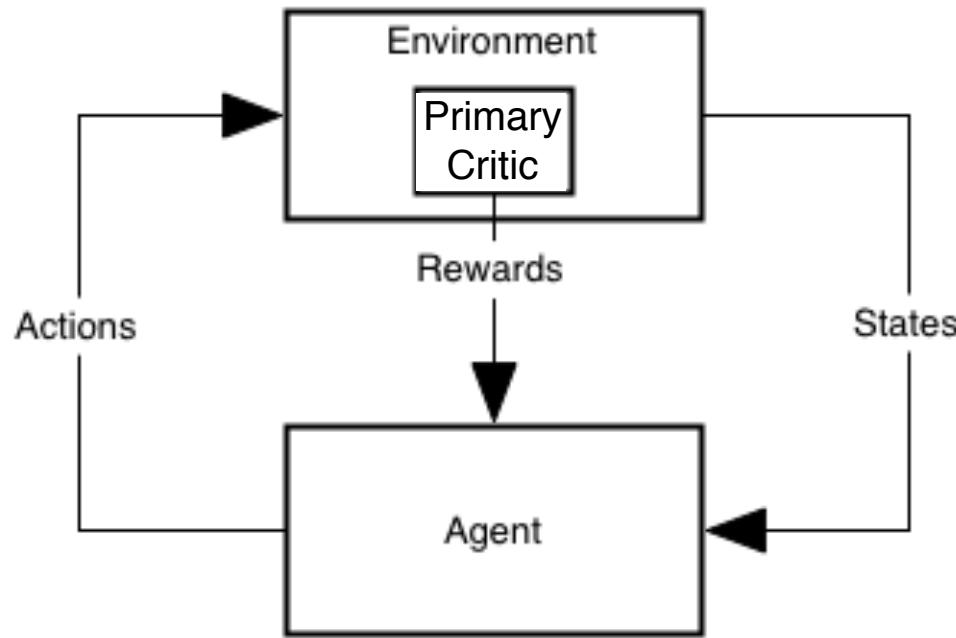
Rewards vs. Reward Signals

Wolfram Schultz (2007), Scholarpedia, 2(6):2184
and 2(3): 1652

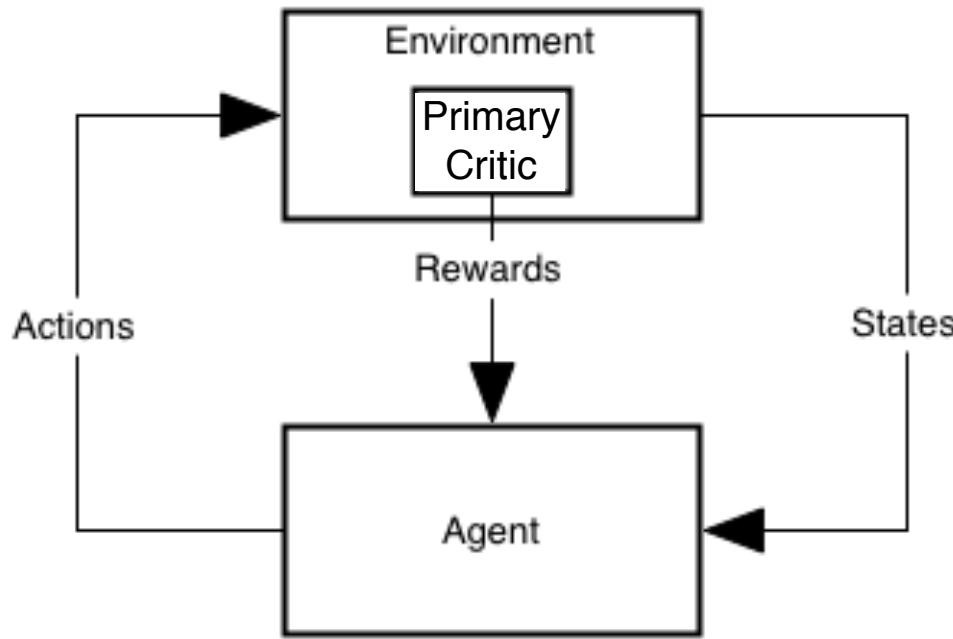
- “**Rewards** are objects or events that make us come back for more. We need them for survival and use them for behavioral choices that maximize them.”
- “Reward neurons produce **reward signals** and use them for influencing brain activity that controls our actions, decisions and choices.”

