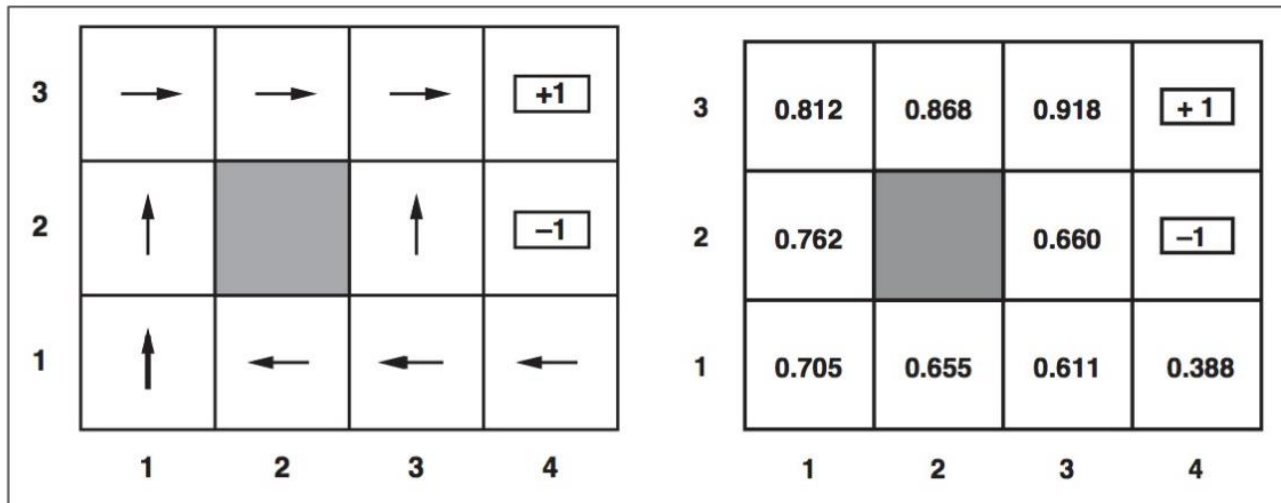


# 21 REINFORCEMENT LEARNING

*In which we examine how an agent can learn from success and failure, from reward and punishment.*



$(1, 1) \xrightarrow{.04} (1, 2) \xrightarrow{.04} (1, 3) \xrightarrow{.04} (1, 2) \xrightarrow{.04} (1, 3) \xrightarrow{.04} (2, 3) \xrightarrow{.04} (3, 3) \xrightarrow{.04} (4, 3) +1$   
 $(1, 1) \xrightarrow{.04} (1, 2) \xrightarrow{.04} (1, 3) \xrightarrow{.04} (2, 3) \xrightarrow{.04} (3, 3) \xrightarrow{.04} (3, 2) \xrightarrow{.04} (3, 3) \xrightarrow{.04} (4, 3) +1$   
 $(1, 1) \xrightarrow{.04} (2, 1) \xrightarrow{.04} (3, 1) \xrightarrow{.04} (3, 2) \xrightarrow{.04} (4, 2) -1$  .

# CS 188: Artificial Intelligence

## Reinforcement Learning

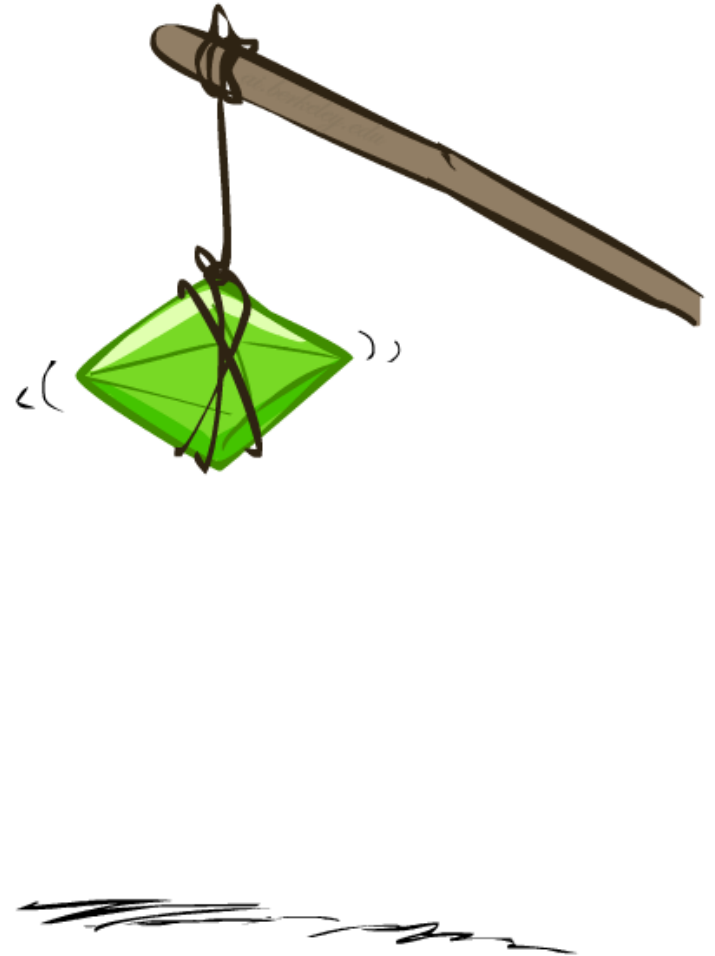
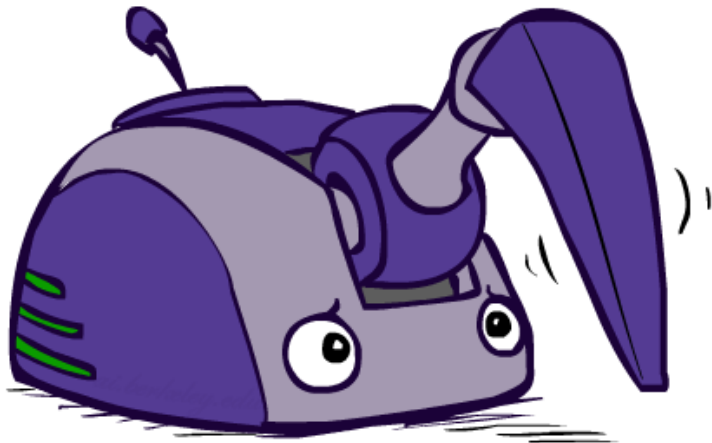


Instructors: Dan Klein and Pieter Abbeel

University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.]

