

CSC242: Intro to AI

Lecture 20

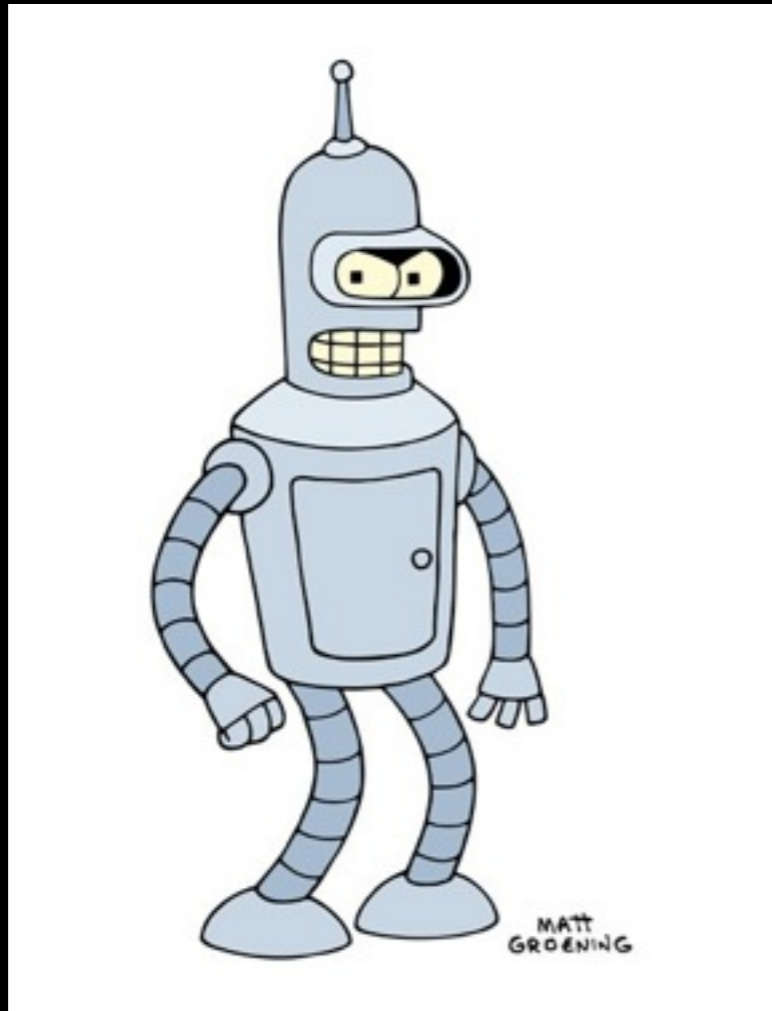
Reinforcement Learning I

A Joke

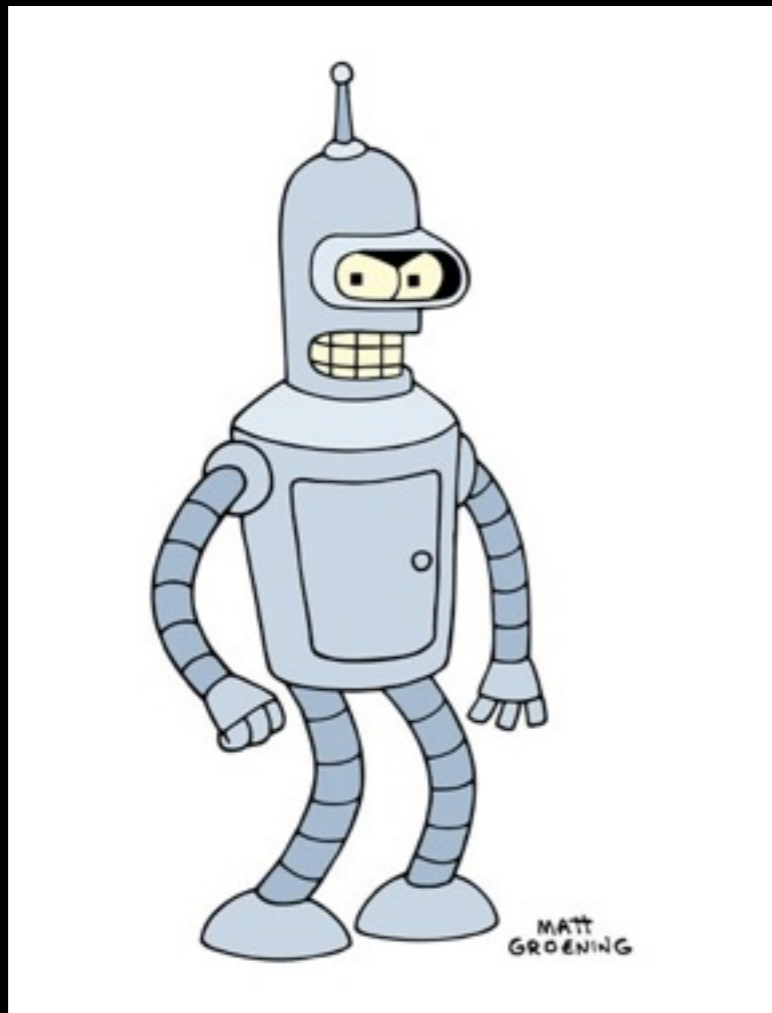
A robot walks up to a counter



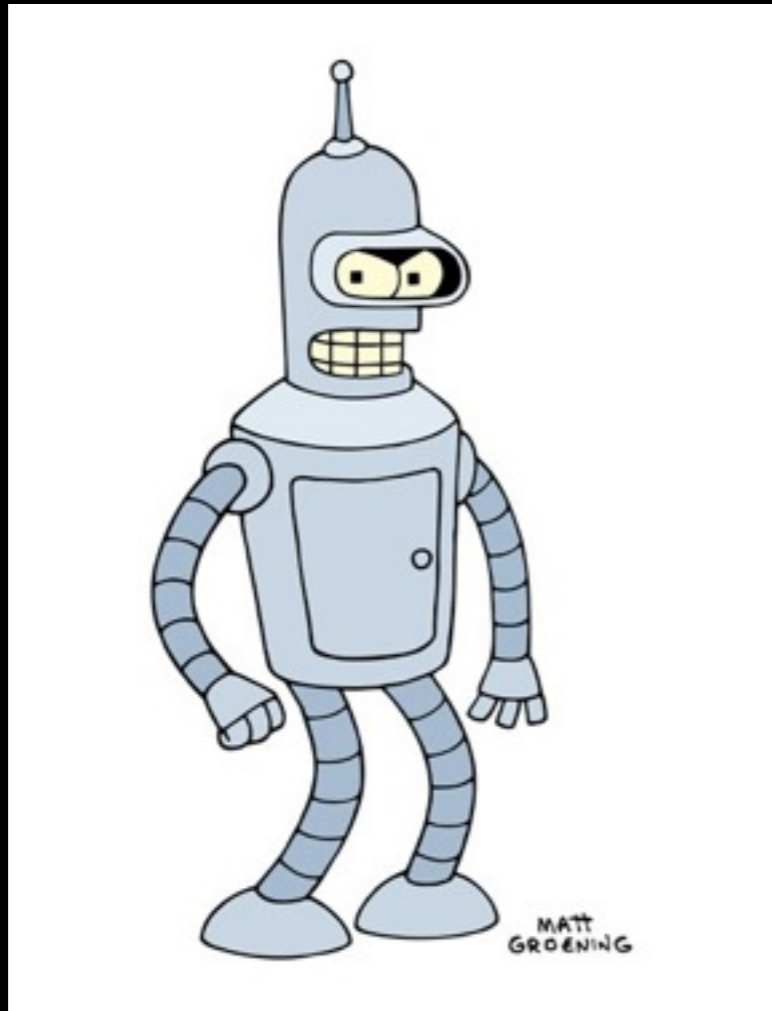
and says, "I'll have a
beer"



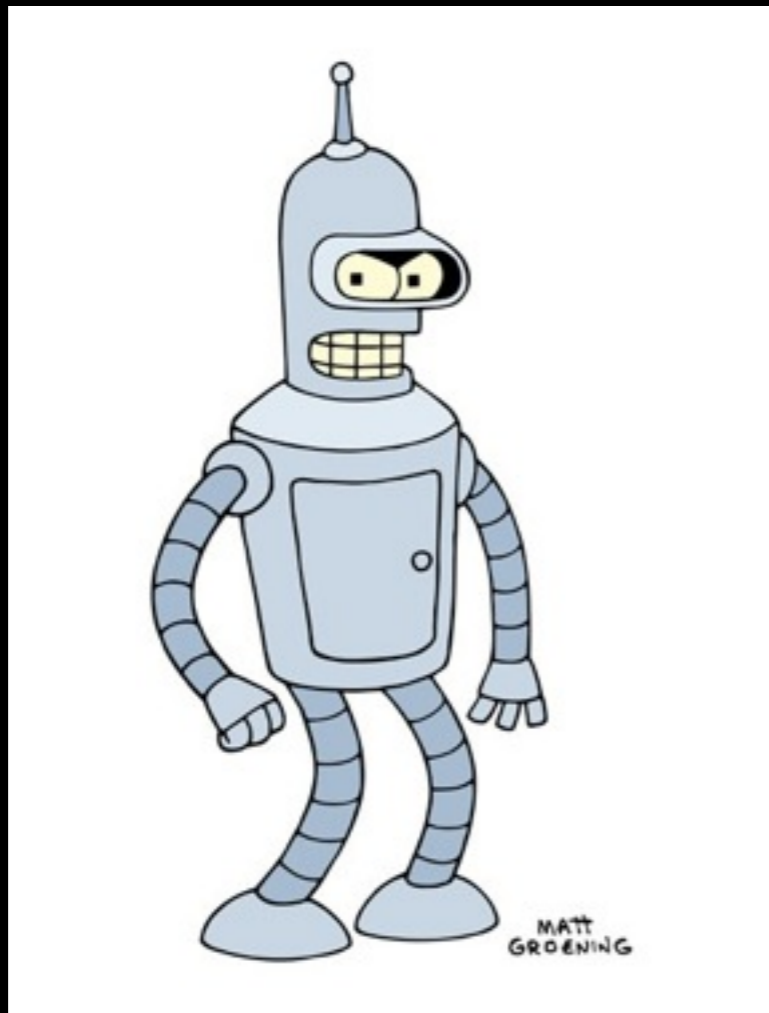
The human at the counter says, "I can't serve you a beer"



The robot says, “Is it because you discriminate against robots?!”



The human says, “No, it’s because
this is a hardware store”



Reinforcement Learning

- Learning how to act from rewards and punishments (reinforcement)
- What should robot learn?
 - Not to order a beer in a hardware store?
 - Not to go to a hardware store when he wants a beer?

Reinforcement Learning

- Learning how to act from rewards and punishments (reinforcement)
- What did the robot do wrong?
 - Ordering a beer?
 - Going into the hardware store?
 - Walking toward the hardware store?