Agent-Environment Interaction

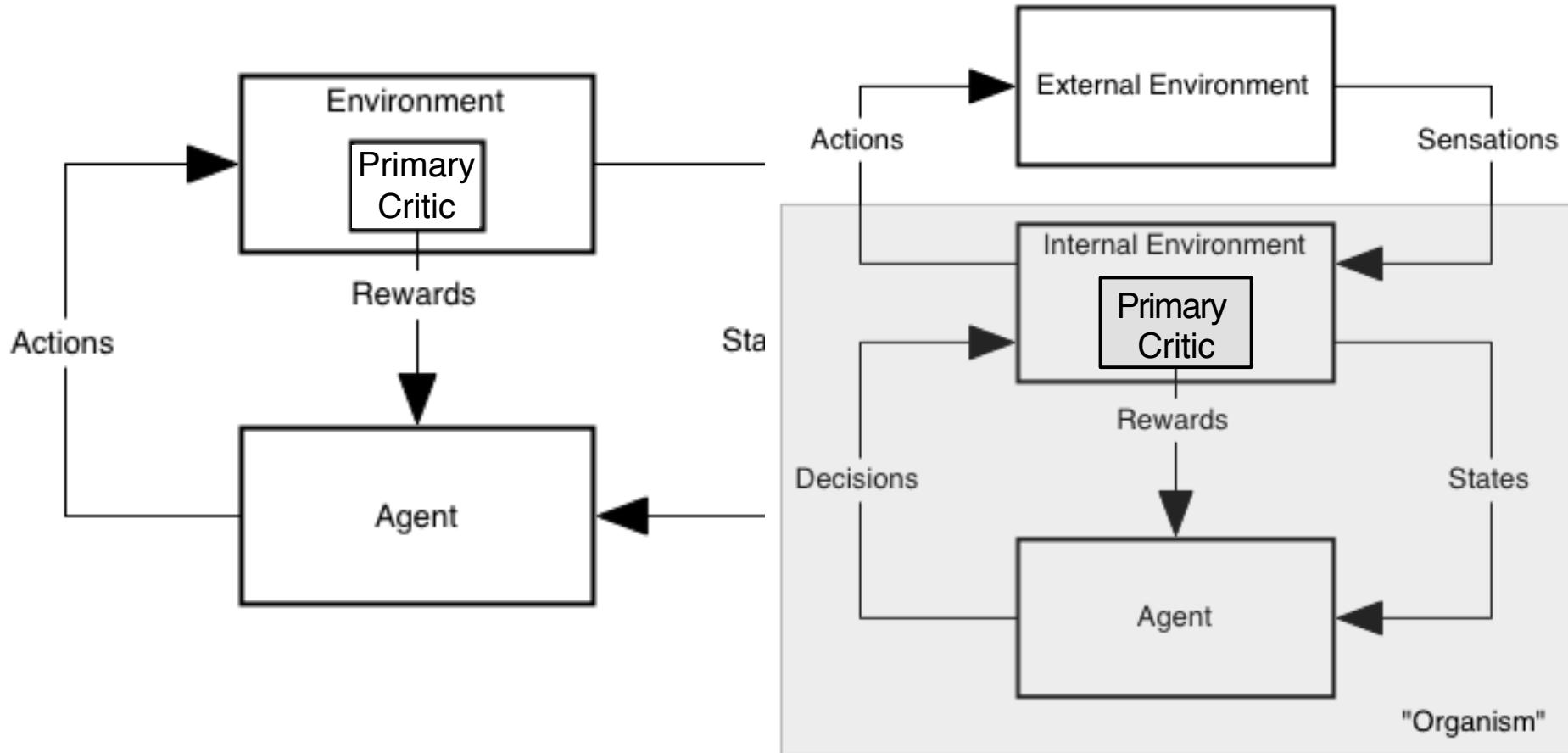
Agent-Environment Interaction



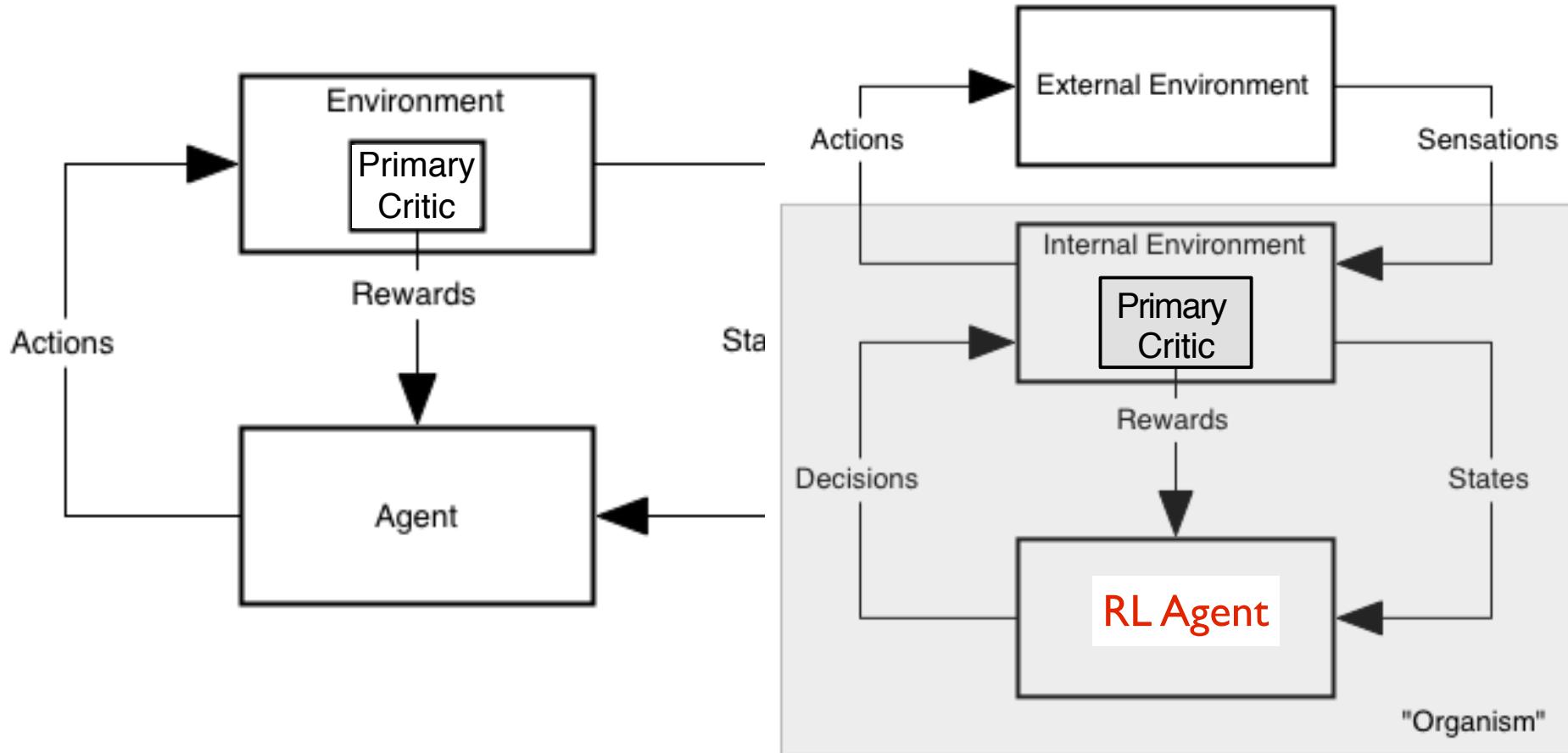
Agent-Environment Interaction



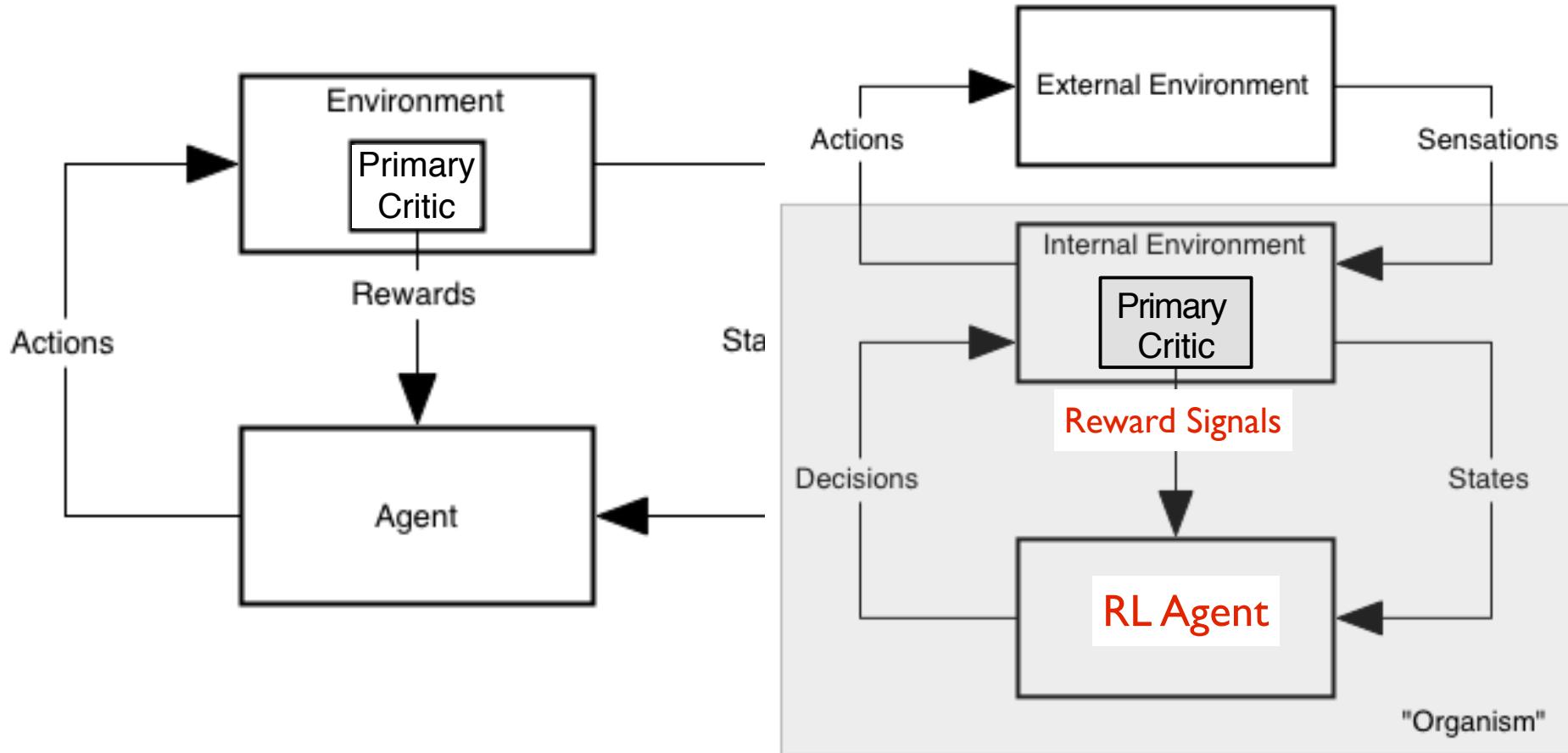
Agent-Environment Interaction



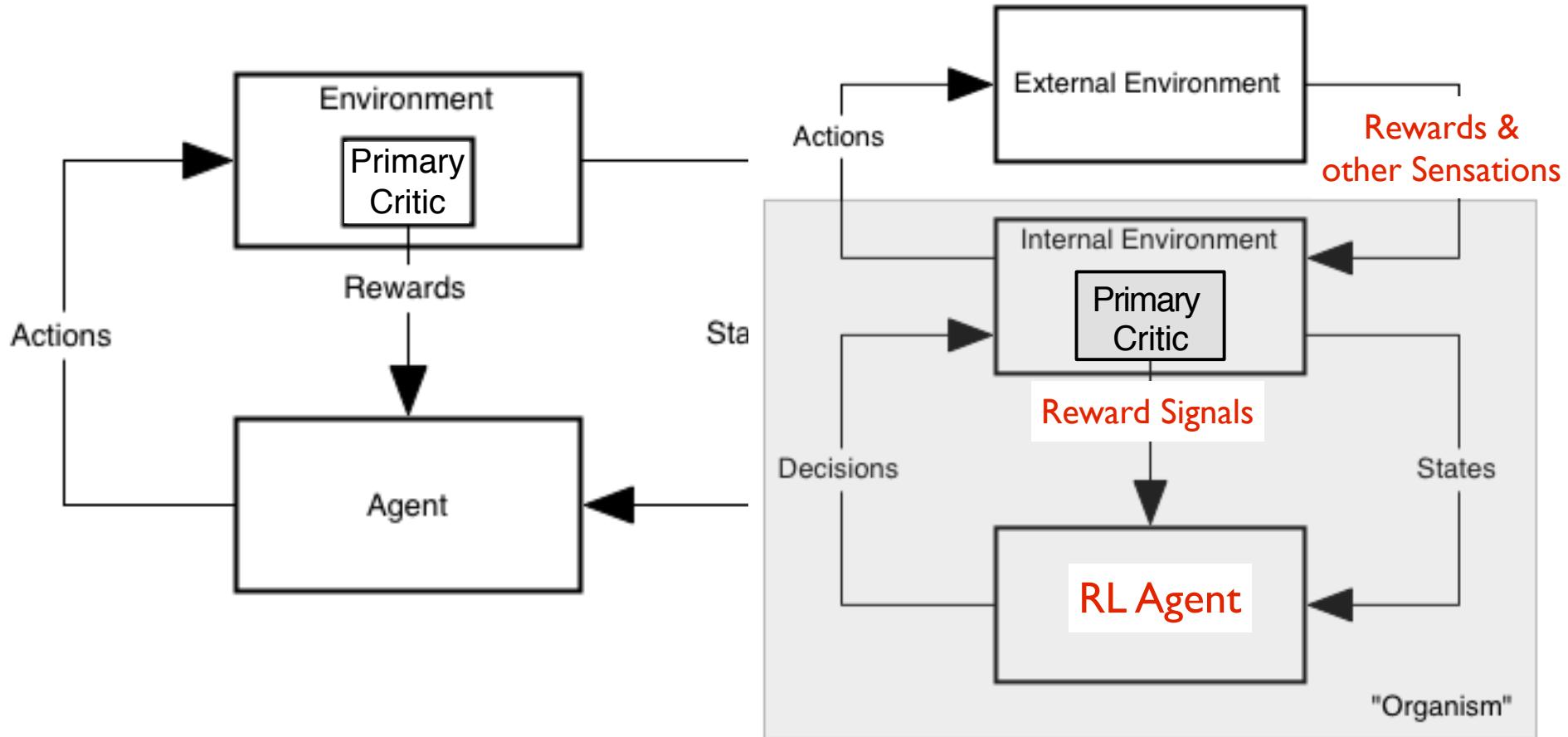
Agent-Environment Interaction



Agent-Environment Interaction



Agent-Environment Interaction



Important RL Distinctions

- Model-free methods
- Model-based methods

- Value function methods
- Direct policy search

Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State - value function for policy π :

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

- The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π :

Action- value function for policy π :

$$Q^\pi(s, a) = E_\pi \left\{ R_t \mid s_t = s, a_t = a \right\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\}$$

Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:
 $\pi \geq \pi'$ if and only if $V^\pi(s) \geq V^{\pi'}(s)$ for all $s \in S$
- There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an **optimal policy**. We denote them all π^* .
- Optimal policies share the same **optimal state-value function**:
$$V^*(s) = \max_{\pi} V^\pi(s) \quad \text{for all } s \in S$$
- Optimal policies also share the same **optimal action-value function**:

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \quad \text{for all } s \in S \text{ and } a \in A(s)$$