# Reinforcement Learning



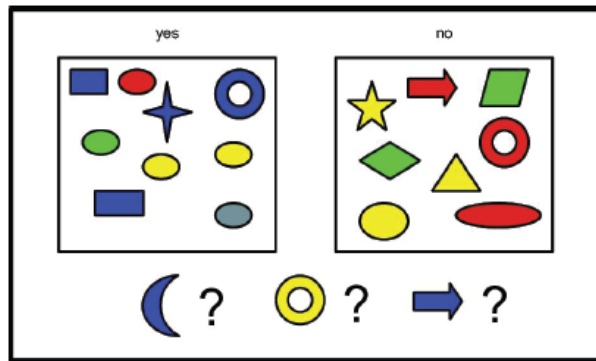
# First Look at Learning

- Common forms of Machine Learning
  - Supervised
  - Unsupervised

# Supervised Learning

## 1.2. Supervised learning

3



(a)

D features (attributes)			Label
Color	Shape	Size (cm)	
Blue	Square	10	1
Red	Ellipse	2.4	1
Red	Ellipse	20.7	0

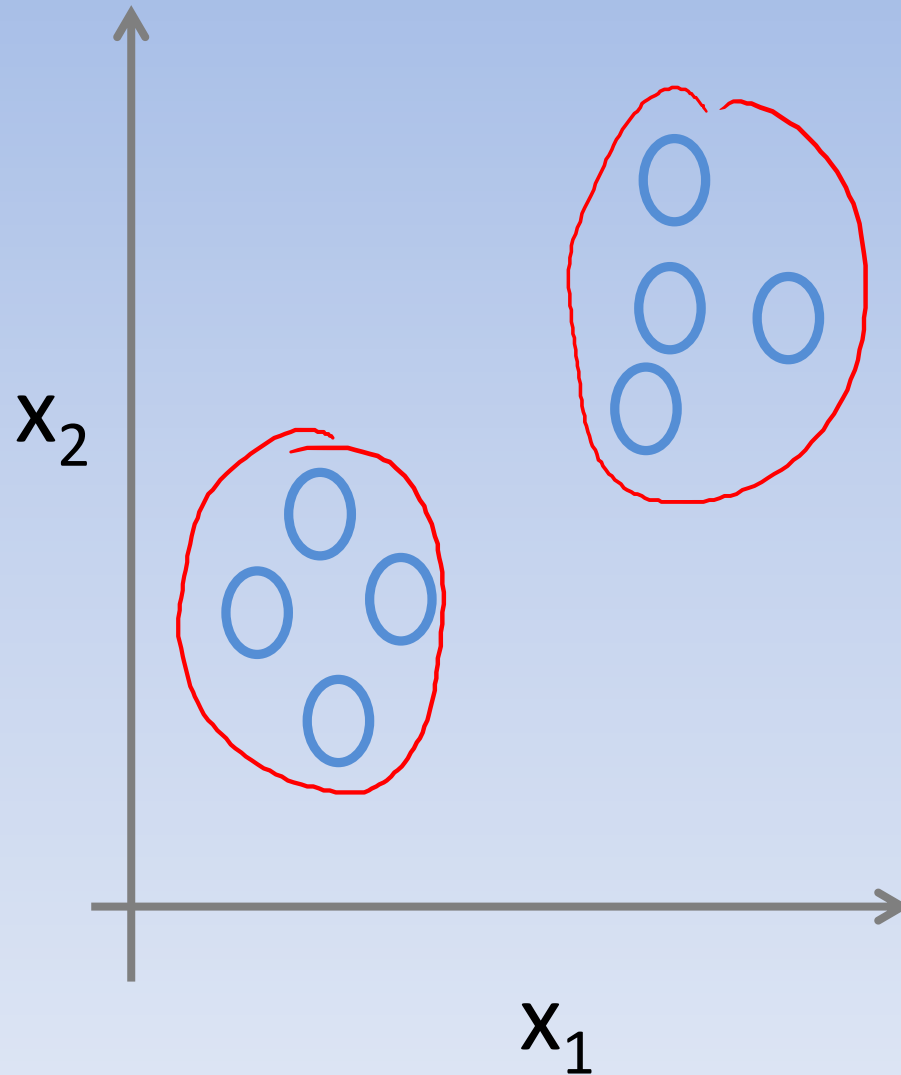
(b)

- Learn a function from examples
- $f$  is the target function
- An example is an input-output pair:  $(x, f(x))$
- Problem:
  - Given a hypothesis space  $H$
  - Given a training set of examples:  $(x_1, f(x_1)), \dots, (x_n, f(x_n))$
  - Find a hypothesis  $h(x)$  such that  $h \sim f$

# Unsupervised Learning

- Data is no longer labeled!
- Learner gets No Feedback.
  - Descriptive
  - Tries to organize its data
- Clustering

# Unsupervised Learning



# Reinforcement Learning

- Now lets look at learning a game (Chess).
- Supervised learning requires labeled examples:
  - i.e., large sample of moves each one labeled as good or bad.
- These sample sets seldom available.
- Still, some kind of feedback is needed to learn!



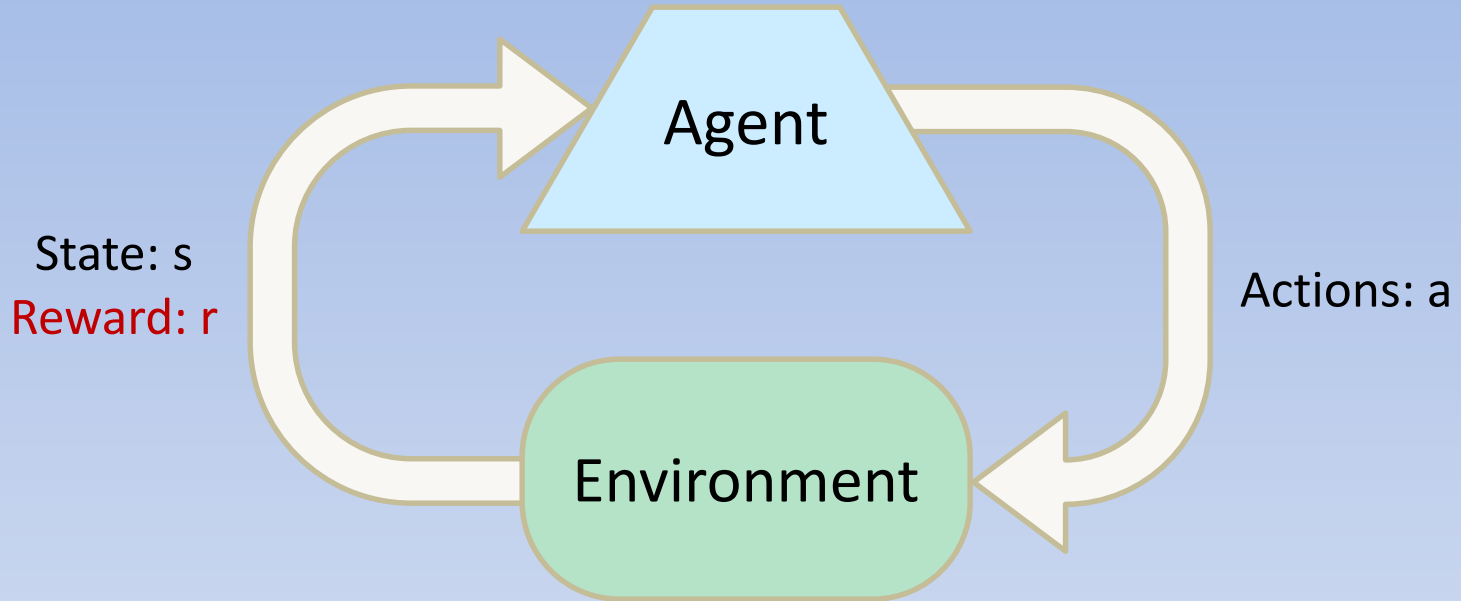
# Rewards/Reinforcements

- Now we learn where Reinforcements arise.
- Agent must know when something good or something bad has happened!
  - Rewards
    - + 20
    - - 200
  - Reinforcements
    - Good Job!
    - More work is needed.