Key Issues

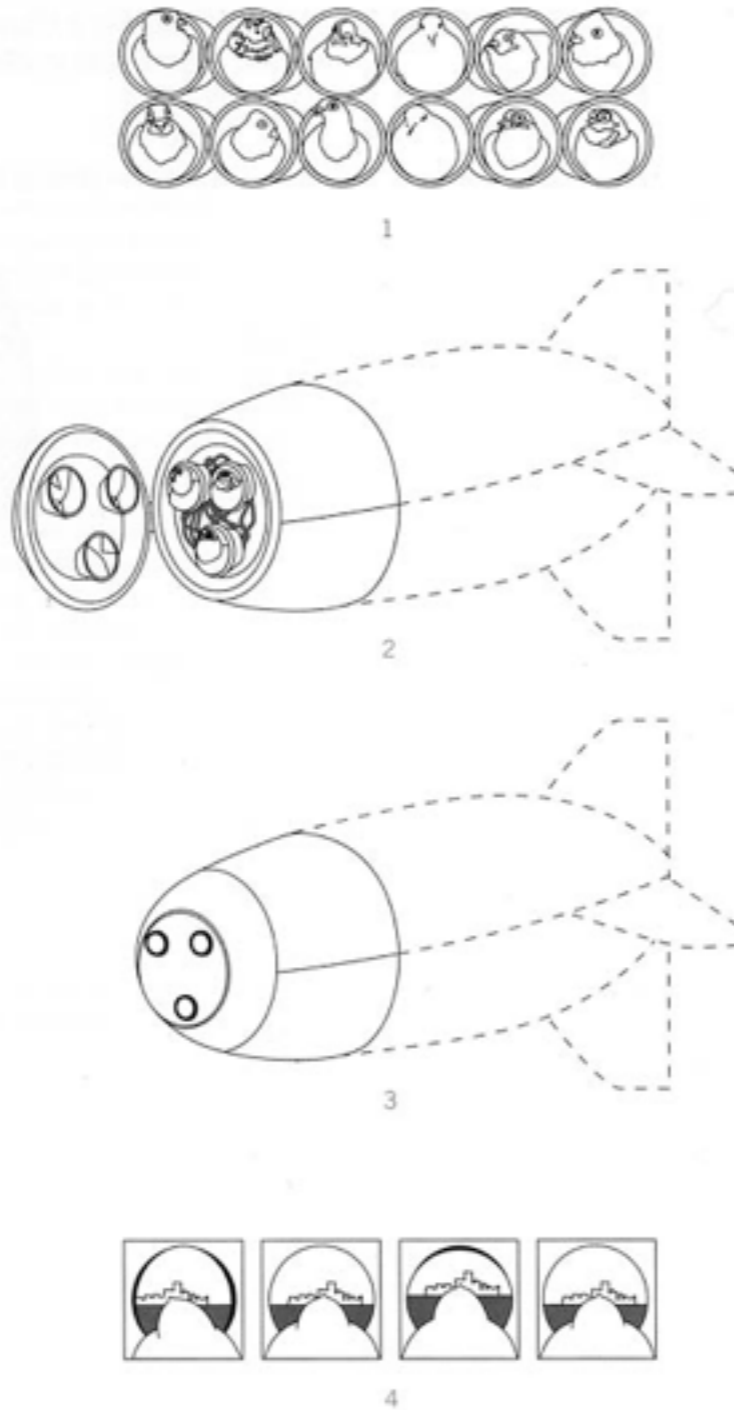
- How to account for the delay between actions and consequences?
- How to simultaneously learn a model of the environment while acting in the environment?
- “Imagine playing a new game whose rules you don’t know; after hundred or so moves, your opponent announces, ‘You lose.’”



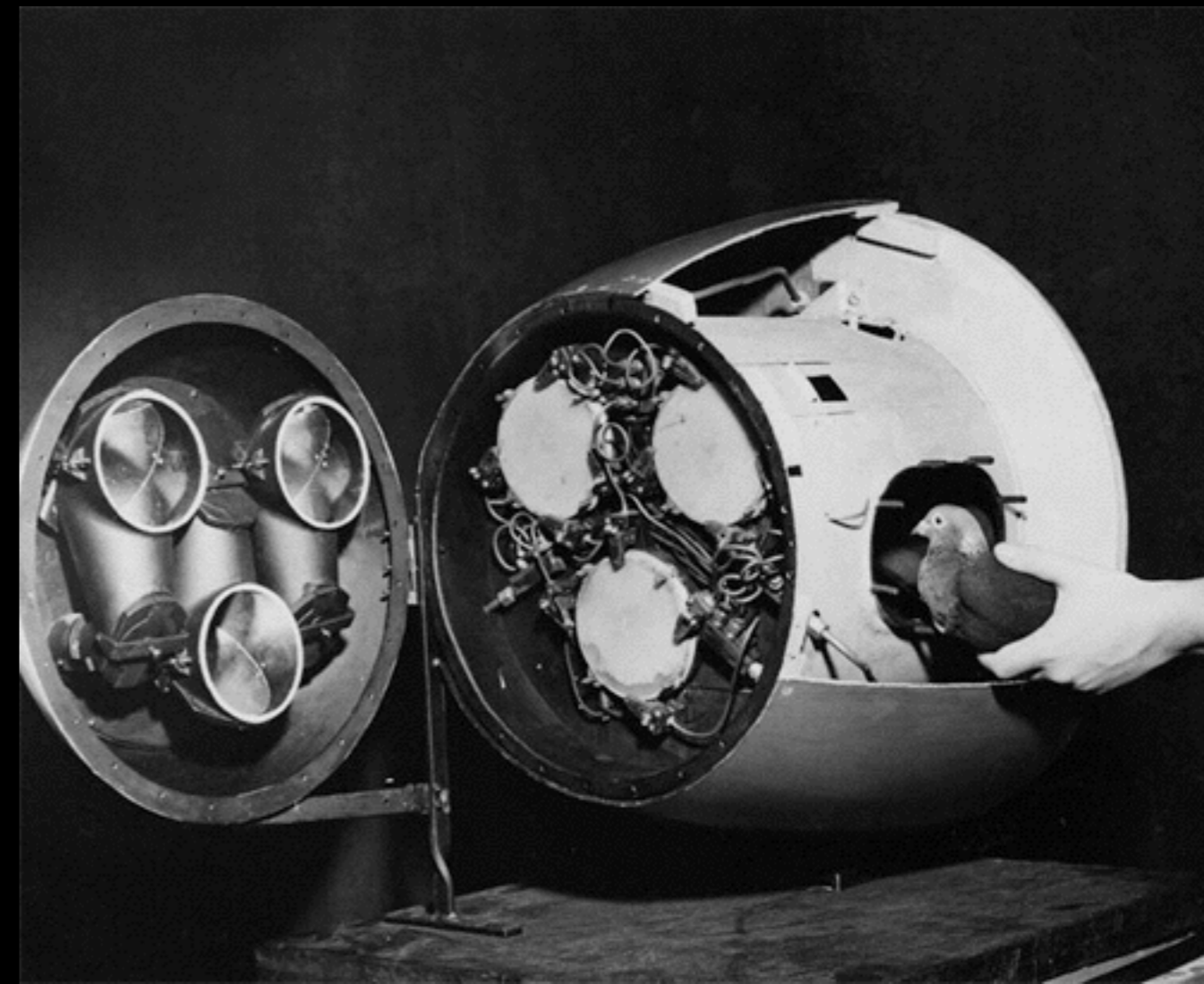
B.F. Skinner (1904-1990)

Project Pigeon

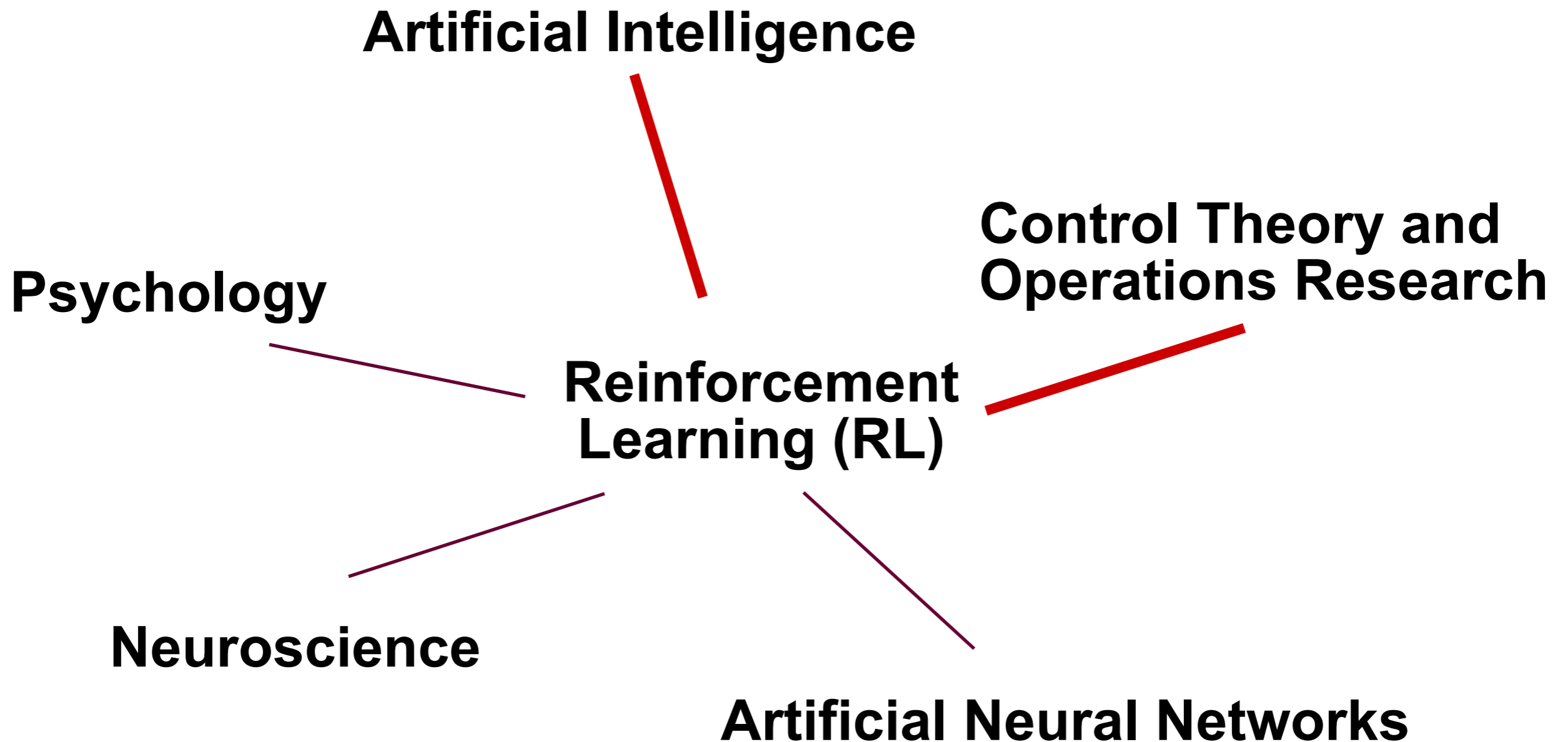
Project Pigeon was a classified research-and-development program during World War II. It was developed at a time when electronic guidance systems did not exist, and the only compensation for the inaccuracy of bombs was dropping them in quantity. This ingenious application of shaping would have dramatically increased the accuracy of bombs and decreased civilian casualties. Despite favorable performance tests, however, the National Defense Research Committee ended the project—it seems they couldn't get over the idea that pigeons would be guiding their bombs.



1. Pigeons were trained to peck at targets on aerial photographs. Once a certain level of proficiency was obtained, pigeons were jacketed and mounted inside tubes.
2. The pigeons in their tubes were inserted into the nosecone of the bomb. Each nosecone used three pigeons in a type of voting system, whereby the pigeon pecks of two birds in agreement would overrule the errant pigeon pecks of a single bird.
3. Sealed in the bomb, the pigeons could see through glass lenses in the nosecone.
4. Once the bomb was released, the pigeons would begin pecking at their view of the target. Their pecks shifted the glass lens off-center, which adjusted the bomb's tail surfaces and, correspondingly, its trajectory.



Learning from Experience Plays a Role in ...

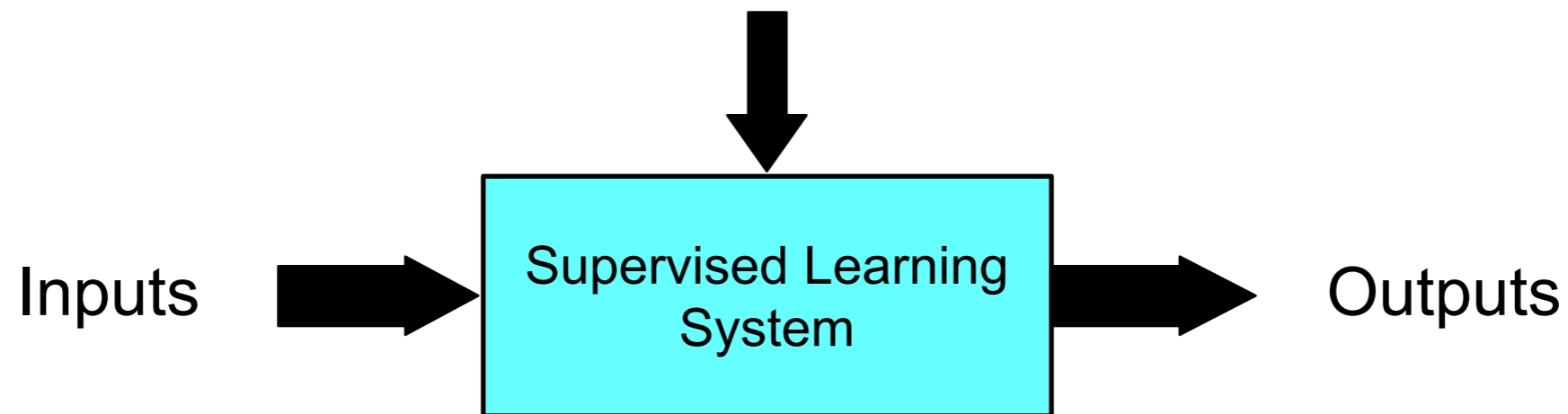


What is Reinforcement Learning?