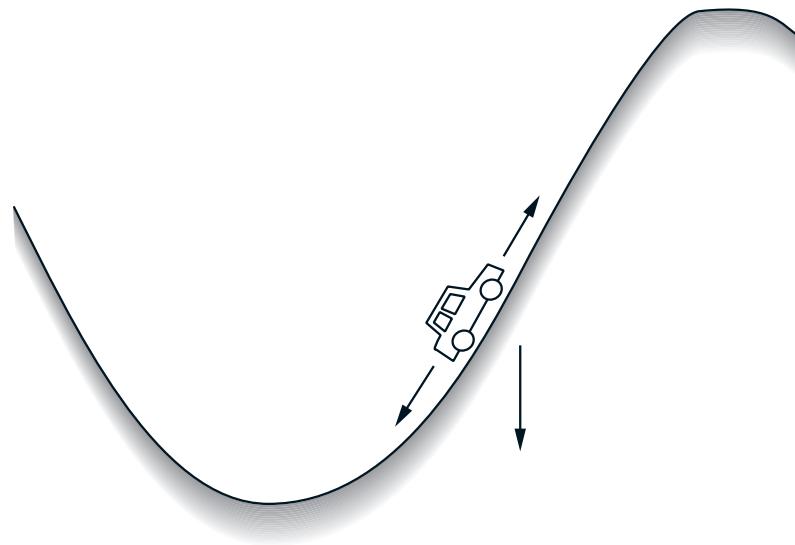
This is the expected return for taking action a in state s and thereafter following an optimal policy.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to V^* is an optimal policy.

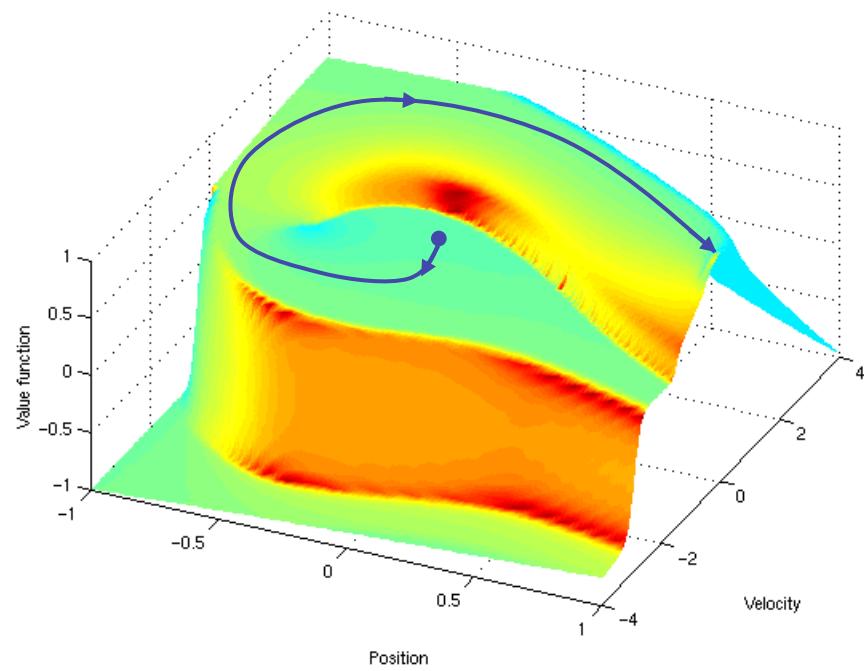
Therefore, given V^* , one-step-ahead search produces the long-term optimal actions.

Car-on-the-Hill Optimal Value Function



Get to the top of the hill
as quickly as possible
(roughly)

Predicted minimum time
to goal (negated)



Munos & Moore “Variable resolution discretization for high-accuracy solutions of optimal control problems”, IJCAI 99.

What can you do with Optimal Action-Value Functions?

Given Q^* , the agent does not even have to do a one-step-ahead search:

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

How do you find value functions?

Somehow solve the “**Bellman Equation**”

- Conventional way: **Dynamic Programming**
- Approximate methods



Richard Bellman
1920-1984

Bellman Equation for V^π

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots \\ &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \dots) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

So:

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\} \end{aligned}$$

Or, without the expectation operator:

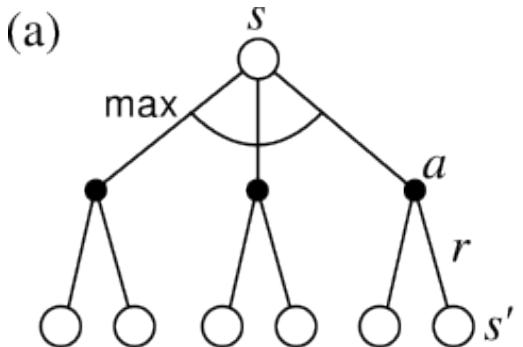
$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$$

Bellman Optimality Equation for V^*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