# Rewards/Reinforcements

- Reinforcements come in the end with games like chess.
- Other environments get feedback more frequently
  - Ping-Pong
  - Learning to crawl
- Reinforcements are a percept, but hardwired differently.
  - Animals perceive hunger/pain differently than vision/auditory percepts.
- Reinforcement Learning studied by psychologists for more than 60 years.

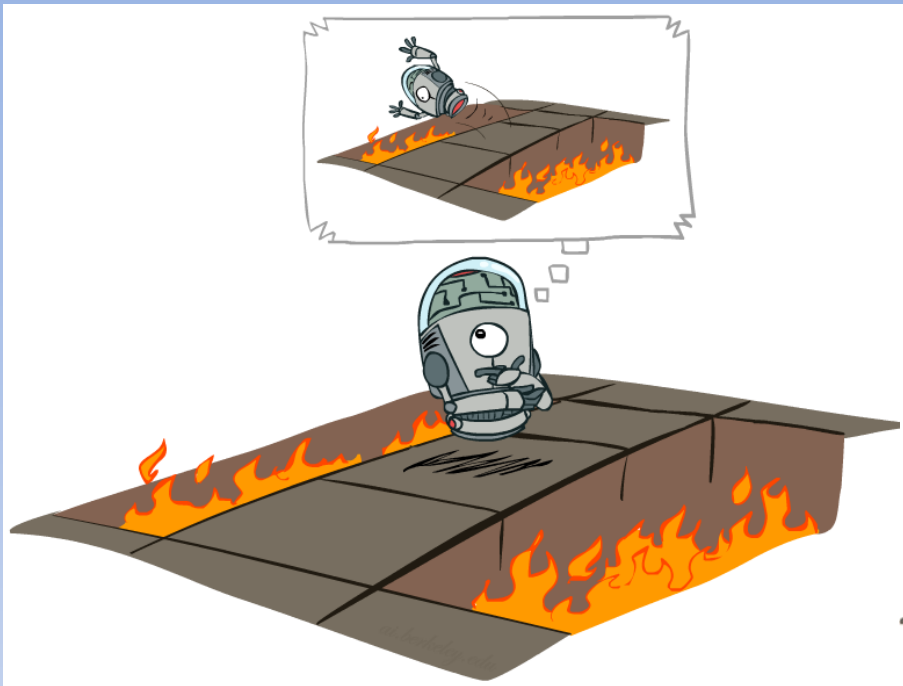
# Reinforcement Learning



- Receive feedback in the form of **rewards**
- Agent's utility is defined by the reward function
- Must (learn to) act so as to **maximize expected rewards**
- All learning is based on observed samples of outcomes!

# Model

- Assume a Markov decision process (MDP):
  - A set of states  $s \in S$
  - A set of actions (per state)  $A$
  - A model  $T(s,a,s') \rightarrow P(s' | s,a)$
  - A reward function  $R(s,a,s')$  – Sometimes  $R(s)$
- Still looking for a policy  $\pi(s)$
- Assume now: we don't know  $T$  or  $R$ 
  - I.e. we don't know which states are good or what the actions do
  - Learning requires actually trying actions and states out



## Offline Solution

- When MDP Known – Solution found offline!
- Without MDP ( $T$ ,  $R$ ), need to take actions to learn!



## Online Learning

# Reinforcement Learning

- What should we learn?
- First thought, what do we not know?
  - What are our unknowns?
- Unknowns:
  - $T(s, a, s')$  or  $P(s' | s, a)$
  - $R(s, a, s')$  or  $R(s)$
- Let's Learn T and R!

# Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct
- Step 1: Learn empirical MDP model
  - Count outcomes  $s'$  for each  $s, a$
  - Normalize to give an estimate of  $\hat{T}(s, a, s')$
  - Discover each  $\hat{R}(s, a, s')$  when we experience  $(s, a, s')$
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

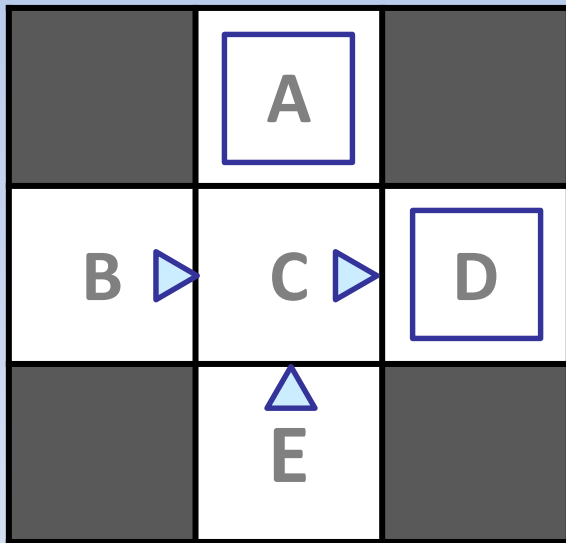
# Passive Learning Agent

- State-based Representation
- Fully observable environment
- Fixed policy  $\pi$
- In state  $s$  agent always executes action  $\pi(s)$



# Example

Input Policy  $\pi$



Assume:  $\gamma = 1$

- Let's put the agent in environment and see what happens!

# Agent Acting

## Observed Episodes (Training)

### Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

- $\hat{R}(s, a, s')$ 
  - (B, east, C) = -1
  - (C, east, D) = -1
  - (D, exit, x) = +10
- $\hat{T}(s, a, s')$ 
  - (B, east, C) = 1.0/1.0
  - (C, east, D) = 1.0/1.0
  - (D, exit, x) = 1.0/1.0

# Agent Acting

## Observed Episodes (Training)

### Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

- $\hat{R}(s, a, s')$ 
  - (B, east, C) = -1
  - (C, east, D) = -1
  - (D, exit, x) = +10
- $\hat{T}(s, a, s')$ 
  - (B, east, C) = 2.0/2.0
  - (C, east, D) = 2.0/2.0
  - (D, exit, x) = 2.0/2.0

# Agent Acting

## Observed Episodes (Training)

### Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

- $\hat{R}(s, a, s')$ 
  - (B, east, C) = -1
  - (C, east, D) = -1
  - (D, exit, x) = +10
  - (E, north, C) = -1
- $\hat{T}(s, a, s')$ 
  - (B, east, C) = 2.0/2.0
  - (C, east, D) = 3.0/3.0
  - (D, exit, x) = 3.0/3.0
  - (E, north, C) = 1.0/1.0

# Agent Acting

## Observed Episodes (Training)

### Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 4

E, north, C, -1  
**C, east, A, -1**  
A, exit, x, -10

- $\hat{R}(s, a, s')$ 
  - (A, exit, x) = -10
  - (B, east, C) = -1
  - (C, east, D) = -1
  - **(C, east, A) = -1.0**
  - (D, exit, x) = +10
  - (E, north, C) = -1
- $\hat{T}(s, a, s')$ 
  - (A, exit, x) = 1.0/1.0
  - (B, east, C) = 2.0/2.0
  - **(C, east, D) = 3.0/4.0**
  - **(C, east, A) = 1.0/4.0**
  - (D, exit, x) = 3.0/3.0
  - (E, north, C) = 2.0/2.0