- Learning from interaction
- Goal-oriented learning
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal

Supervised Learning

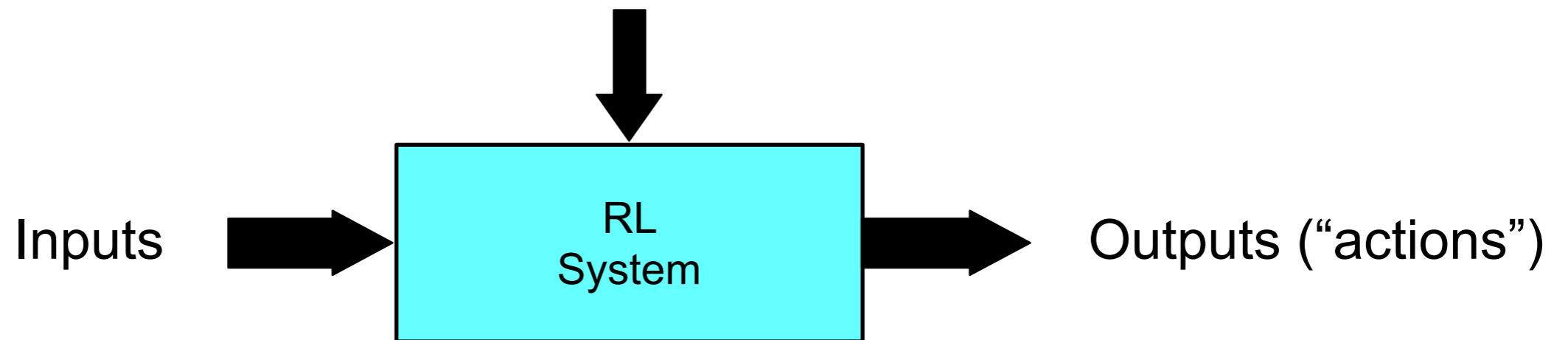
Training Info = desired (target) outputs



$\text{Error} = (\text{target output} - \text{actual output})$

Reinforcement Learning

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



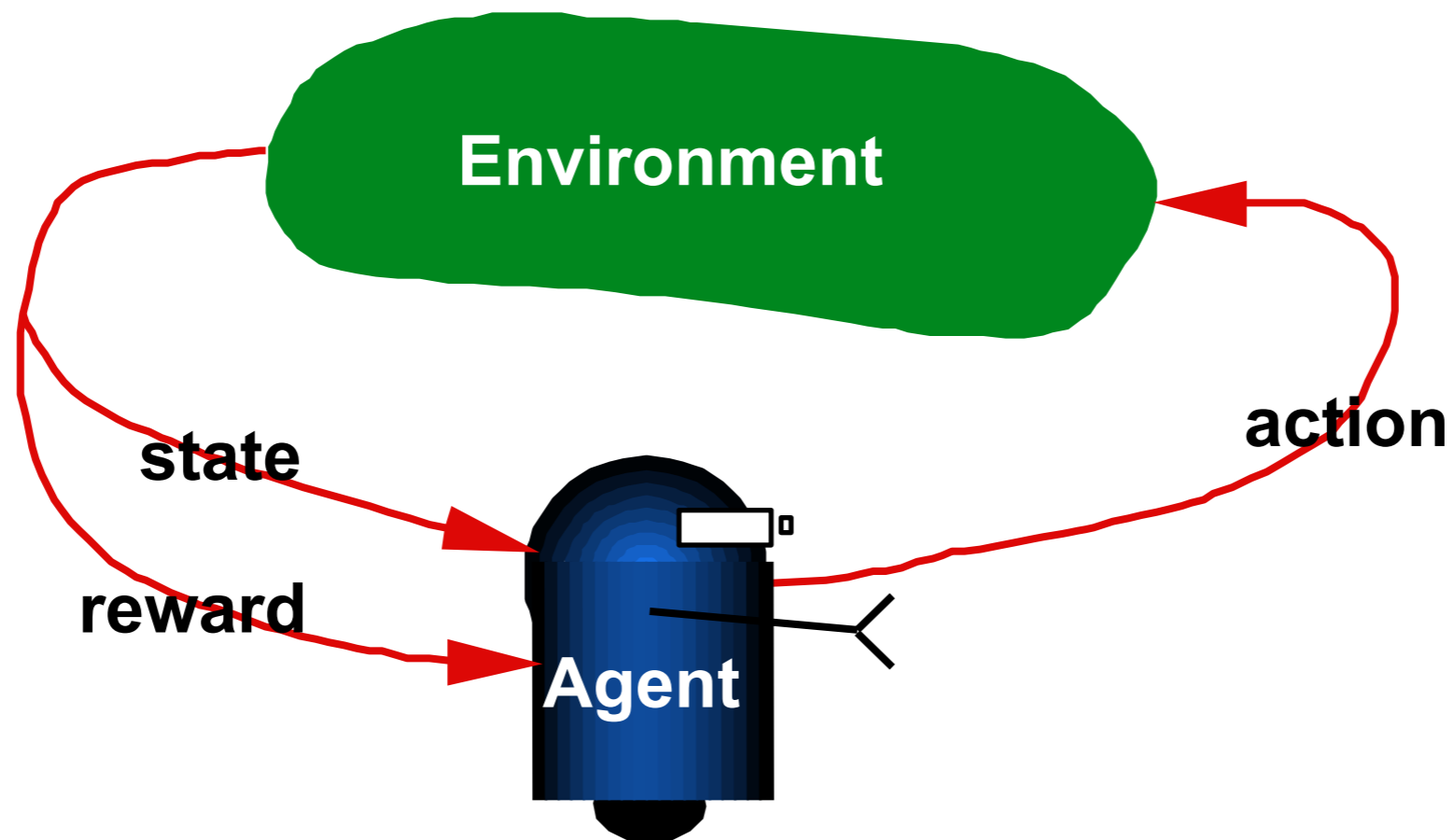
Objective: get as much reward as possible

Key Features of RL

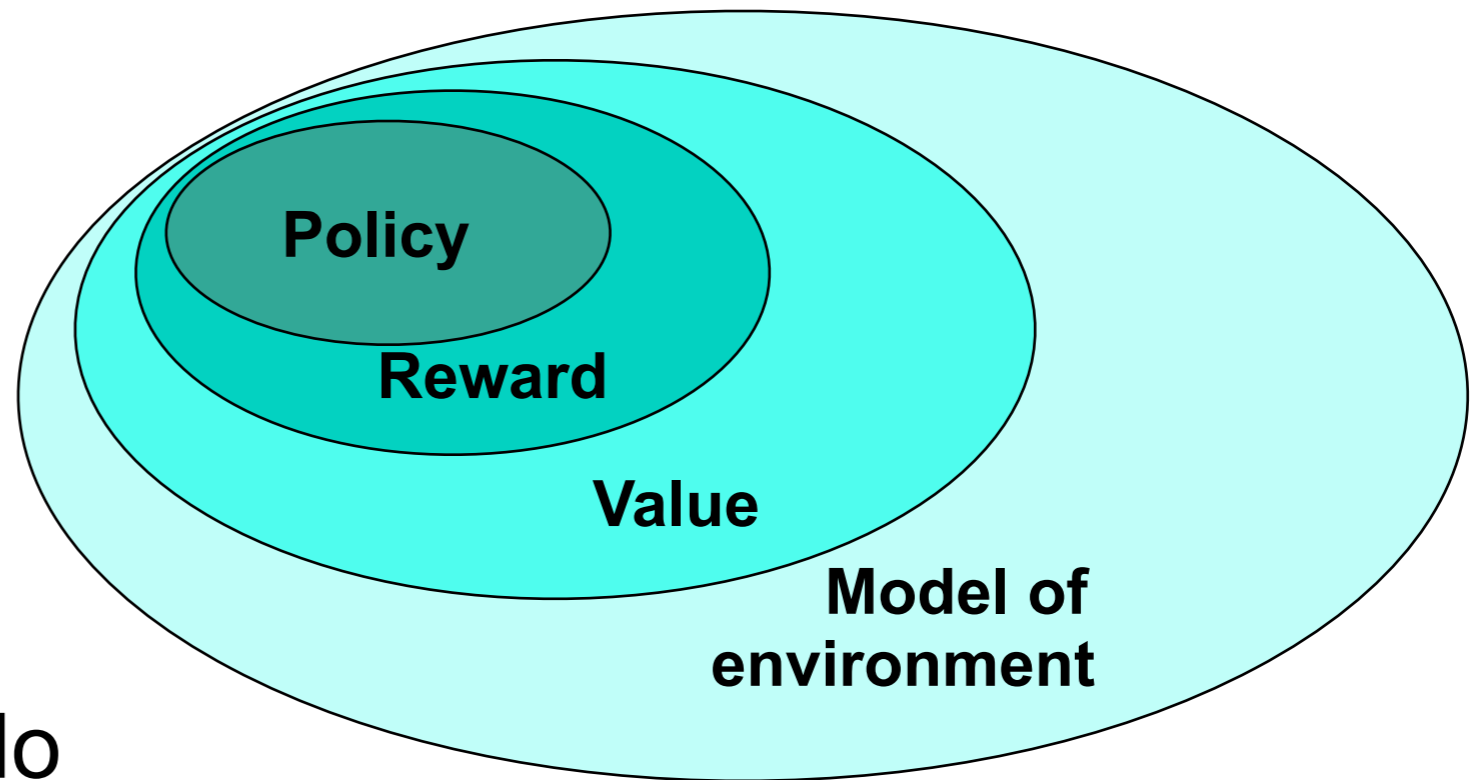
- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward (sacrifice short-term gains for greater long-term gains)
- The need to *explore* and *exploit*
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

Complete Agent

- Temporally situated
- Continual learning and planning
- Object is to *affect* the environment
- Environment is stochastic and uncertain

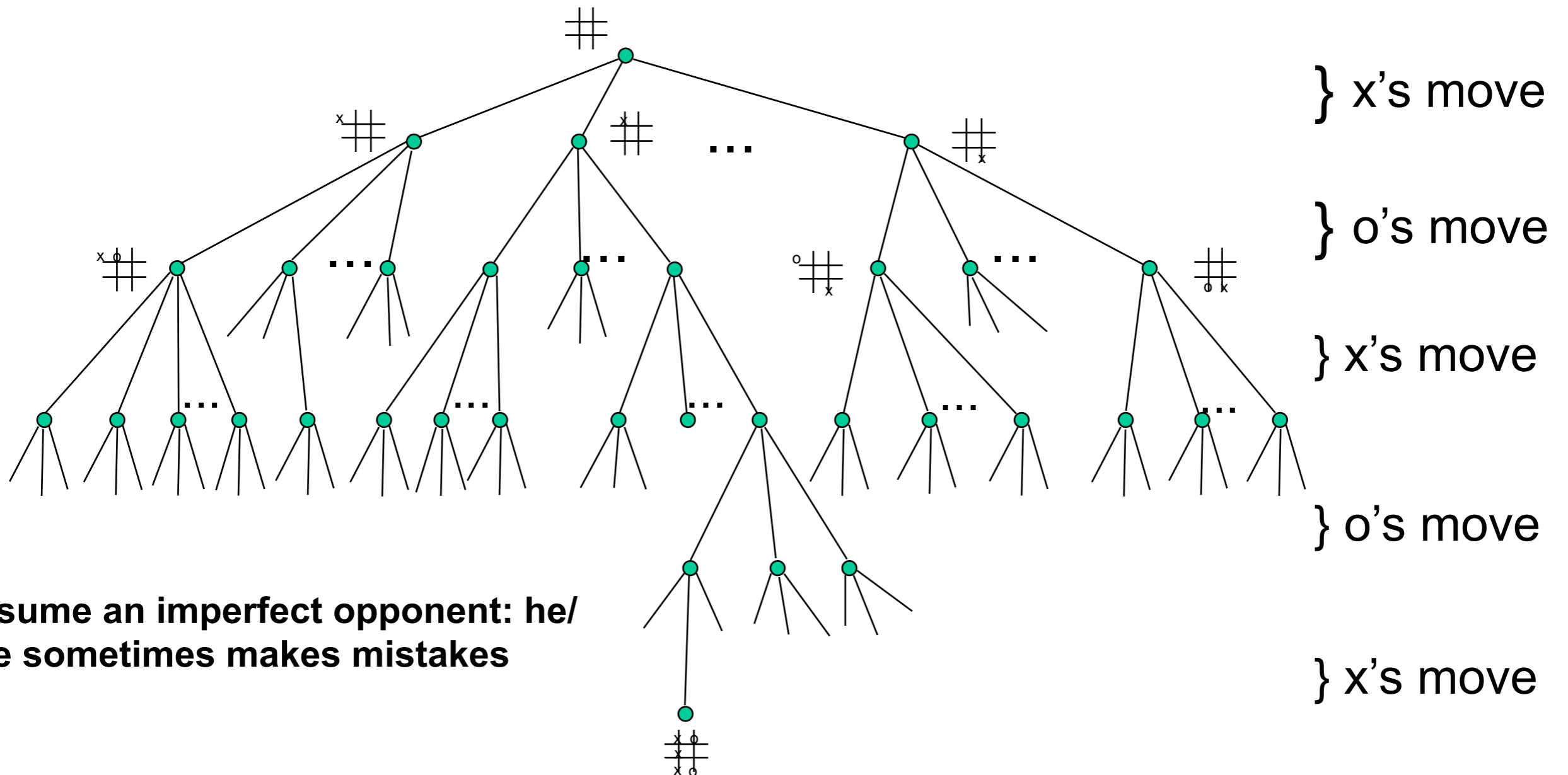
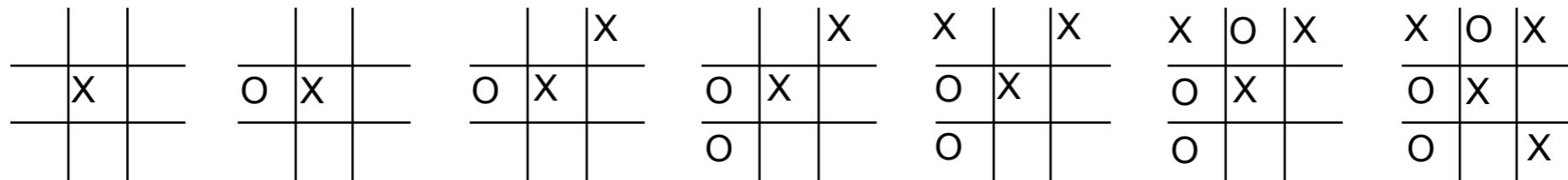