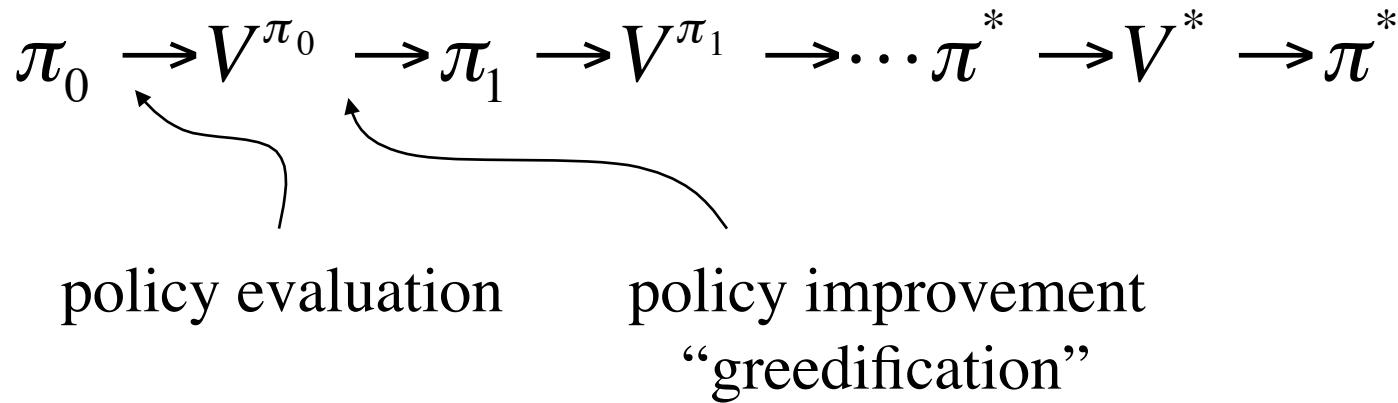
$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s, a) \\ &= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\} \\ &= \max_{a \in A(s)} \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^*(s')] \end{aligned}$$

The relevant backup diagram:



V^* is the unique solution of this system of nonlinear equations.

Policy Iteration



Solving Bellman Optimality Equations

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
 - accurate knowledge of environment dynamics;
 - we have enough space and time to do the computation;
 - the Markov Property.
- How much space and time do we need?
 - polynomial in number of states (via dynamic programming),
 - BUT, number of states is often huge (e.g., backgammon has about $10^{**}20$ states).
- We usually have to settle for approximations.
- Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

Semi-Markov Decision Processes (SMDPs)

- Generalization of an MDP where there is a waiting, or dwell, time τ in each state
- Transition probabilities generalize to $P(s', \tau | s, a)$
- Bellman equations generalize, e.g., for a discrete time SMDP:

$$V^*(s) = \max_{a \in A(s)} \sum_{s', \tau} P(s', \tau | s, a) [R_{ss'}^a + \gamma^\tau V^*(s')]$$

where $R_{ss'}^a$ is now the amount of discounted reward expected to accumulate over the waiting time in s upon doing a and ending up in s'

Partially-Observable MDPs (POMDPs)

Decision processes in which environment acts like an MDP but the agent cannot observe states. Instead it receives observations which depend probabilistically on the underlying states.

Challenges for RL

- Curse of Dimensionality
- Curse of Real-World Samples
- Curse of Under-Modeling
- Curse of Goal Specification

from “Reinforcement Learning in Robotics: A Survey
Kober, Bagnell, and Peters, 2013

Summary

- What is computational RL?
 - Ancient history
 - RL and supervised learning
- MDPs
- Understanding the degree of abstraction
- Value functions
- Bellman equations
- SMDPs
- POMDPs
- Major challenges

Next

- Understanding TD prediction
- TD control
 - Sarsa
 - Q-Learning
 - Actor-Critic architecture
- Function approximation
- Direct policy search
- Hierarchical RL
- Intrinsically motivated RL
- Summing up

Temporal Difference (TD) Prediction

Policy Evaluation (the prediction problem):

for a given policy π , compute the state-value function V^π

Simple Monte Carlo method :

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

↑
target: the actual return after time t

The simplest TD method, TD(0) :