# Now Know T, R

- $\hat{R}(s, a, s')$ 
  - (A, exit, x) = -10
  - (B, east, C) = -1
  - (C, east, D) = -1
  - (C, east, A) = -1.0
  - (D, exit, x) = +10
  - (E, north, C) = -1
- $\hat{R}(s, a, s')$ 
  - (A, exit, x) = 1.0/1.0 = 1.0
  - (B, east, C) = 2.0/2.0 = 1.0
  - (C, east, D) = 3.0/4.0 = 0.75
  - (C, east, A) = 1.0/4.0 = 0.25
  - (D, exit, x) = 3.0/3.0 = 1.0
  - (E, north, C) = 2.0/2.0 = 1.0
- We can use Policy Evaluation to Calculate utility of states.

```

function PASSIVE-ADP-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal  $r'$ 
  persistent:  $\pi$ , a fixed policy
               mdp, an MDP with model  $P$ , rewards  $R$ , discount  $\gamma$ 
                $U$ , a table of utilities, initially empty
                $N_{sa}$ , a table of frequencies for state–action pairs, initially zero
                $N_{s'|sa}$ , a table of outcome frequencies given state–action pairs, initially zero
                $s, a$ , the previous state and action, initially null

  if  $s'$  is new then  $U[s'] \leftarrow r'$ ;  $R[s'] \leftarrow r'$ 
  if  $s$  is not null then
    increment  $N_{sa}[s, a]$  and  $N_{s'|sa}[s', s, a]$ 
    for each  $t$  such that  $N_{s'|sa}[t, s, a]$  is nonzero do
       $P(t | s, a) \leftarrow N_{s'|sa}[t, s, a] / N_{sa}[s, a]$ 
     $U \leftarrow \text{POLICY-EVALUATION}(\pi, U, \text{mdp})$ 
  if  $s'.\text{TERMINAL?}$  then  $s, a \leftarrow \text{null}$  else  $s, a \leftarrow s', \pi[s']$ 
  return  $a$ 

```

**Figure 21.2** A passive reinforcement learning agent based on adaptive dynamic programming. The POLICY-EVALUATION function solves the fixed-policy Bellman equations, as described on page 657.

# Can we do better?

- Effort expended learning model, while model is not our goal.
- Frequently better to learn values closest to what's needed.
- Still, what then should we learn??



# Example: Expected Age

Goal: Compute expected age of cs166 students

Known  $P(A)$

$$E[A] = \sum_a P(a) \cdot a = 0.35 \times 20 + \dots$$

Without  $P(A)$ , instead collect samples  $[a_1, a_2, \dots, a_N]$

Unknown  $P(A)$ : “Model Based”

Why does this work?  
Because eventually you learn the right model.

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$
$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Unknown  $P(A)$ : “Model Free”

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because samples appear with the right frequencies.

# Model-Free Learning

- Learn the utility of states directly!
- Why worry about MDP!
  - Sounds good.
  - Don't worry, we'll get back to MDP.



# Passive Reinforcement Learning

- Passive Reinforcement Learning:
  - State-based Representation
  - Fully observable environment
  - Fixed policy  $\pi$ 
    - In state  $s$  agent always executes action  $\pi(s)$
- What should we learn?
  - First thought, what do we not know?
  - What are our unknowns?
    - Unknowns:
      - $T(s, a, s')$  or  $P(s' | s, a)$
      - $R(s, a, s')$  or  $R(s)$
  - This time.. Let's learn utility of states directly.
- Passive Reinforcement Learning:
  - Learner is “along for the ride”
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

# Model-Free Learning

- Late 1950's Widrow & Hoff in adaptive control theory invented Direct Utility Estimation
- Key: Utility of a state is the expected total reward from that state forward.
  - Reward-to-go
- Each episode provides a sample of this quantity.
  - Average all episodes



# Bernard Widrow

Professor Emeritus  
Electrical Engineering Department  
Stanford University

## Research

Prof. Widrow's research focuses on adaptive signal processing, adaptive control systems, adaptive neural networks, human memory, and human-like memory for computers. Applications include signal processing, prediction, noise cancelling, adaptive arrays, control systems, and pattern recognition.

---

## Courses Taught

EE373A **Adaptive Signal Processing**

EE373B **Adaptive Neural Networks**

EE375 **Quantization Noise**

---

## Short CV

## Patents

## Publications

---

## Hearing Aid Device

A directional acoustic receiving system is constructed in the form of a



necklace, including an array of two or more microphones mounted on a housing supported on the chest of the user by a conducting loop encircling the user's neck. This method enables the design of highly-directional hearing instruments which are comfortable, inconspicuous, and convenient to use. The array provides the user with a dramatic improvement in speech perception over existing

hearing aid designs, particularly in the presence of background noise, reverberation, and feedback.

B. Widrow, "A Microphone Array for Hearing Aids," *IEEE Circuits and Systems Magazine*, 1(2):26-32, 2001.

## Quantization Noise

Prof. Widrow's most recent book, *Quantization Noise*, co-authored with Istvan Kollar, is available for purchase at the Cambridge University Press website. Click on the book at left to browse through it or to purchase a copy.



# Mathematics Genealogy Project

Home

Search

Extrema

About MGP ▶

Links

FAQs

Posters

Submit Data

Contact

Mirrors ▶

Donate

## Bernard Widrow

Sc.D. [Massachusetts Institute of Technology](#) 1956



Dissertation: *A Study of Rough Amplitude Quantization by Means of Nyquist Sampling Theory*

Mathematics Subject Classification: 94—Information and communication, circuits

Advisor: [William Kirby Linvill](#)

Students:

Click [here](#) to see the students ordered by family name.

Name	School	Year	Descendants
<a href="#">Mattson, Richard</a>	Stanford University	1962	371
<a href="#">Smith, Alvy</a>	Stanford University	1970	
<a href="#">Larimore, Michael</a>	Stanford University	1977	
<a href="#">Treichler, John</a>	Stanford University	1977	
<a href="#">Stevenson, Maryhelen</a>	Stanford University	1991	
<a href="#">Wan, Eric</a>	Stanford University	1994	2
<a href="#">Aarabi, Parham</a>	Stanford University	2001	4
<a href="#">Prieto, Ramon</a>	Stanford University	2003	
<a href="#">Flores, Aaron</a>	Stanford University	2006	
<a href="#">Nejati, Negin</a>	Stanford University	2011	

A service of the [NDSU Department of Mathematics](#), in association with the [American Mathematical Society](#).





# Mathematics Genealogy Project

## Richard Lewis Mattson

[MathSciNet](#)

Ph.D. [Stanford University](#) 1962



Dissertation: *The Analysis and Synthesis of Adaptive Systems Which Use Networks of Threshold Elements*

Advisor: [Bernard Widrow](#)

Students:

Click [here](#) to see the students listed in chronological order.