Elements of RL



- **Policy**: what to do
- **Reward**: what is good
- **Value**: what is good because it *predicts* reward
- **Model**: what follows what






An Extended Example: Tic-Tac-Toe



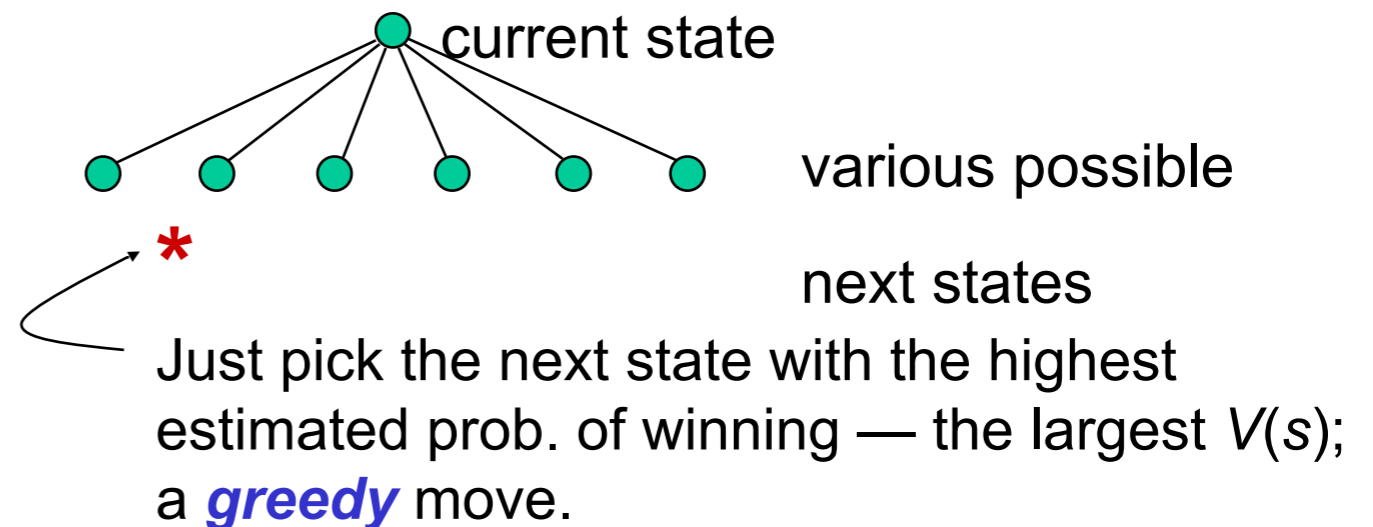
Assume an imperfect opponent: he/she sometimes makes mistakes

An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

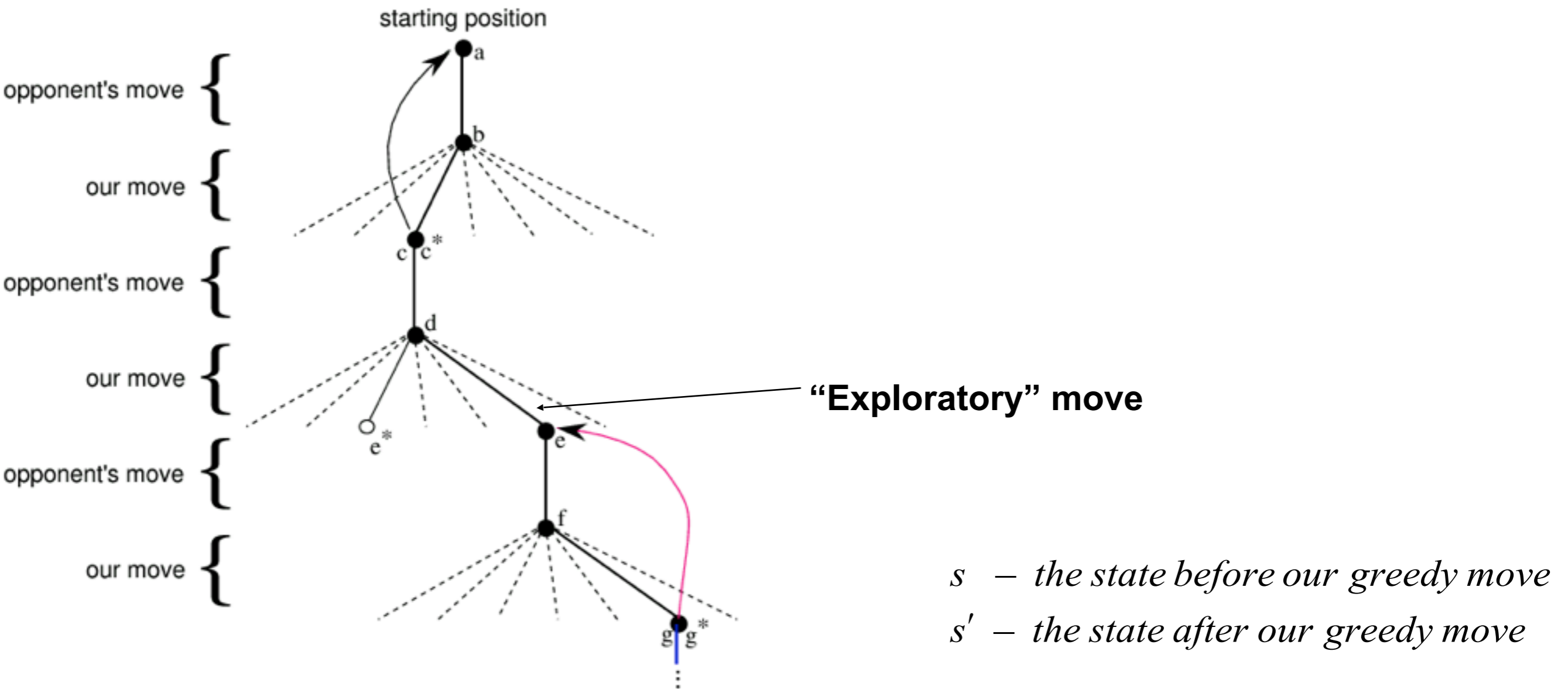
State	$V(s)$	estimated probability of winning
	.5	?
	.5	?
⋮	⋮	
	1	win
⋮	⋮	
	0	loss
⋮	⋮	
	0	draw

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** :

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

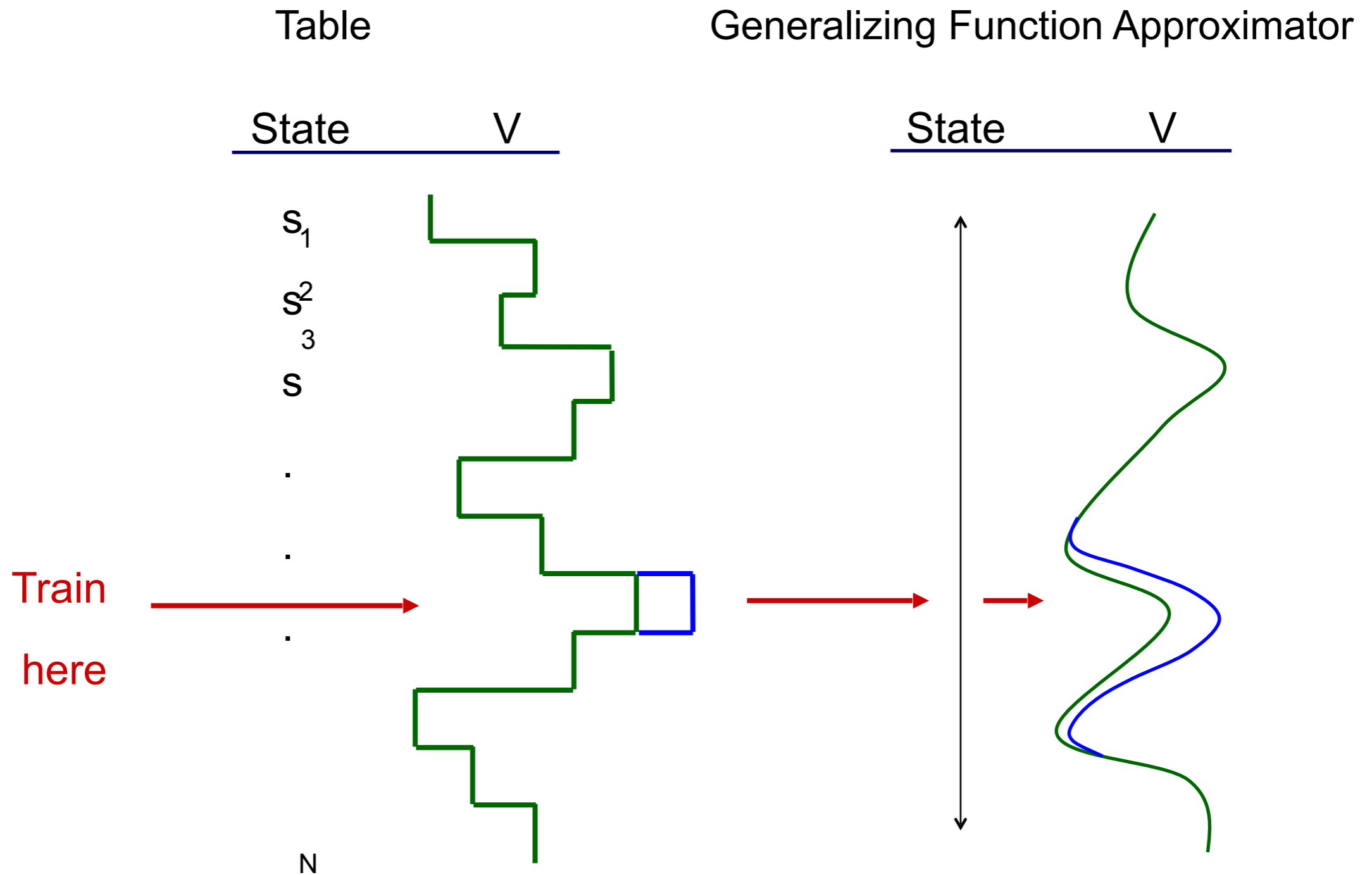
— a small positive fraction, e.g., $\alpha = .1$

the **step - size parameter**

How can we improve this T.T.T. player?