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

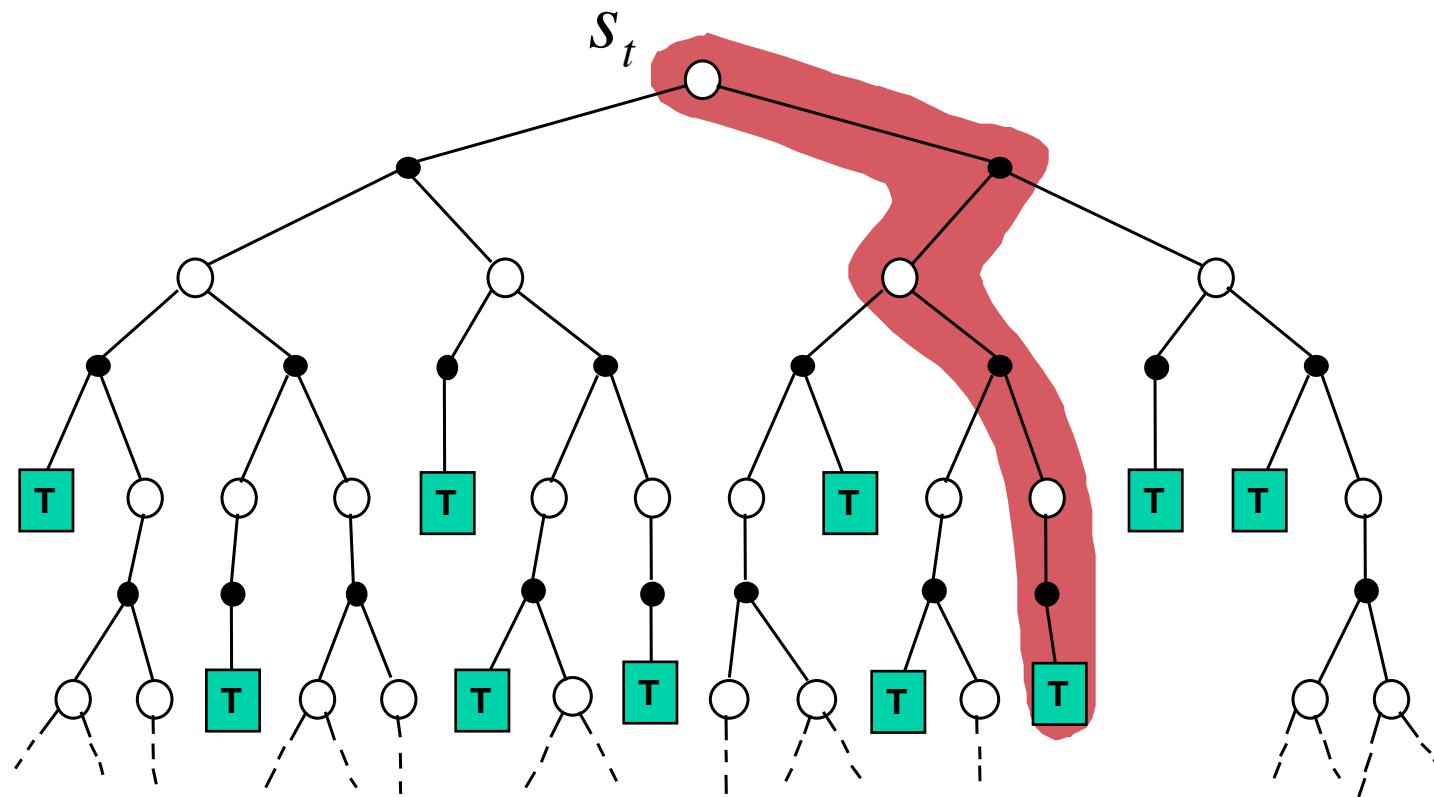


target: an estimate of the return

Simple Monte Carlo

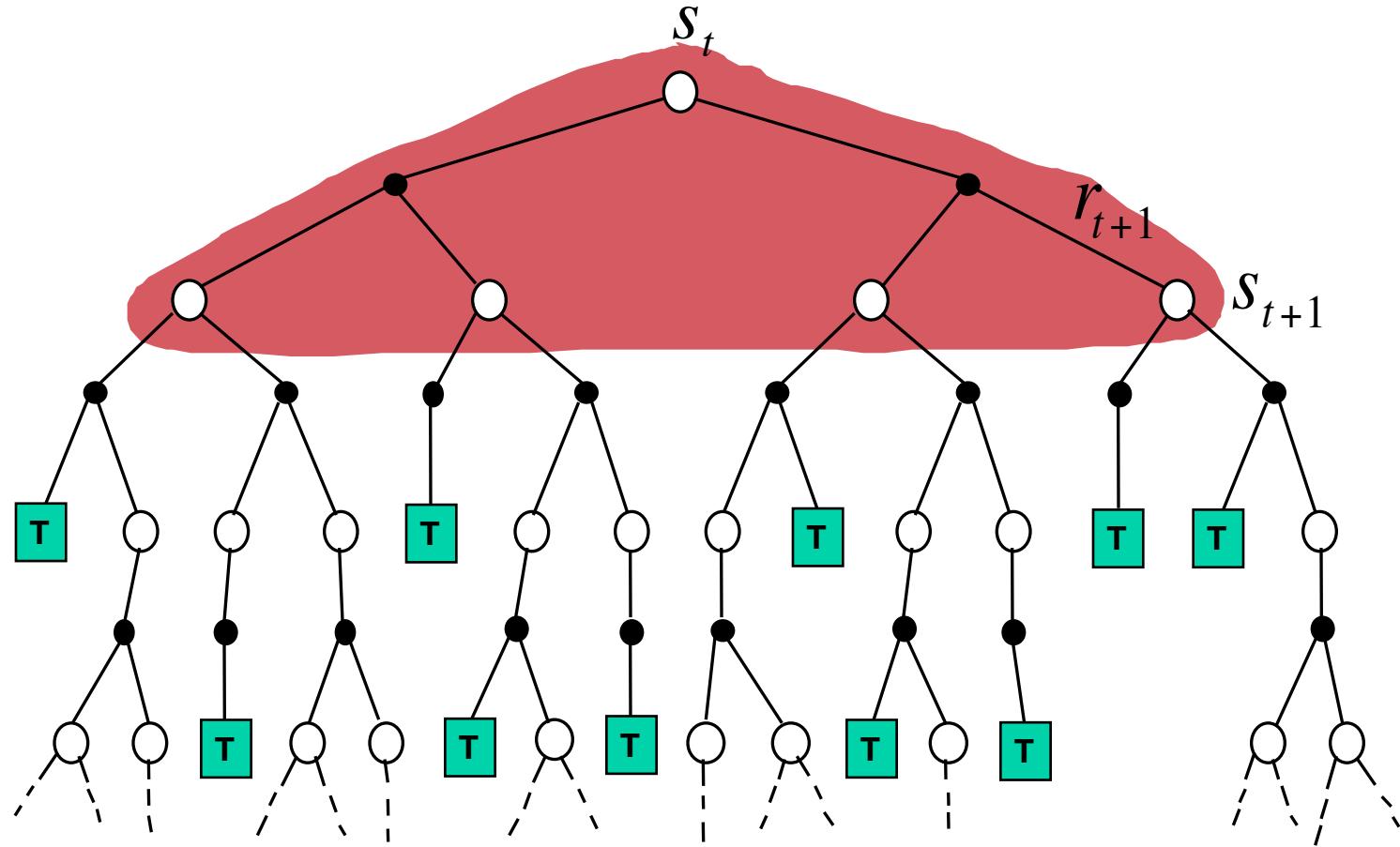
$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

where R_t is the actual return following state s_t .



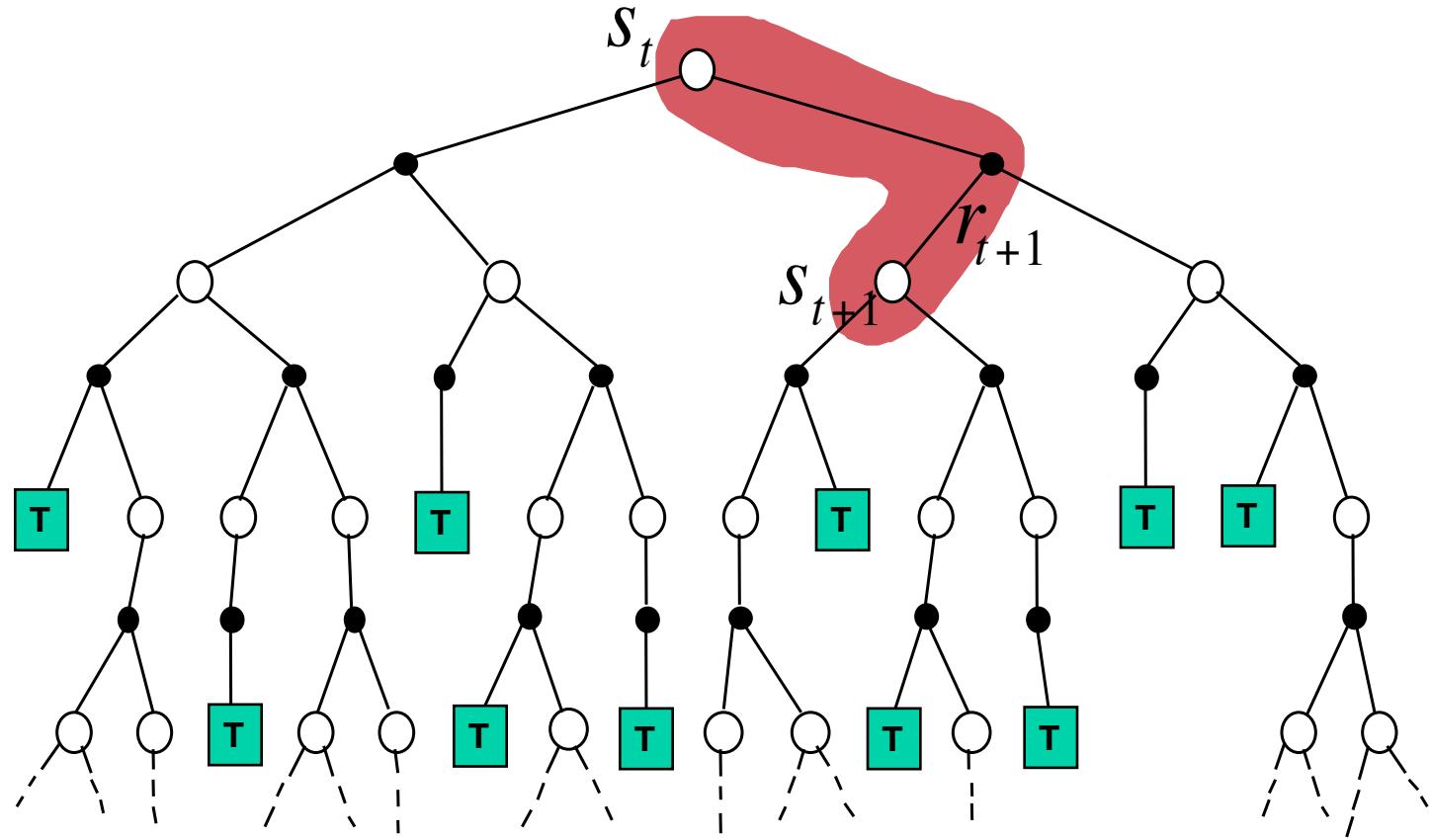
Dynamic Programming

$$V(s_t) \leftarrow E_{\pi} \{ r_{t+1} + \gamma V(s_t) \}$$



Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$



You are the predictor...

Suppose you observe the following 8 episodes:

A, 0, B, 0

B, 1

B, 1

B, 1

$V(A) ?$

B, 1

B, 1

$V(B) ?$

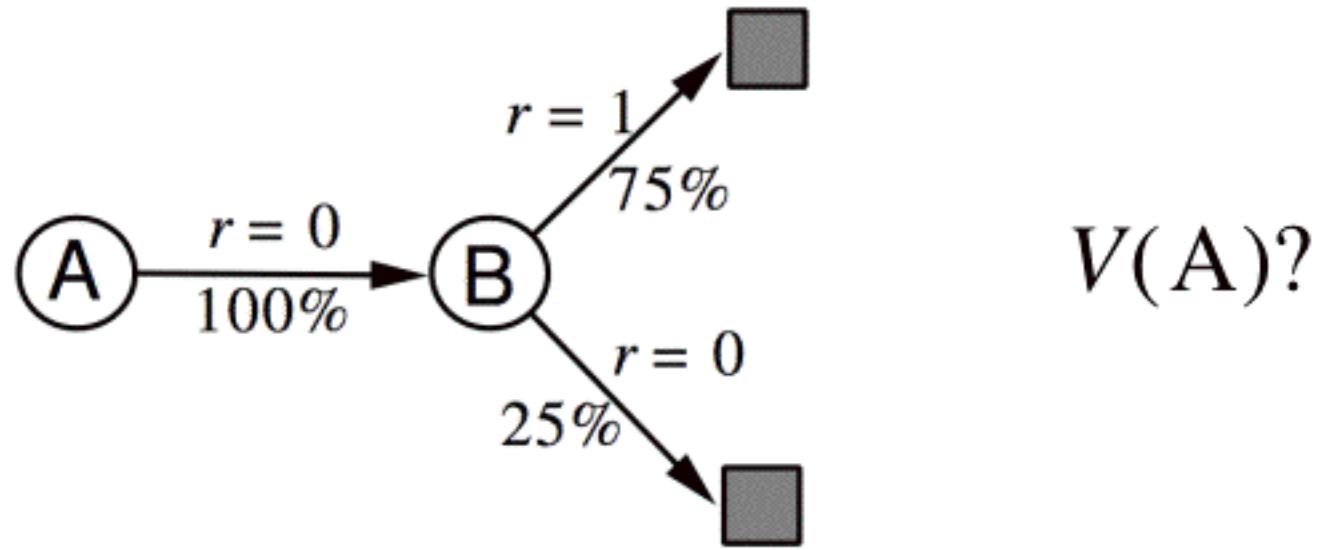
B, 1

B, 0

You are the predictor...