Name	School	Year	Descendants
<a href="#">Hopcroft, John</a>	Stanford University	1964	296
<a href="#">Patt, Yale</a>	Stanford University	1966	73

According to our current on-line database, Richard Mattson has 2 [students](#) and 371 [descendants](#).

We welcome any additional information.

If you have additional information or corrections regarding this mathematician, please use the [update form](#). To submit students of this mathematician, please use the [new data form](#), noting this mathematician's MGP ID of 91422 for the advisor ID.

[Home](#)

[Search](#)

[Extrema](#)

[About MGP](#)

[Links](#)

[FAQs](#)

[Posters](#)

[Submit Data](#)

[Contact](#)

[Mirrors](#)

[Donate](#)

A service of the [NDSU Department of Mathematics](#), in association with the [American Mathematical Society](#).

# INTRODUCTION TO AUTOMATA THEORY, LANGUAGES, AND COMPUTATION

JOHN E. HOPCROFT  
JEFFREY D. ULLMAN



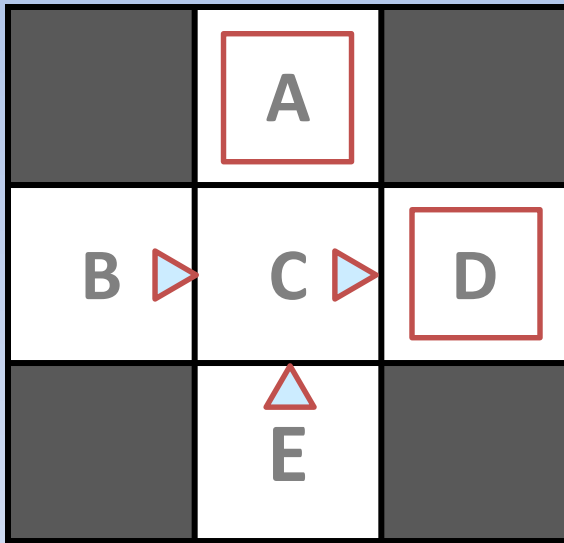


# Direct-Evaluation

- Goal: Compute values for each state under  $\pi$
- Idea: Average together observed sample values
  - Act according to  $\pi$
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

# Example: Direct Evaluation

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

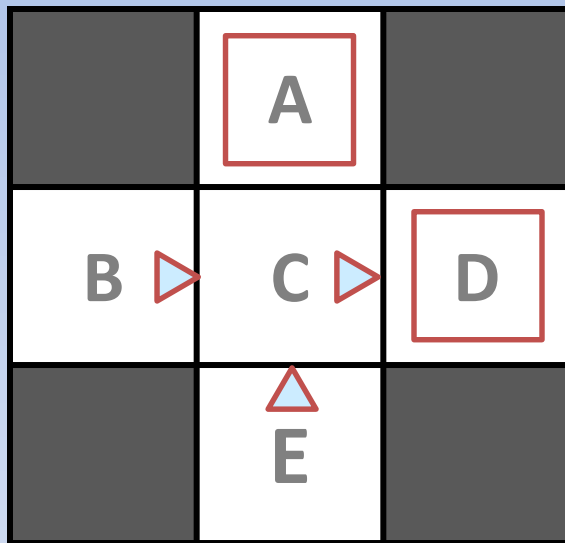
Episode 4

E, north, C, -1  
C, east, A, -1  
A, exit, x, -10

- Utility of E equals Average of
  - $-1+(-1)+10, -1+(-1)-10$

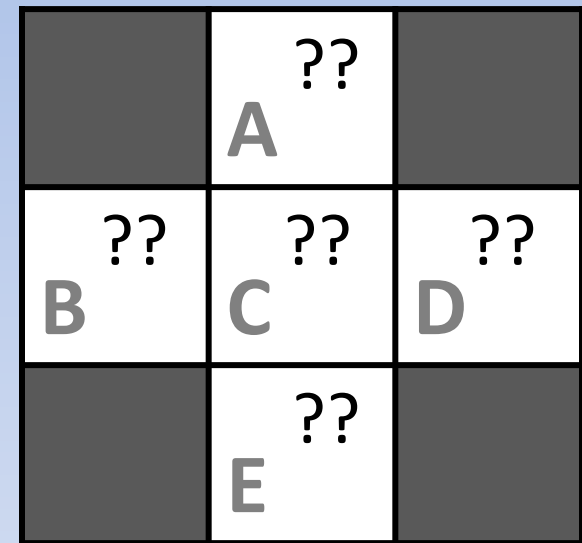
# Example: Direct Evaluation

Input Policy  $\pi$



Assume:  $\gamma = 1$

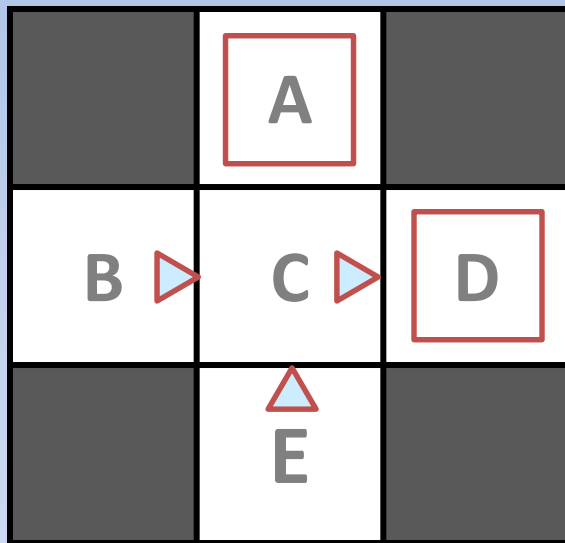
Output Values



- Episodes 1, 2, 3, 4

# Example: Direct Evaluation

Input Policy  $\pi$



Assume:  $\gamma = 1$

Output Values

	-10 A	
+8 B	+4 C	+10 D
	-2 E	

- Episodes 1, 2, 3, 4

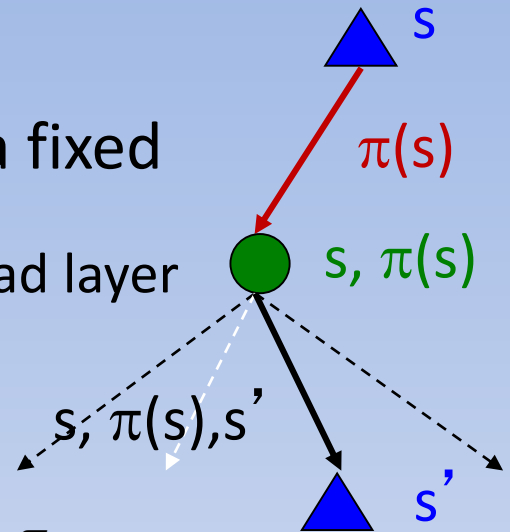
# Evaluating Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of  $T$ ,  $R$
  - It eventually computes the correct average values, using just sample transitions
- What bad about it?
  - It wastes information about state connections
  - **Each state must be learned separately**
  - So, it takes a long time to learn
- Need more MDP!
  - Bellman Equations!!!