- Take advantage of symmetries
 - representation/generalization
 - How might this backfire?
- Do we need “random” moves? Why?
 - Do we always need a full 10%?
- Can we learn from “random” moves?
- Can we learn offline?
 - Pre-training from self play?
 - Using learned models of opponent?
- ...

e.g. Generalization



How is Tic-Tac-Toe Too Easy?

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

Some Notable RL Applications

■ **TD-Gammon:** Tesauro

- world's best backgammon program

■ **Elevator Control:** Crites & Barto

- high performance down-peak elevator controller

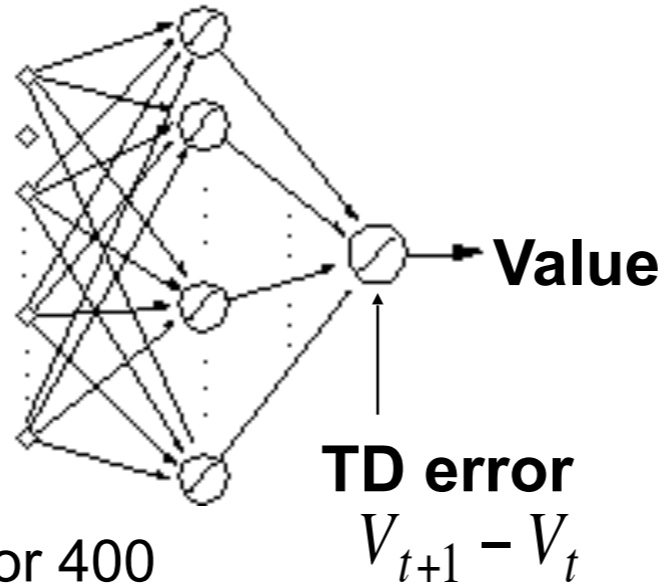
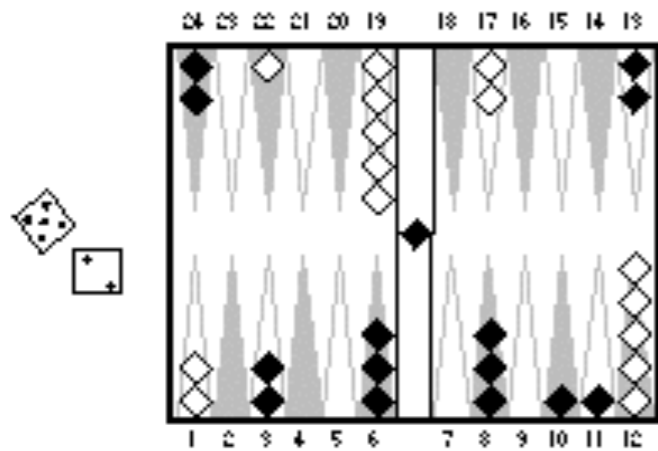
■ **Dynamic Channel Assignment:** Singh & Bertsekas, Nie & Haykin

- high performance assignment of radio channels to mobile telephone calls

■ ...

TD-Gammon

Tesauro, 1992–1995



Action selection
by 2–3 ply search

Effective branching factor 400

Start with a random network

Play very many games against self

Learn a value function from this simulated experience

This produces arguably the best player in the world

Evaluative Feedback

- **Evaluating** actions vs. **instructing** by giving correct actions
- Pure evaluative feedback depends totally on the action taken. Pure instructive feedback depends not at all on the action taken.
- Supervised learning is instructive; optimization is evaluative
- **Associative** vs. **Nonassociative**:
 - Associative: inputs mapped to outputs; learn the best output **for each** input
 - Nonassociative: “learn” (find) one best output
- n -armed bandit (at least how we treat it) is:
 - Nonassociative
 - Evaluative feedback

The n -Armed Bandit Problem

- 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 then **exploit** the best of them.

The Exploration/Exploitation Dilemma

- Suppose you form estimates

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

- The **greedy** action at t is

$$a_t^* = \operatorname{argmax}_a Q_t(a)$$

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

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

- 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

Action-Value Methods

- Methods that adapt action-value estimates and nothing else, e.g.: suppose by the t -th play, action a had been chosen k_a times, producing rewards r_1, r_2, \dots, r_{k_a} , then

$$Q_t(a) = \frac{r_1 + r_2 + \dots + r_{k_a}}{k_a} \quad \text{“sample average”}$$

$$\lim_{k_a \rightarrow \infty} Q_t(a) = Q^*(a)$$

ϵ -Greedy Action Selection

- Greedy action selection:

$$a_t = a_t^* = \underset{a}{\operatorname{arg\,max}} Q_t(a)$$

- ϵ -Greedy:

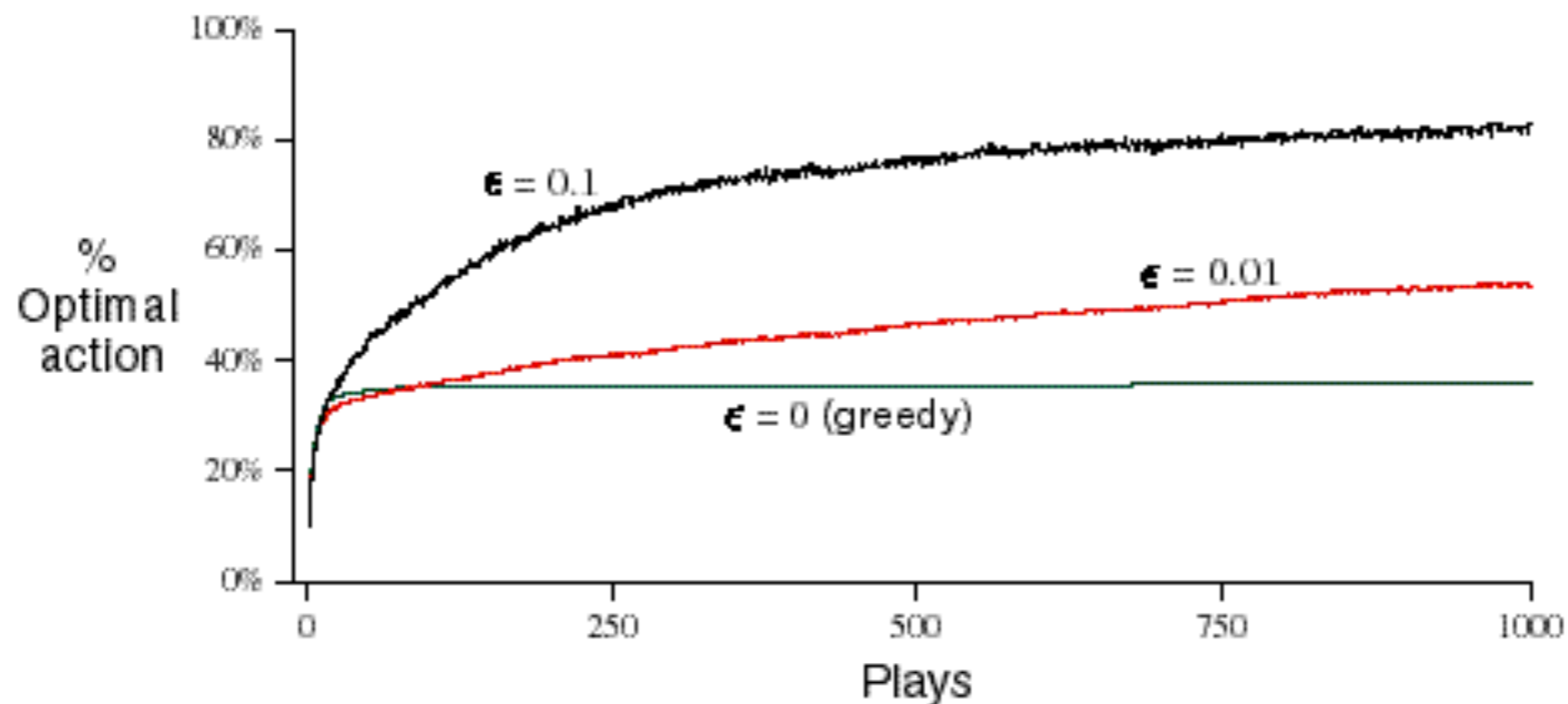
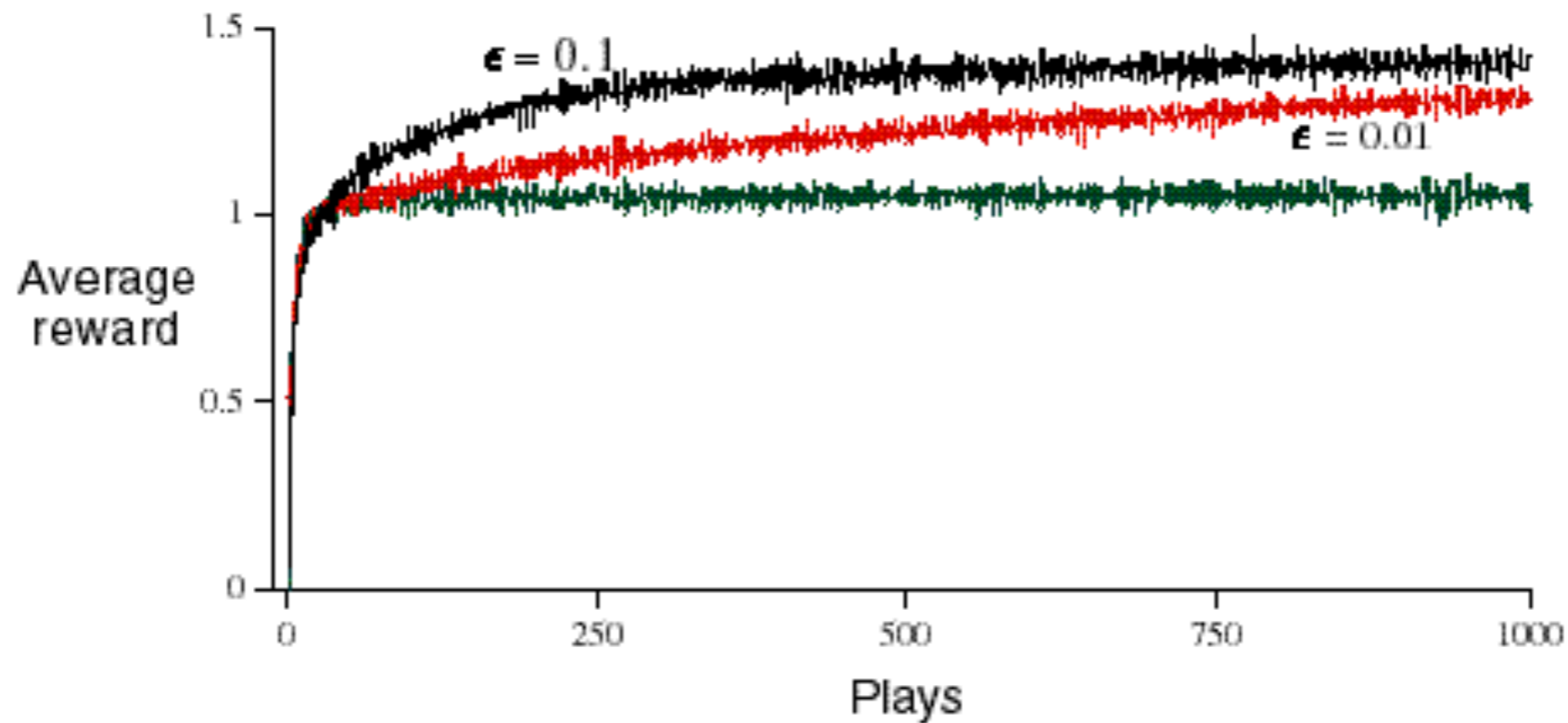
$$a_t = \begin{cases} a_t^* & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

... the simplest way to try to balance exploration and exploitation

10-Armed Testbed

- $n = 10$ possible actions
- Each $Q^*(a)$ is chosen randomly from a normal distribution: $N(0,1)$
- each r_t is also normal: $N(Q^*(a_t),1)$
- 1000 plays
- repeat the whole thing 2000 times (with reselecting $Q^*(a)$ and average the results)
- *Evaluative versus instructive feedback*