- The prediction that best matches the training data is $V(A)=0$
 - This **minimizes the mean-square-error** on the training set
 - This is what a Monte Carlo method gets
- If we consider the sequentiality of the problem, then we would set $V(A)=.75$
 - This is correct for the **maximum likelihood** estimate of a Markov model generating the data
 - i.e., if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts (how?)
 - This is called the **certainty-equivalence estimate**
 - This is what TD(0) gets

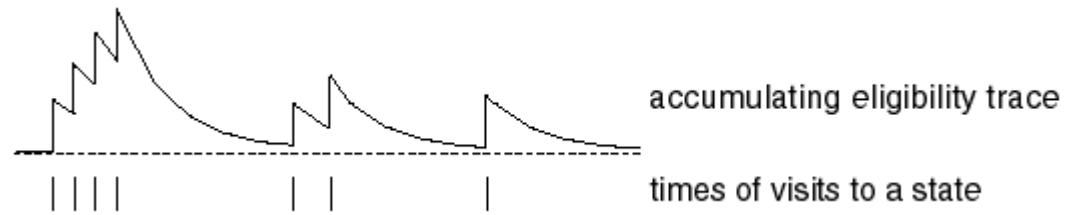
You are the predictor...



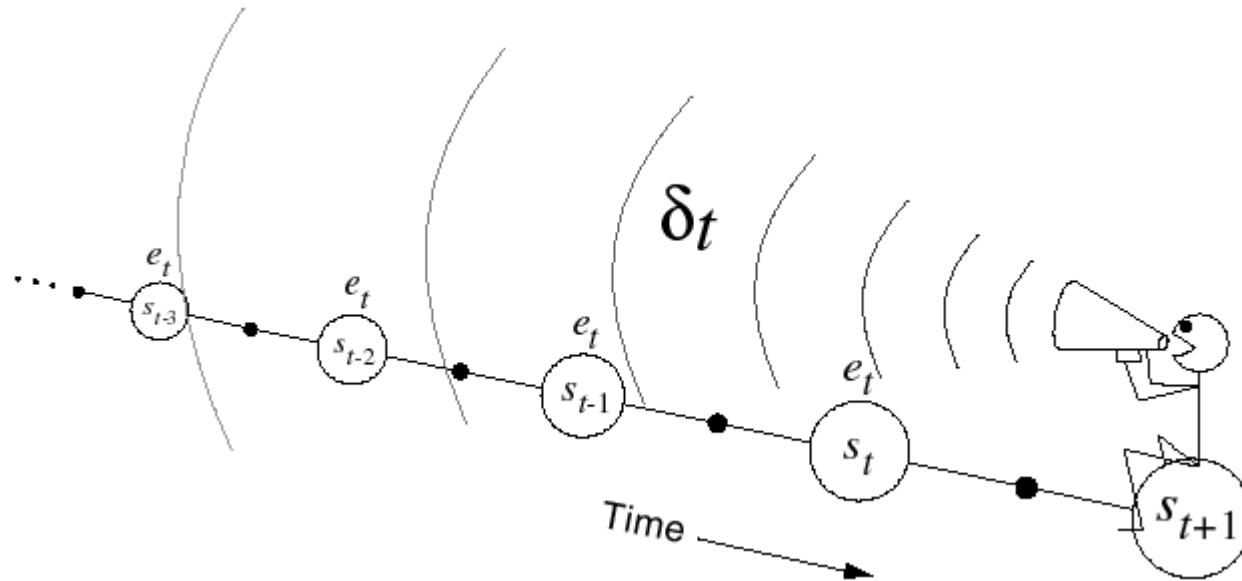
TD(λ)

- New variable called *eligibility trace*: $e_t(s) \in \Re^+$
 - On each step, decay all traces by $\gamma \lambda$ and increment the trace for the current state by 1
 - Accumulating trace

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma \lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$



TD(λ)



$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

- Shout δ_t backwards over time
- The strength of your voice decreases with temporal distance by $\gamma\lambda$

TD(λ)

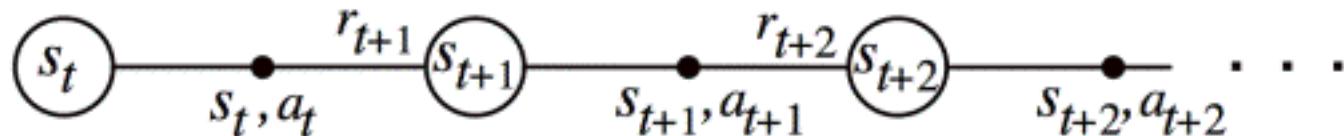
- Using update rule:

$$\Delta V_t(s) = \alpha \delta_t e_t(s)$$

- As before, if you set λ to 0, you get to TD(0)
- If you set λ to 1, you get MC but in a better way
 - Can apply TD(1) to continuing tasks
 - Works incrementally and on-line (instead of waiting to the end of the episode)

Learning and Action-Value Function

Estimate Q^π for the current behavior policy π .



After every transition from a nonterminal state s_t , do this :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$.

TD Learning is not RL!

TD learning is a variant of supervised learning:

It is about prediction, not control

Sarsa

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

 Initialize s

 Choose a from s using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

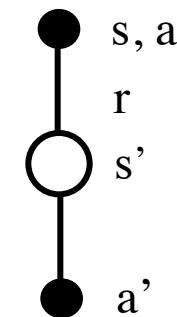
 Take action a , observe r, s'

 Choose a' from s' using policy derived from Q (e.g., ϵ -greedy)

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$$

$s \leftarrow s'; a \leftarrow a'$;

 until s is terminal

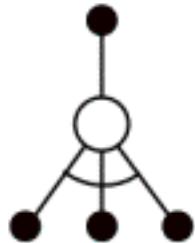


Q-Learning

Chris Watkins, 1989

One - step Q - learning :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$



Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

 Initialize s

 Repeat (for each step of episode):

 Choose a from s using policy derived from Q (e.g., ε -greedy)

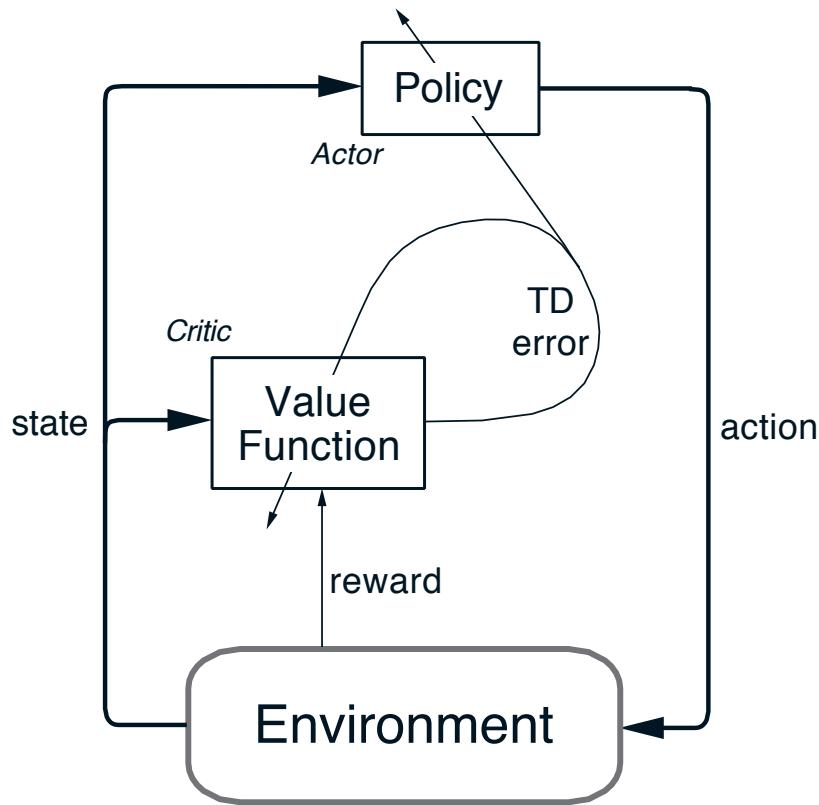
 Take action a , observe r, s'