- Simplified Bellman updates calculate  $V$  for a fixed policy:
  - Each round, replace  $V$  with a one-step-look-ahead layer over  $V$

$$V_0^\pi = 0$$

$$V_{k+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



- This approach fully exploited the connections between the states
- Unfortunately, we need  $T$  and  $R$  to do it!
- **Key question: how can we do this update to  $V$  without knowing  $T$  and  $R$ ?**
  - In other words, how to we take a weighted average without knowing the weights?

# Model-Free

$$V_{k+1}^{\pi}(s) =$$

$$\sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

- $E(X) = \sum_{x \in X} P(x)[x]$

- $E(X) = \frac{1}{n} \sum_i x_i$

- Drop  $P(x)$  with samples

- Need samples of:

- $R(s, \pi(s), s') + \gamma V_k^{\pi}(s')$

- Available from episodes

# Sample-Based Policy Evaluation?

$$V_{k+1}^{\pi}(s) =$$

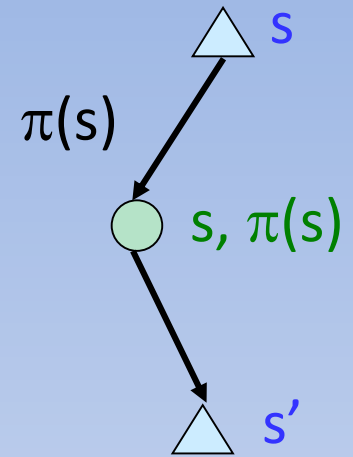
$$\sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

- Idea: Take samples of outcomes  $s'$  (by doing the action!) and average
  - $\text{sample}_1 = R(s, \pi(s), s') + \gamma V_k^{\pi}(s')$
  - $\text{sample}_2 = R(s, \pi(s), s') + \gamma V_k^{\pi}(s')$
  - \*\*\*  $\text{sample}_n = R(s, \pi(s), s') + \gamma V_k^{\pi}(s')$

$$V_{k+1}^{\pi}(s) = \frac{1}{n} \sum_i \text{sample}_i$$



# Running Average



$$V_{k+1}^{\pi}(s) = \frac{1}{n} \sum_i \text{sample}_i$$

- Still not convenient.
- Reformulate as running average:

$$\text{sample} = R(s, \pi(s), s') + \gamma V_k^{\pi}(s')$$

$$V_{k+1}^{\pi}(s) \leftarrow (1 - \alpha) V_k^{\pi}(s) + \alpha * \text{sample}$$

$$V_{k+1}^{\pi}(s) \leftarrow V_k^{\pi}(s) + \alpha(\text{sample} - V_k^{\pi}(s))$$

# Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update  $V(s)$  each time we experience a transition  $(s, a, s', r)$
  - Likely outcomes  $s'$  will contribute updates more often
- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

# Exponential moving average

- Exponential moving average

- The running interpolation update:  $\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
- Makes recent samples more important:

$$\bar{x}_n = \frac{x_n + (1 - \alpha) \cdot x_{n-1} + (1 - \alpha)^2 \cdot x_{n-2} + \dots}{1 + (1 - \alpha) + (1 - \alpha)^2 + \dots}$$

- Forgets about the past (distant past values were wrong anyway)
- Decreasing learning rate (alpha) can give converging averages

# Example: Temporal Difference Learning

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

## Observed Transitions

B, east, C, -2

C, east, D, -2

	0	
0	0	8
	0	

	??	
??	??	??
	??	

	??	
??	??	??
	??	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha \left[ R(s, \pi(s), s') + \gamma V^\pi(s') \right]$$

# Example: Temporal Difference Learning

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

## Observed Transitions

B, east, C, -2

	0	
0	0	8
	0	

	??	
??	??	??
	??	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

# Example:

## Temporal Difference Learning

	0	
0	0	8
	0	

B, east, C, -2

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

- State is: B
- Action is: east
- New state  $s'$  is: C
- Update  $V_k^\pi(B)$
- Sample =  $R(B, east, C) + \gamma V_k^\pi(C)$ 
  - $-2 + 1 \cdot 0 = -2$
- Update =  $\frac{1}{2} \cdot (V_k^\pi(B) = 0) + \frac{1}{2} \cdot (\text{Sample} = -2) = -1$

# Example: Temporal Difference Learning

	0	
0	0	8
	0	

B, east, C, -2

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

- State is: B
- Action is: east
- New state  $s'$  is: C
- Update  $V_k^\pi(B)$
- Sample =  $R(B, \text{east}, C) + \gamma V_k^\pi(C)$ 
  - $-2 + 1 \cdot 0 = -2$
- Update =  $\frac{1}{2} \cdot (V_k^\pi(B) = 0) + \frac{1}{2} \cdot (\text{Sample} = -2) = -1$

	??	
-1	??	??
	??	

# Example: Temporal Difference Learning

	0	
0	0	8
	0	

B, east, C, -2

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

- State is: B
- Action is: east
- New state  $s'$  is: C
- Update  $V_k^\pi(B)$
- Sample =  $R(B, east, C) + \gamma V_k^\pi(C)$ 
  - $-2 + 1 \cdot 0 = -2$
- Update =  $\frac{1}{2} \cdot (V_k^\pi(B) = 0) + \frac{1}{2} \cdot (\text{Sample} = -2) = -1$

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

	0	
-1	0	8
	0	



# Example: Temporal Difference Learning

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

## Observed Transitions

B, east, C, -2

C, east, D, -2

	0	
0	0	8
	0	

	0	
-1	0	8
	0	

	??	
??	??	??
	??	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

# Example:

## Temporal Difference Learning

	0	
-1	0	8
	0	

C, east, D, -2

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

- State is: C
- Action is: east
- New state  $s'$  is: D
- Update  $V_k^\pi(C)$
- Sample =  $R(C, \text{east}, D) + \gamma V_k^\pi(C)$ 
  - $-2 + 1 \cdot 8 = 6$
- Update =  $\frac{1}{2} \cdot (V_k^\pi(C) = 0) + \frac{1}{2} \cdot (\text{Sample} = 6) = 3$

# Example: Temporal Difference Learning

	0	
-1	0	8
	0	

C, east, D, -2

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

- State is: C
- Action is: east
- New state  $s'$  is: D
- Update  $V_k^\pi(C)$
- Sample =  $R(C, \text{east}, D) + \gamma V_k^\pi(C)$ 
  - $-2 + 1 \cdot 8 = 6$
- Update =  $\frac{1}{2} \cdot (V_k^\pi(C) = 0) + \frac{1}{2} \cdot (\text{Sample} = 6) = 3$

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

	0	
-1	3	8
	0	

# Temporal Difference Learning

```
function PASSIVE-TD-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal  $r'$ 
  persistent:  $\pi$ , a fixed policy
                $U$ , a table of utilities, initially empty
                $N_s$ , a table of frequencies for states, initially zero
                $s, a, r$ , the previous state, action, and reward, initially null

  if  $s'$  is new then  $U[s'] \leftarrow r'$ 
  if  $s$  is not null then
    increment  $N_s[s]$ 
     $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$ 
  if  $s'.\text{TERMINAL?}$  then  $s, a, r \leftarrow \text{null}$  else  $s, a, r \leftarrow s', \pi[s'], r'$ 
  return  $a$ 
```