ϵ -Greedy Methods on the 10-Armed Testbed



Softmax Action Selection

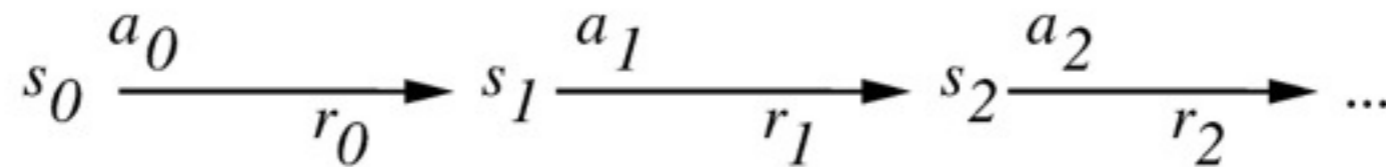
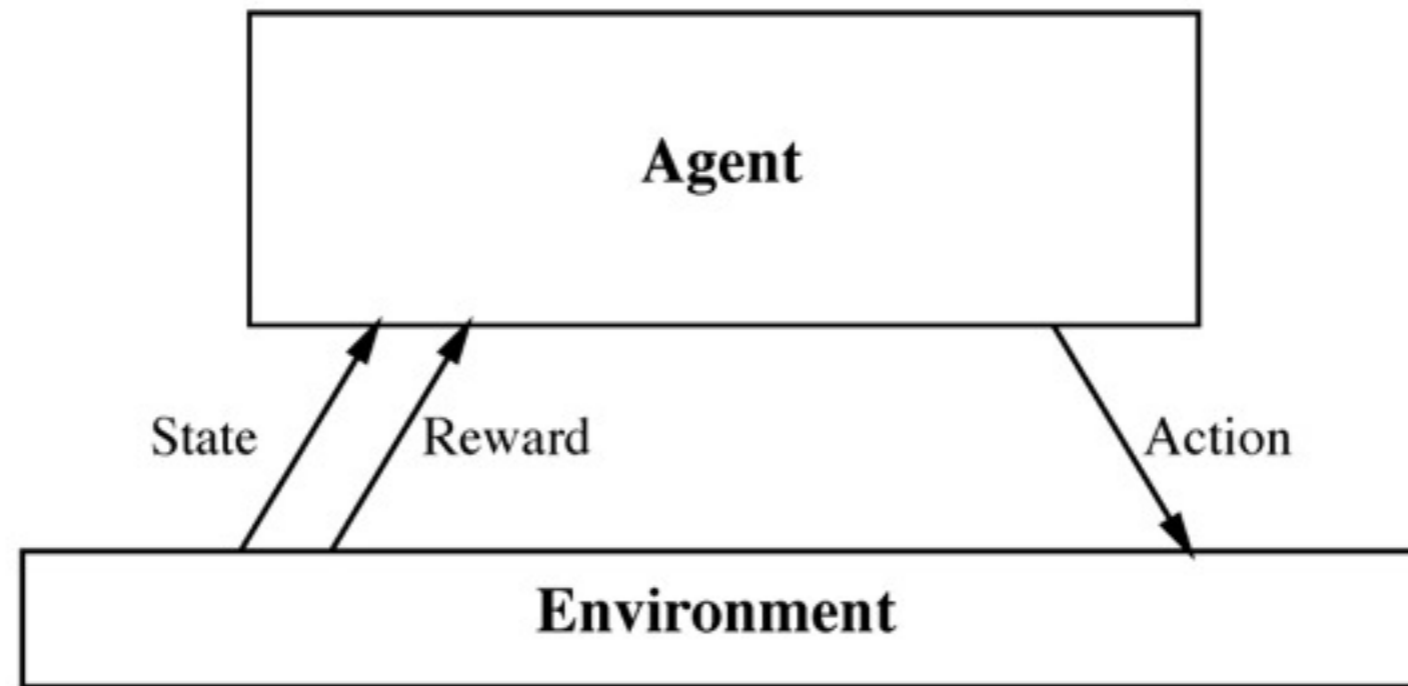
- Softmax action selection methods grade action probs. by estimated values.
- The most common softmax uses a Gibbs, or Boltzmann, distribution:

Choose action a on play t with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^n e^{Q_t(b)/\tau}},$$

where τ is the “computational temperature”

Reinforcement Learning Problem



Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \text{ where } 0 \leq \gamma < 1$$

Markov Decision Processes

Assume

- finite set of states S
- set of actions A
- at each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$
- then receives immediate reward r_t
- and state changes to s_{t+1}
- Markov assumption: $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
 - i.e., r_t and s_{t+1} depend only on *current* state and action
 - functions δ and r may be nondeterministic
 - functions δ and r not necessarily known to agent

Value Function

To begin, consider deterministic worlds...

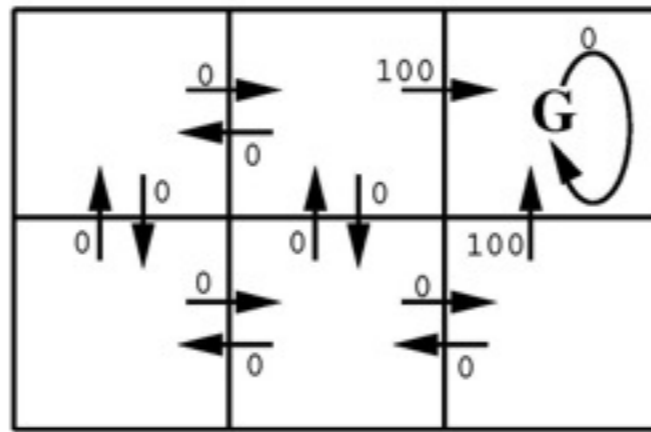
For each possible policy π the agent might adopt, we can define an evaluation function over states

$$\begin{aligned} V^\pi(s) &\equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

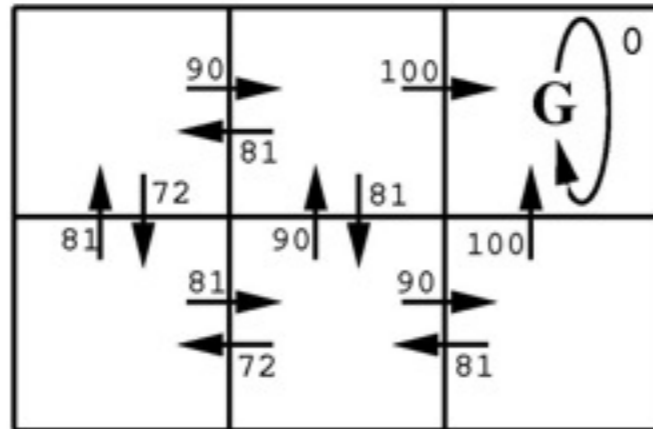
where r_t, r_{t+1}, \dots are generated by following policy π starting at state s

Restated, the task is to learn the optimal policy π^*

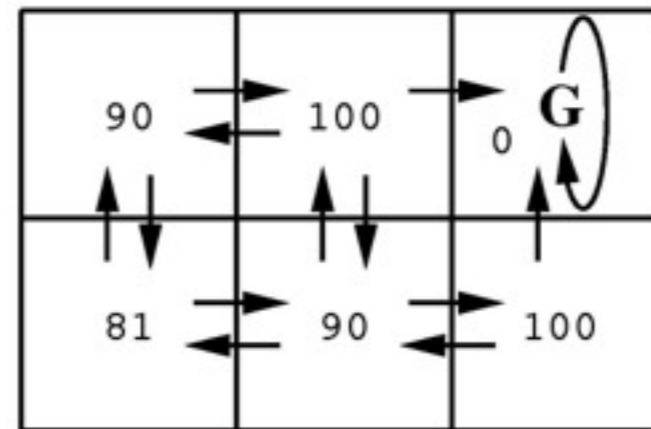
$$\pi^* \equiv \operatorname{argmax}_{\pi} V^\pi(s), (\forall s)$$



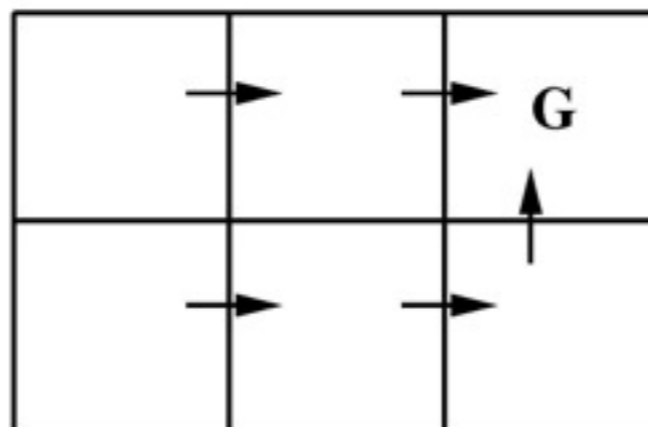
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy

What to Learn

We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*)

It could then do a lookahead search to choose best action from any state s because

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

A problem:

- This works well if agent knows $\delta : S \times A \rightarrow S$, and $r : S \times A \rightarrow \mathfrak{R}$
- But when it doesn't, it can't choose actions this way

Q Function

Define new function very similar to V^*

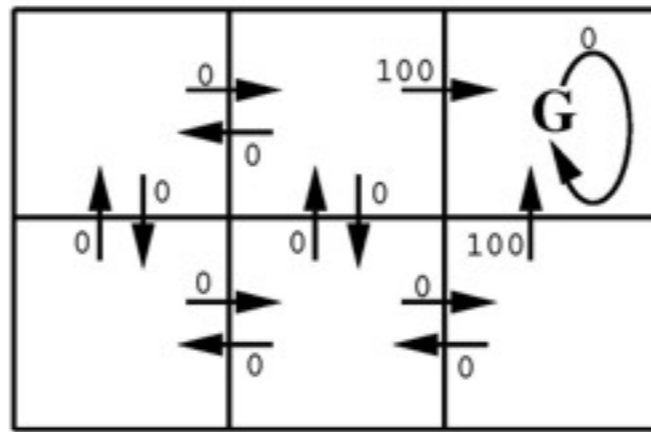
$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

If agent learns Q , it can choose optimal action even without knowing δ !

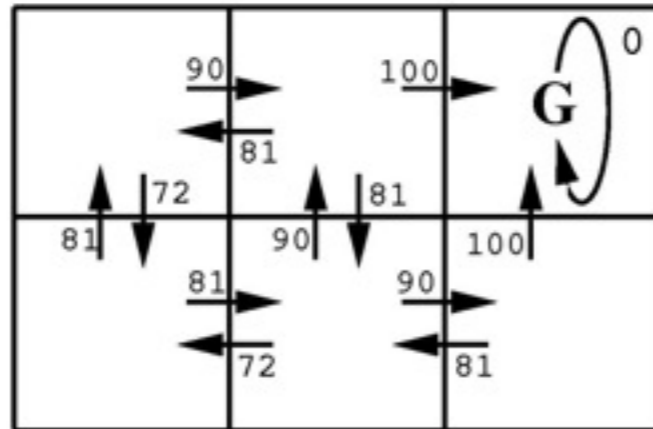
$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

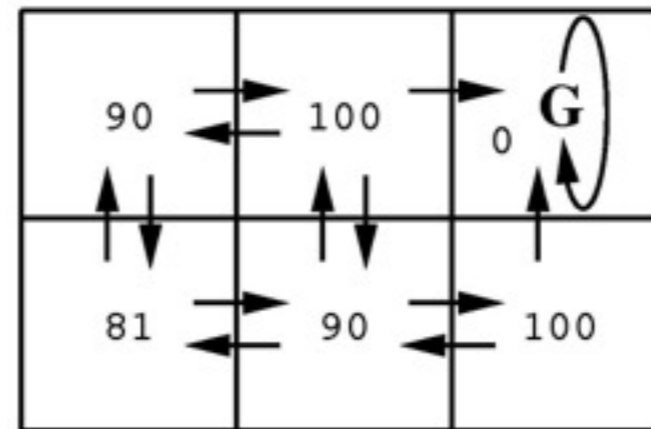
Q is the evaluation function the agent will learn



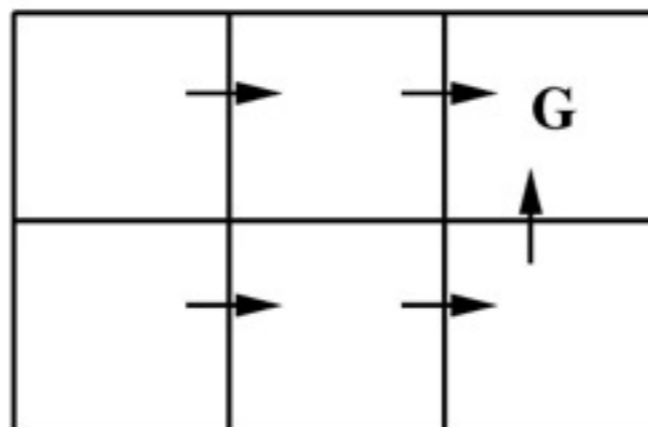
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy

Training Rule to Learn Q

Note Q and V^* closely related:

$$V^*(s) = \max_{a'} Q(s, a')$$

Which allows us to write Q recursively as

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

Nice! Let \hat{Q} denote learner's current approximation to Q . Consider training rule

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

where s' is the state resulting from applying action a in state s

Q Learning for Deterministic Worlds

For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state s

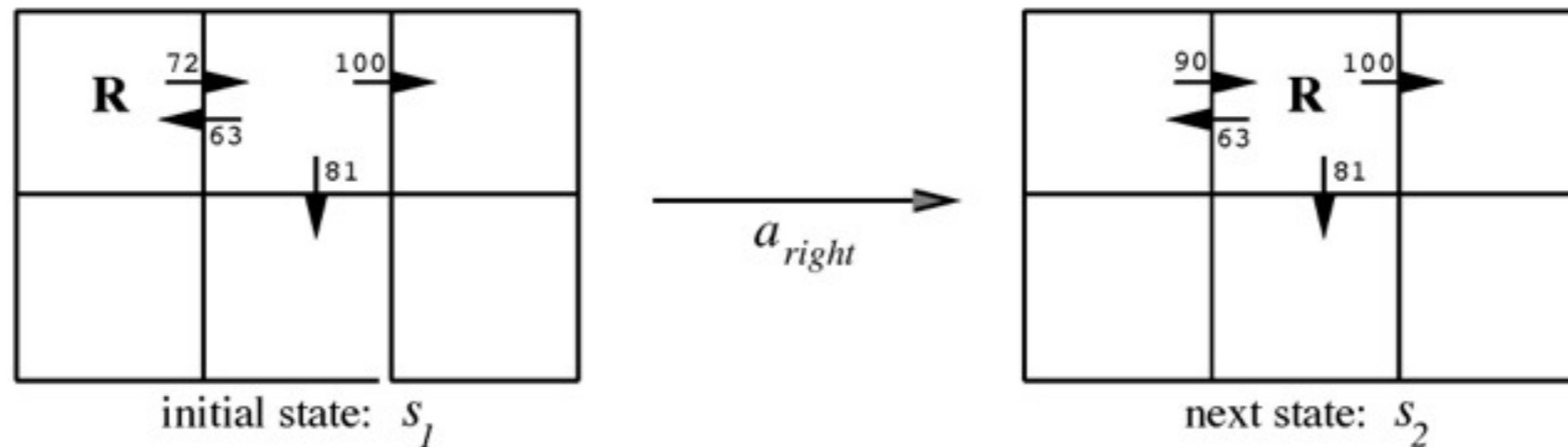
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

Updating \hat{Q}



$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{63, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

notice if rewards non-negative, then

$$(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

and

$$(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

Nondeterministic Case

What if reward and next state are non-deterministic?

We redefine V, Q by taking expected values

$$\begin{aligned} V^\pi(s) &\equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots] \\ &\equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right] \end{aligned}$$

$$Q(s, a) \equiv E[r(s, a) + \gamma V^*(\delta(s, a))]$$

Nondeterministic Case

Q learning generalizes to nondeterministic worlds

Alter training rule to

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n[r + \max_{a'} \hat{Q}_{n-1}(s', a')]$$

where

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$

Can still prove convergence of \hat{Q} to Q [Watkins and Dayan, 1992]

Temporal Difference Learning

Q learning: reduce discrepancy between successive Q estimates

One step time difference:

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

Why not two steps?

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

Or n ?

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \dots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

Blend all of these:

$$Q^\lambda(s_t, a_t) \equiv (1-\lambda) [Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \dots]$$

Temporal Difference Learning

$$Q^\lambda(s_t, a_t) \equiv (1-\lambda) [Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \dots]$$

Equivalent expression:

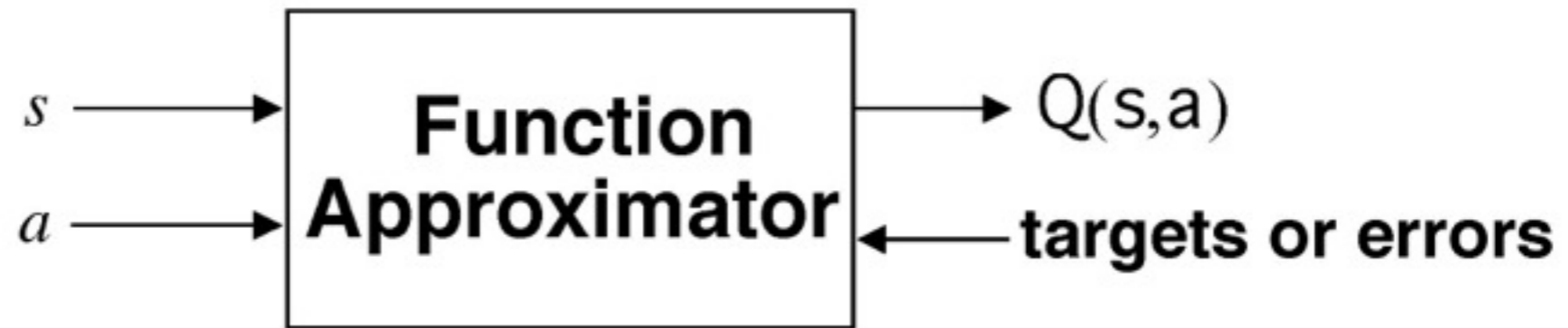
$$Q^\lambda(s_t, a_t) = r_t + \gamma [(1 - \lambda) \max_a \hat{Q}(s_t, a) + \lambda Q^\lambda(s_{t+1}, a_{t+1})]$$

TD(λ) algorithm uses above training rule

- Sometimes converges faster than Q learning
- converges for learning V^* for any $0 \leq \lambda \leq 1$ (Dayan, 1992)
- Tesauro's TD-Gammon uses this algorithm

Function Approximation and Reinforcement Learning

General Idea



Could be:

- table
 - ⇒ • Backprop Neural Network
 - ⇒ • Radial-Basis-Function Network
 - Tile Coding (CMAC)
 - Nearest Neighbor, Memory Based
 - Decision Tree
- } gradient-descent methods

Neural Networks as FAs

$$Q(s, a) = f(s, a, w)$$

weight vector

standard
backprop
gradient

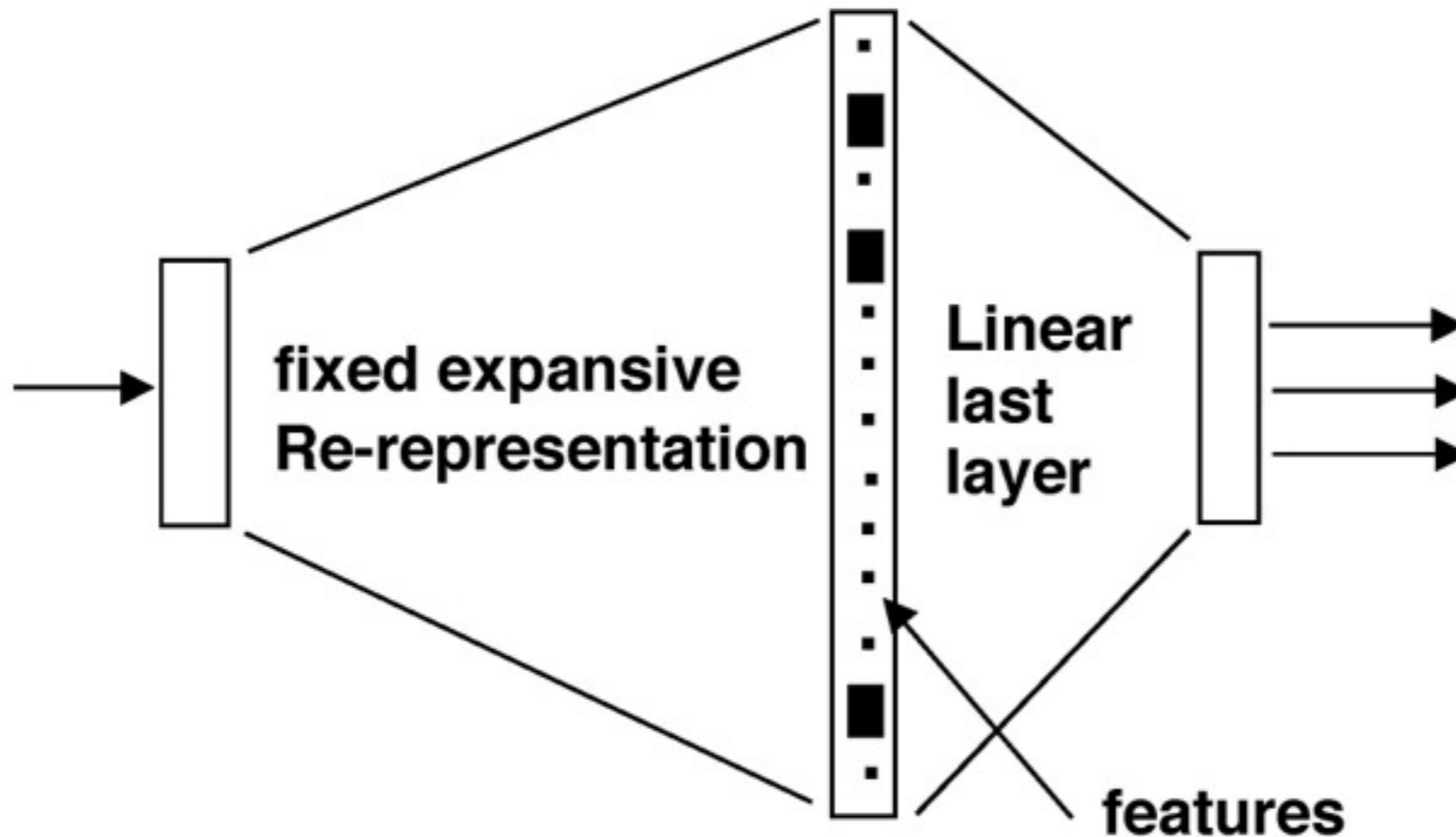
e.g., gradient-descent Sarsa:

$$w \leftarrow w + \alpha \left[\underbrace{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})}_{\text{target value}} - \underbrace{Q(s_t, a_t)}_{\text{estimated value}} \right] \nabla_w f(s_t, a_t, w)$$