$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

$s \leftarrow s'$;

 until s is terminal

Actor-Critic Methods



- Explicit representation of policy as well as value function
- Minimal computation to select actions
- Can learn an explicit stochastic policy
- Can put constraints on policies
- Appealing as psychological and neural models

Function Approximation

As usual: Policy Evaluation (the prediction problem):
for a given policy π , compute the state-value function V^π

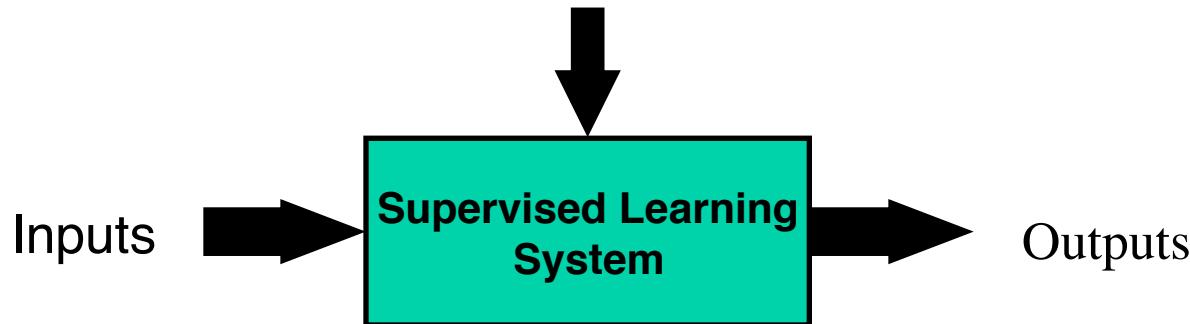
In earlier lectures, value functions were stored in lookup tables.

Here, the value function estimate at time t , V_t , depends
on a **parameter vector** θ_t , and only the parameter vector
is updated.

e.g., θ_t could be the vector of connection weights
of a neural network.

Adapt Supervised-Learning Algorithms

Training Info = desired (target) outputs



Training example = {input, target output}

Error = (target output – actual output)

Backups as Labeled Examples

e.g., the TD(0) backup :

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

As a training example:

$$\{\text{description of } s_t, r_{t+1} + \gamma V(s_{t+1})\}$$



input



target output

Do you always need value functions?

Parameterize the policy

Search directly in the policy-parameter space

Quadrupedal Locomotion

Kohl and Stone, UT Austin 2004

Before Learning

After 1000 trials, or about 3 hours

Quadrupedal Locomotion

Kohl and Stone, UT Austin 2004



Before Learning



After 1000 trials, or about 3 hours

Quadrupedal Locomotion

Kohl and Stone, UT Austin 2004



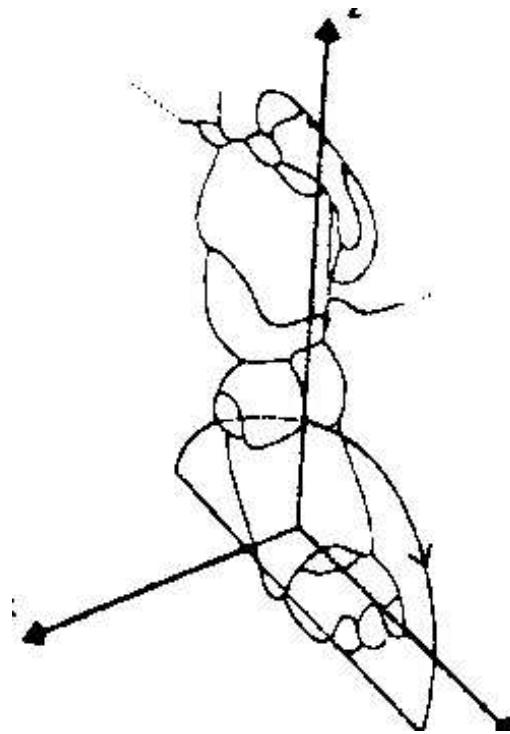
Before Learning



After 1000 trials, or about 3 hours

Direct Policy Search

Policy Parameterization:



- Half-elliptical locus for each foot
- 12 parameters:
 - Position of front locus (x, y, z)
 - Position of rear locus (x, y, z)
 - Locus length
 - Locus skew (for turning)
 - Height of front of body
 - Height of rear of body
 - Time for each foot to move through locus
 - Fraction of time each foot spends on the ground

Simple stochastic hillclimbing to increase speed

Direct Policy Search

□ Why?

- Value functions can be very complex for large problems, while policies have a simpler form.
- Convergence of learning algorithms not guaranteed for approximate value functions whereas policy gradient methods are well-behaved with function approximation.
- Value function methods run into a lot of problems in partially observable environments. Policy gradient methods can be better behaved in this scenario.

Hierarchical RL

RL typically solves a *single* problem *monolithically*.

Hierarchical RL:

- Create and use higher-level macro-actions.
- Problem now contains *subproblems*.
- Each subproblem may also be an RL problem.

Hierarchical RL provides a theoretical basis for skill acquisition.

Options Framework: methods for learning and planning using higher-level actions (*options*). (Sutton, Precup and Singh, 1999)

“Temporal Abstraction”

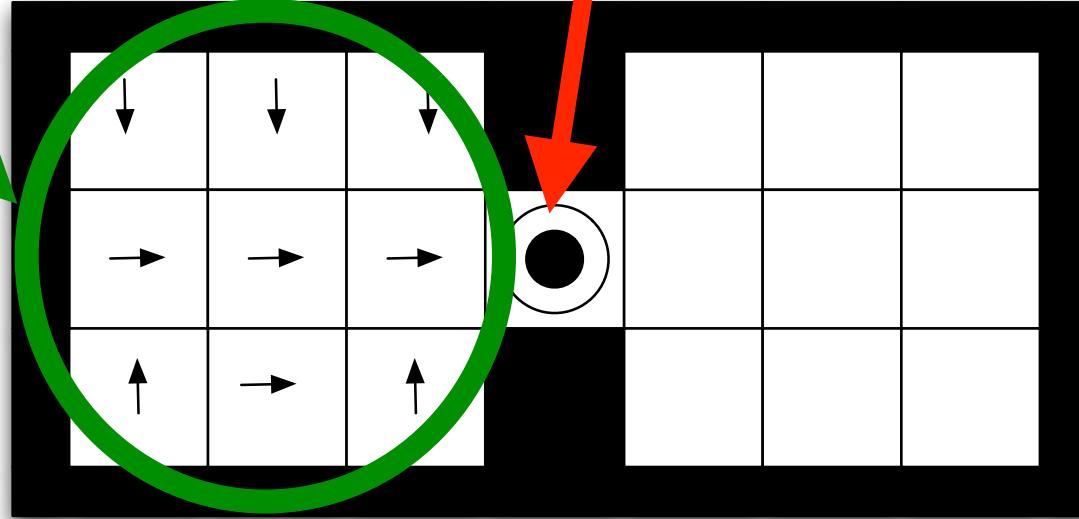
How can an agent represent stochastic, closed-loop, temporally-extended courses of action? How can it act, learn, and plan using such representations?