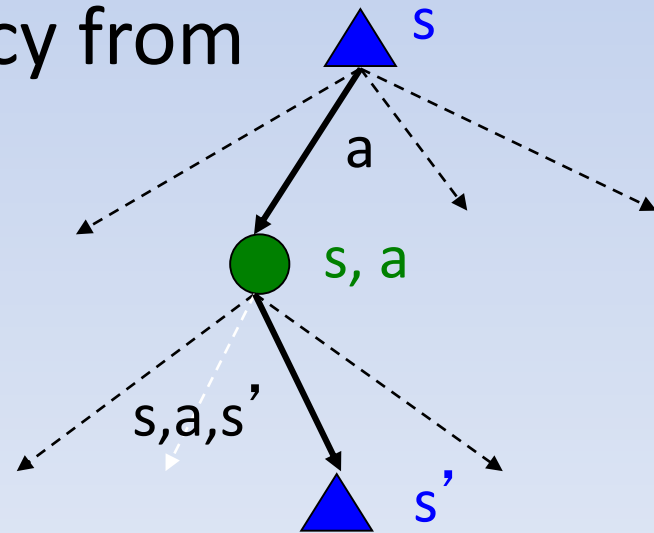
**Figure 21.4** A passive reinforcement learning agent that learns utility estimates using temporal differences. The step-size function  $\alpha(n)$  is chosen to ensure convergence, as described in the text.

# Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
  - YaY!
- How do we generate new policy from values???

$$\pi(s) = \arg \max_a Q(s, a)$$

$$Q(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V(s')]$$



# Problem

- Our error was not being focused on goal
  - Policy
- Learning needs to support goal!
- What should we learn to support goal??

# Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions  $T(s,a,s')$
  - You don't know the rewards  $R(s,a,s')$
  - You choose the actions now
  - Goal: learn the optimal policy / values
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens...

# Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with  $V_0(s) = 0$ , which we know is right
  - Given  $V_k$ , calculate the depth  $k+1$  values for all states:

$$V_{k+1}^{\pi}(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k^{\pi}(s')]$$

- Rather than the value of state, focus on Q-States
  - Q-values are more useful
  - Start with  $Q_0(s, a) = 0$ , which we know is right
  - Given  $Q_k$ , calculate the depth  $k+1$  q-values for all q-states:

$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_a Q_k(s', a)]$$



# Now w/ Model-Free Learning

## Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_a Q_k(s', a)]$$

- Learn  $Q(s,a)$  values as you go
  - Receive a sample  $(s,a,s',r)$
  - Consider your old estimate:
  - Consider your new sample estimate:

$$\text{sample} = R(s, a, s') + \gamma \max_a Q_k(s', a)$$

- Incorporate the new estimate into a running average:

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha(\text{sample})$$
$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha(\text{sample} - Q_k(s, a))$$

```

function Q-LEARNING-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal  $r'$ 
  persistent:  $Q$ , a table of action values indexed by state and action, initially zero
                $N_{sa}$ , a table of frequencies for state–action pairs, initially zero
                $s, a, r$ , the previous state, action, and reward, initially null

  if TERMINAL?( $s$ ) then  $Q[s, \text{None}] \leftarrow r'$ 
  if  $s$  is not null then
    increment  $N_{sa}[s, a]$ 
     $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$ 
     $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$ 
  return  $a$ 

```

**Figure 21.8** An exploratory Q-learning agent. It is an active learner that learns the value  $Q(s, a)$  of each action in each situation. It uses the same exploration function  $f$  as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.

# Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called **off-policy learning**
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - ... but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)

# Learning from Delayed Rewards

Machine Learning, 8, 279-292 (1992)

© 1992 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Christopher John Cornish Hellaby Watkins

King's College

## *Technical Note* Q-Learning

CHRISTOPHER J.C.H. WATKINS

*25b Framfield Road, Highbury, London N5 1UU, England*

PETER DAYAN

*Centre for Cognitive Science, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9EH, Scotland*

**Abstract.** Q-learning (Watkins, 1989) is a simple way for agents to learn how to act optimally in controlled Markovian domains. It amounts to an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states.

This paper presents and proves in detail a convergence theorem for Q-learning based on that outlined in Watkins (1989). We show that Q-learning converges to the optimum action-values with probability 1 so long as all actions are repeatedly sampled in all states and the action-values are represented discretely. We also sketch extensions to the cases of non-discounted, but absorbing, Markov environments, and where many Q values can be changed each iteration, rather than just one.

**Keywords.** Q-learning, reinforcement learning, temporal differences, asynchronous dynamic programming

Thesis Submitted for Ph.D.

May, 1989



## About us home

[News and events](#)[Experts](#)[Directory](#)[Allen, Nicholas Dr](#)[Alston, Richard Professor](#)[Anderberg, Dan Professor](#)[Annabel Valentine](#)[Ansari, Sarah Professor](#)[Armstrong, Tim Professor](#)[Benedetto, Giacomo Dr](#)

[Home](#) > [About us home](#) > [News and events](#) > [Experts](#) > [Watkins, Chris Dr](#)

## Dr Chris Watkins

**Reader in Artificial Intelligence**  
**Department of Computer Science**

### Area of expertise

Machine learning: how machines can learn from experience in performing and in making predictions; Big Data; Visualisation of complex data; Empowering the community to respond better to pandemic infectious disease with mobile phones and online social media.



To arrange an interview with this expert or speak to them about their expertise, please contact:

The Press Office on 01784 443552

## Find your course



“

How to construct machines that learn is one of the central problems of science. Although whole new industrial sectors have sprung from machine learning in the last fifteen years, we are still far from making any truly intelligent machine.

Chris Watkins



[See full research profile](#)

### Search for an expert:

---

# Playing Atari with Deep Reinforcement Learning

---

Volodymyr Mnih   Koray Kavukcuoglu   David Silver   Alex Graves   Ioannis Antonoglou

Daan Wierstra   Martin Riedmiller

DeepMind Technologies

`{vlad, koray, david, alex.graves, ioannis, daan, martin.riedmiller} @ deepmind.com`

## Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

# Variant of Q-Learning

## Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

---

## Playing Atari with Deep Reinforcement Learning

---

Volodymyr Mnih   Koray Kavukcuoglu   David Silver   Alex Graves   Ioannis Antonoglou

Daan Wierstra   Martin Riedmiller

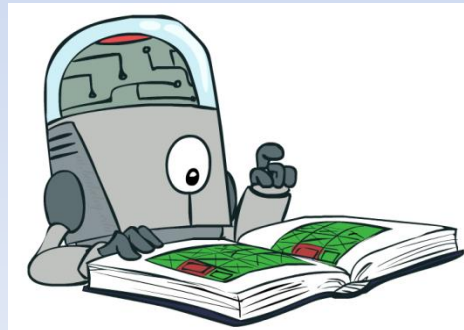
DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com