target value

estimated value

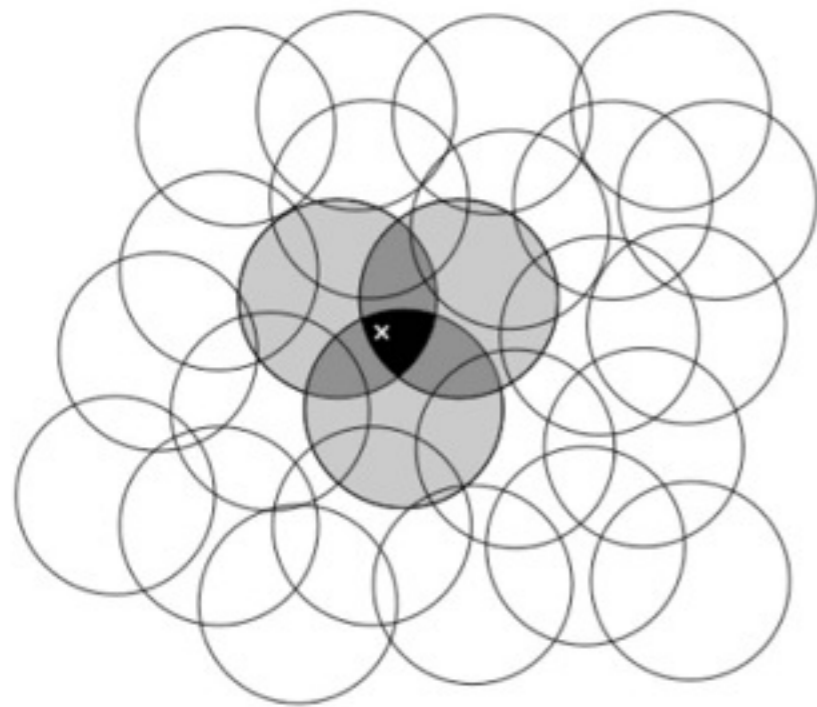
Sparse Coarse Coding



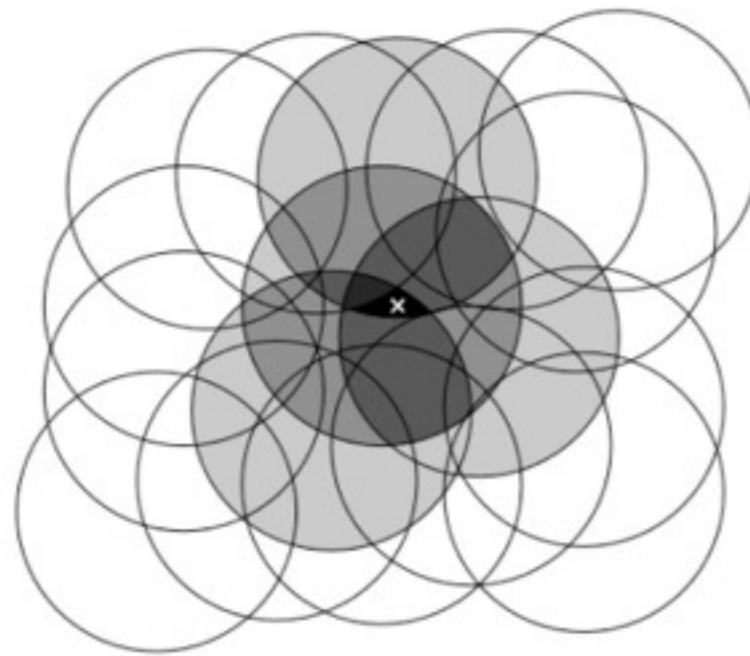
Coarse: Large receptive fields

Sparse: Few features present at one time

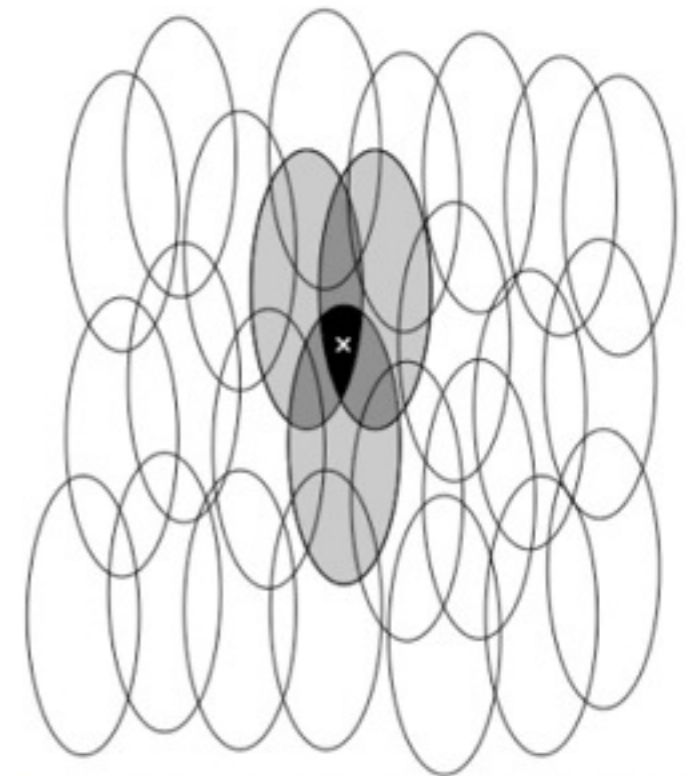
Shaping *Generalization* in *Coarse Coding*



a) Narrow generalization

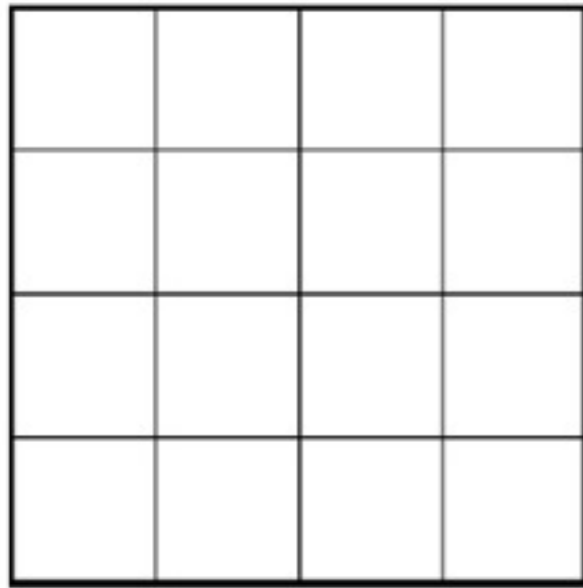


b) Broad generalization

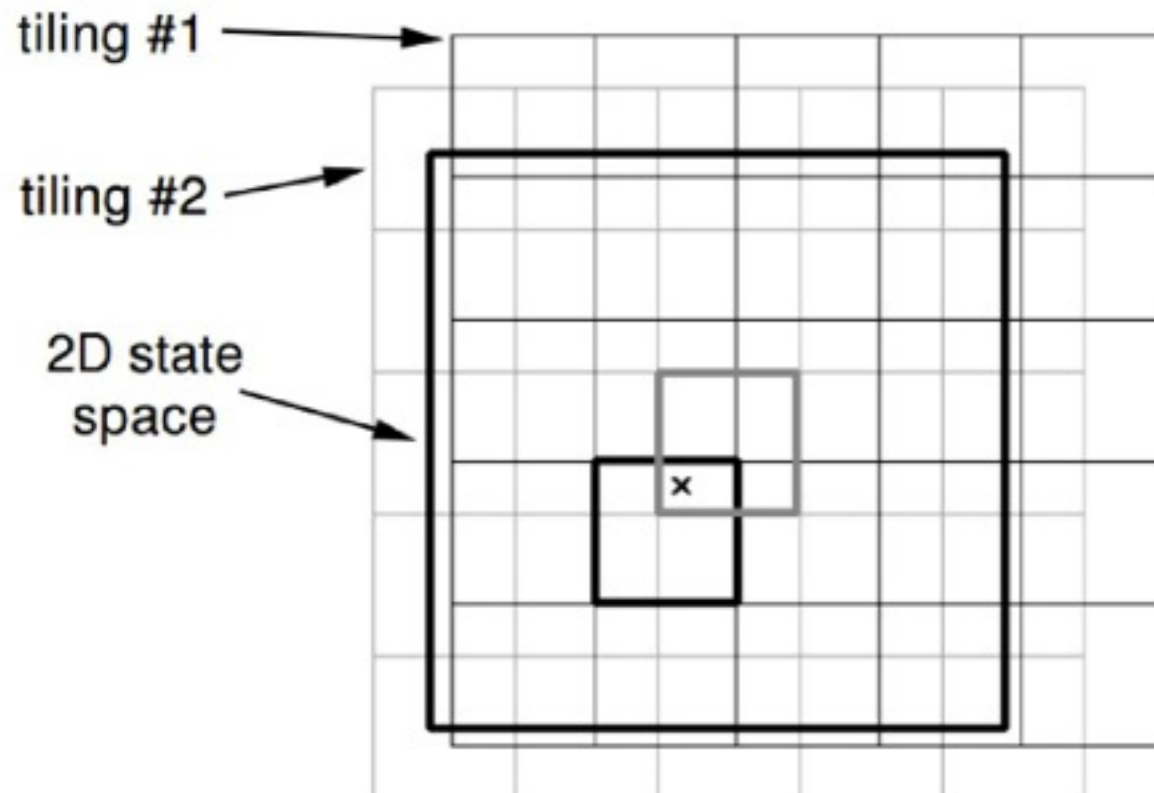


c) Asymmetric generalization

Tile Coding



- Binary feature for each tile
- Number of features present at any one time is constant
- Binary features means weighted sum easy to

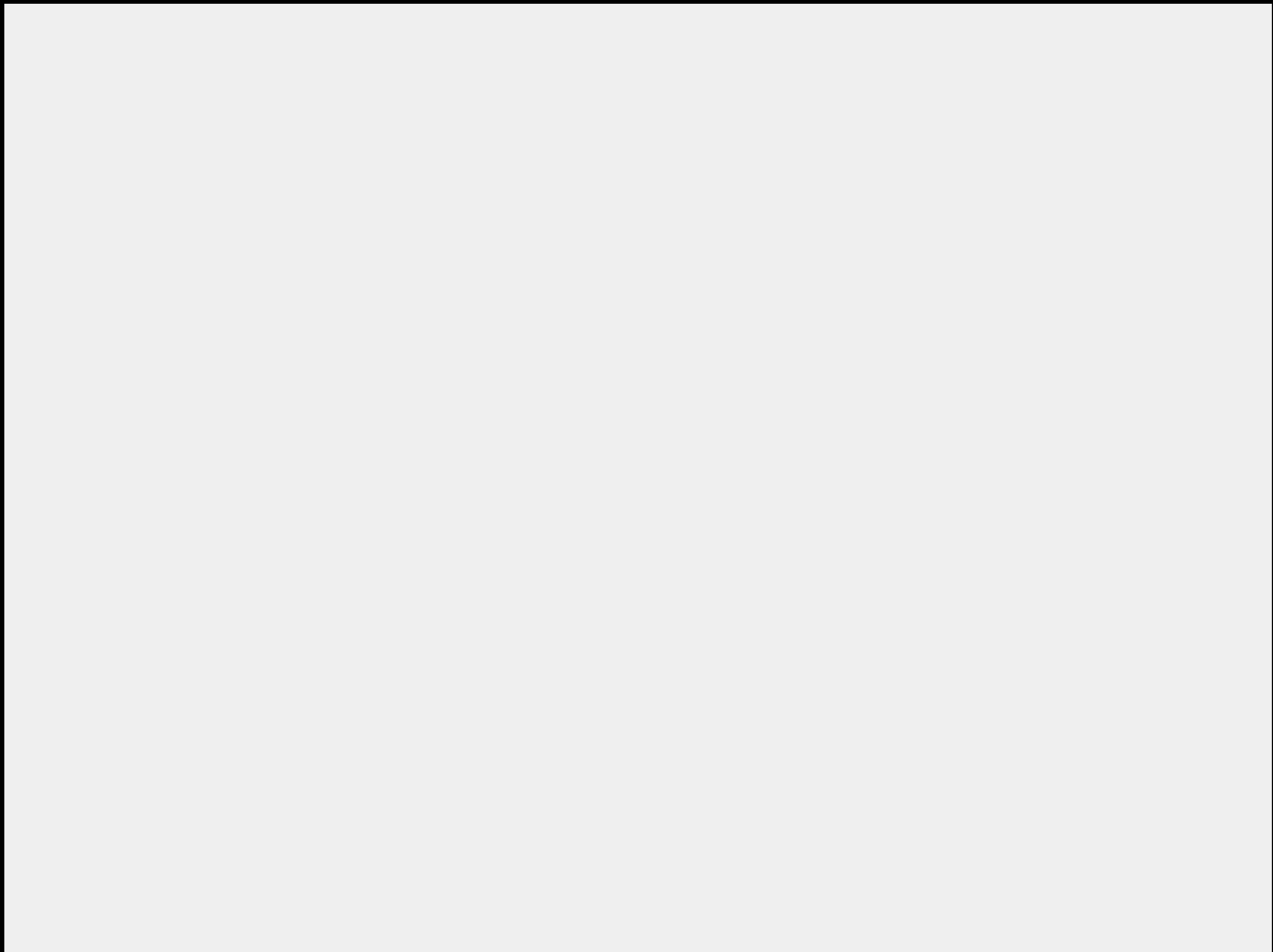


Shape of tiles \Rightarrow Generalization

#Tilings \Rightarrow Resolution of final approximation

FAs & RL

- Linear FA (divergence can happen)
Nonlinear Neural Networks (theory is not well developed)
Non-parametric, e.g., nearest-neighbor (provably not divergent; bounds on error)
Everyone uses their favorite FA... little theoretical guidance yet!
- Does FA really beat the curse of dimensionality?
 - Probably; with FA, computation seems to scale with the complexity of the solution (crinkliness of the value function) and how hard it is to find it
- Empirically it works
 - though many folks have a hard time making it so
 - no off-the-shelf FA+RL yet



Learning for control from multiple demonstrations

Andrew Ng (Stanford U)

We consider the problem of learning to follow a desired trajectory when given a small number of demonstrations from a sub-optimal expert. We present an algorithm that (i) extracts the desired trajectory from the sub-optimal expert's demonstrations and (ii) learns a local model suitable for control along the learned trajectory. We apply our algorithm to the problem of autonomous helicopter flight. In all cases, the autonomous helicopter's performance exceeds that of our expert helicopter pilot's demonstrations.

Learning with PyBrain