- HAMs (Parr & Russell 1998; Parr 1998)
- MAXQ (Dietterich 2000)
- Options framework (Sutton, Precup & Singh 1999; Precup 2000)

Options

An option o is a closed loop control policy unit:

- Initiation set
- Termination condition
- Option policy



Intrinsic Motivation

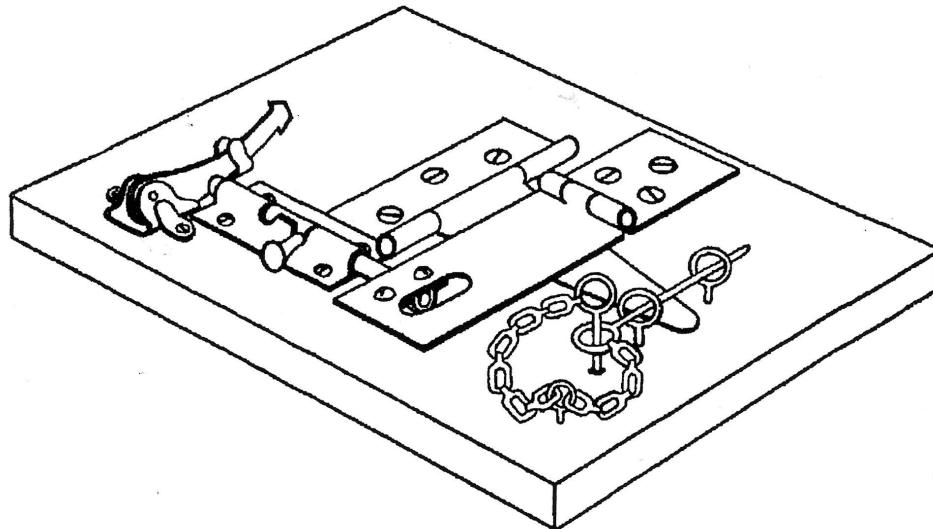
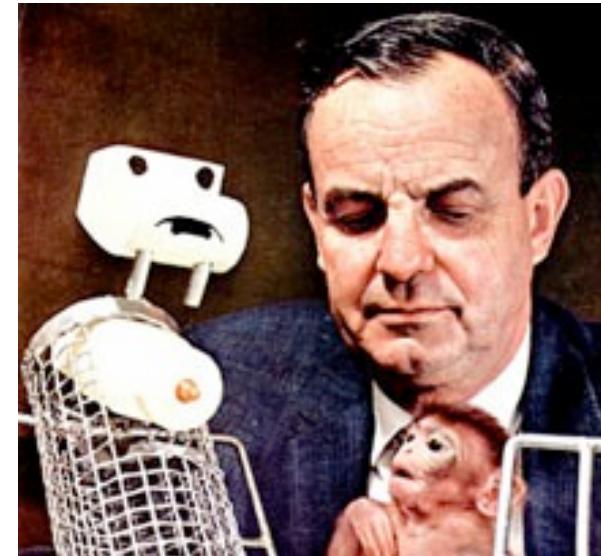


FIG. 1. SIX-DEVICE MECHANICAL PUZZLE



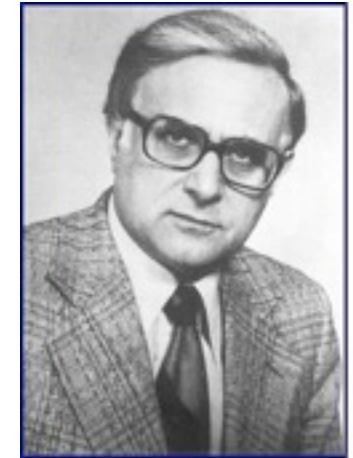
H. Harlow, “Learning and Satiation of Response in Intrinsic Motivation Complex Puzzle Performance”, *Journal of Comparative and Physiological Psychology*, Vol. 43, 1950

Harlow 1950

“A manipulation drive, strong and extremely persistent, is postulated to account for learning and maintenance of puzzle performance. It is further postulated that drives of this class represent a form of motivation which may be as primary and as important as the homeostatic drives.”

Daniel Berlyne 1924 - 1976

“Curiosity and Exploration”, *Science*, 1966



“As knowledge accumulated about the conditions that govern exploratory behavior and about how quickly it appears after birth, it seemed less and less likely that this behavior could be a derivative of hunger, thirst, sexual appetite, pain, fear of pain, and the like, or that stimuli sought through exploration are welcomed because they have previously accompanied satisfaction of these drives.”

Some Suggested Principles behind IM

- Conflict (Berlyne)
 - Novelty
 - Surprise
 - Incongruity
- Mastery/Control:
 - 1901 Groos: "We demand a knowledge of effects, and to be ourselves the producers of effects."
 - 1942 Hendrick: instinct to master by which an animal has an inborn drive to do and to learn how to do."
 - 1953 Skinner: behaviors that occur in the absence of obvious rewards may be maintained by control over the environment.
 - D. Polani and colleagues: Empowerment
- Learning progress (Schmidhuber, Kaplan & Oudeyer)
- Compression (Schmidhuber)
- Others...

Barto, A. G. (2013). **Intrinsic Motivation and Reinforcement Learning**. In: Intrinsically Motivated Learning in Natural and Artificial Systems, Baldassarre, G. and Mirolli, M., eds., Springer-Verlag, Berlin, pp. 17–47.

Singh, S., Lewis, R.L., & Barto, A.G. (2009).

Where Do Rewards Come From? *Proceedings of the 31st Annual Conference of the Cognitive Science Society (CogSci 2009)*, pp. 2601-2606

Singh, S., Lewis, R.L., Barto, A.G., & Sorg, J. (2010).

Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective. *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 70–82.

Sorg, J., Singh, S., & Lewis, R. (2010).

Internal Rewards Mitigate Agent Boundedness

Proceedings of the 27th International Conference on Machine Learning (ICML).

The Plan

Part I

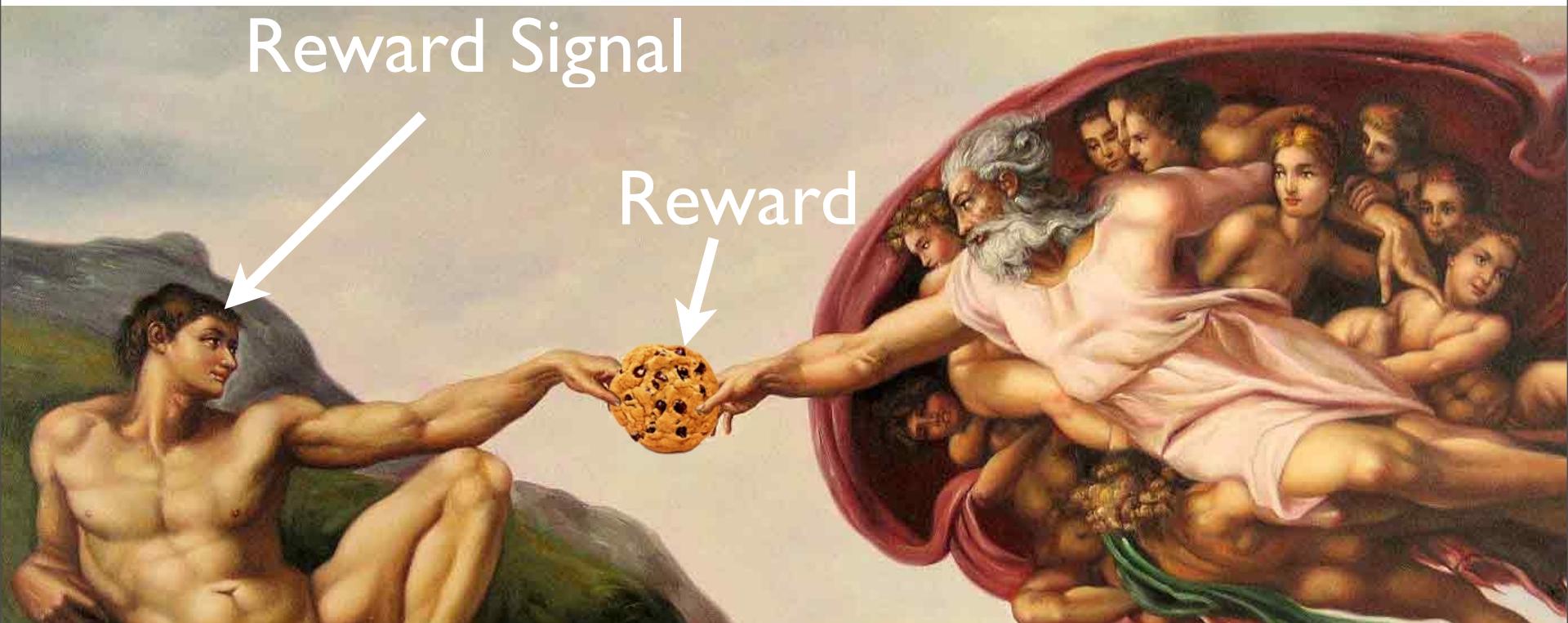
- What is computational reinforcement learning?
 - Ancient history
 - RL and supervised learning
- Agent-Environment interaction
 - Markov Decision Processes (MDPs)
 - Some standard simple examples
- Understanding the degree of abstraction
- Value functions
- Bellman Equations
- Semi-Markov Decision Processes (SMDPs)
- Partially Observable MDPs (POMDPs)
- Major challenges for RL

The Overall Plan

Part 2

- Understanding Temporal Difference prediction
- Temporal difference control
 - Sarsa
 - Q-Learning
 - Actor-Critic architecture
- Function approximation
- Direct policy search
- Hierarchical RL
- Intrinsically motivated RL
- Summing up

Thanks!



Thanks to Will Dabney (and Michelangelo)