# Generalizing Across States

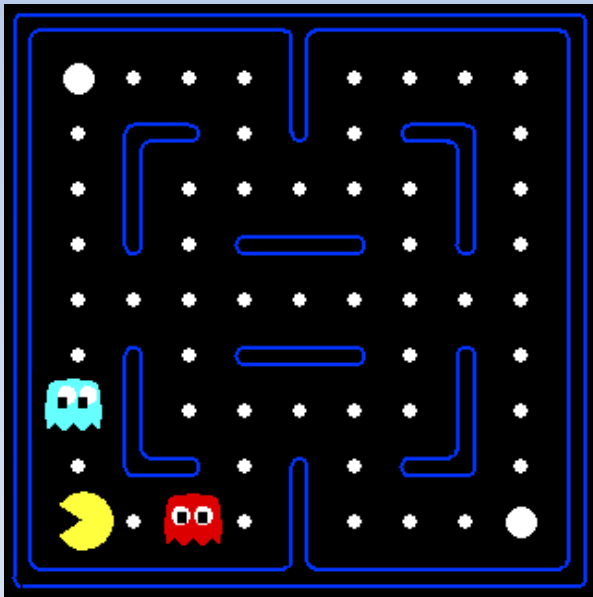
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again



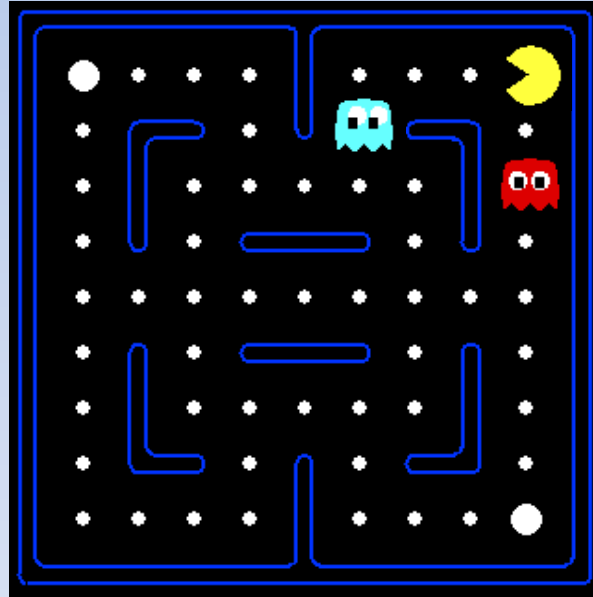


# Example: Pacman

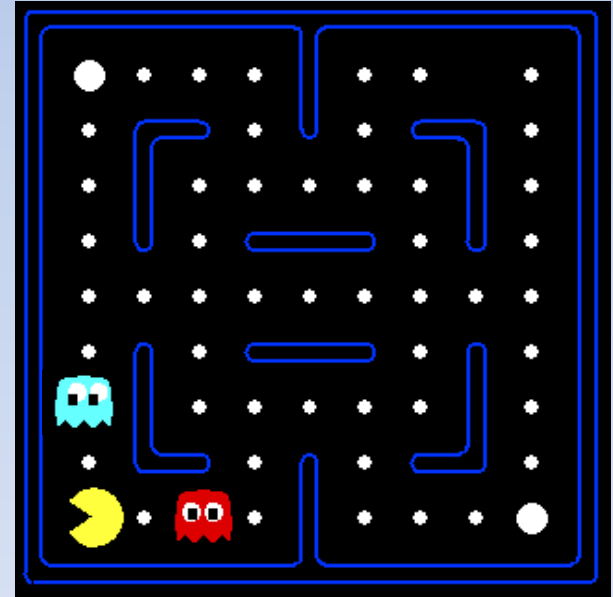
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:

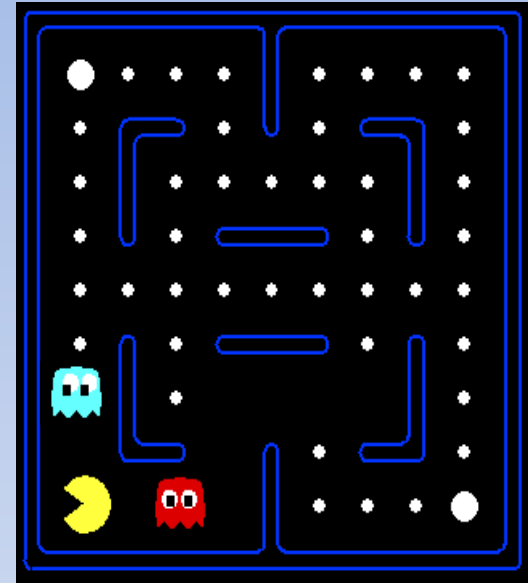


Or even this one!



# Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - $1 / (\text{dist to dot})^2$
    - Is Pacman in a tunnel? (0/1)
    - ..... etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state  $(s, a)$  with features (e.g. action moves closer to food)



# Linear Value Functions

- Using a feature representation, we can write a  $q$  function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!

# Approximate Q-Learning

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$$

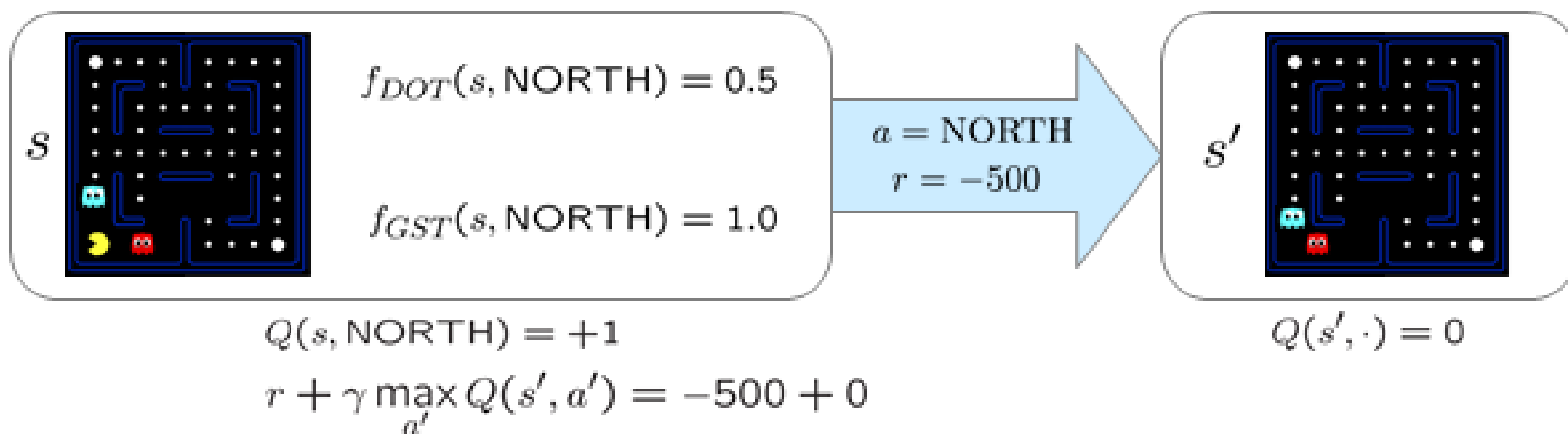
$$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{difference}] \quad \text{Exact Q's}$$

$$w_i \leftarrow w_i + \alpha [\text{difference}] f_i(s, a) \quad \text{Approximate Q's}$$

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares

## Example: Q-Pacman

$$Q(s, a) = 4.0f_{DOT}(s, a) - 1.0f_{GST}(s, a)$$



difference = -501  $\longrightarrow$

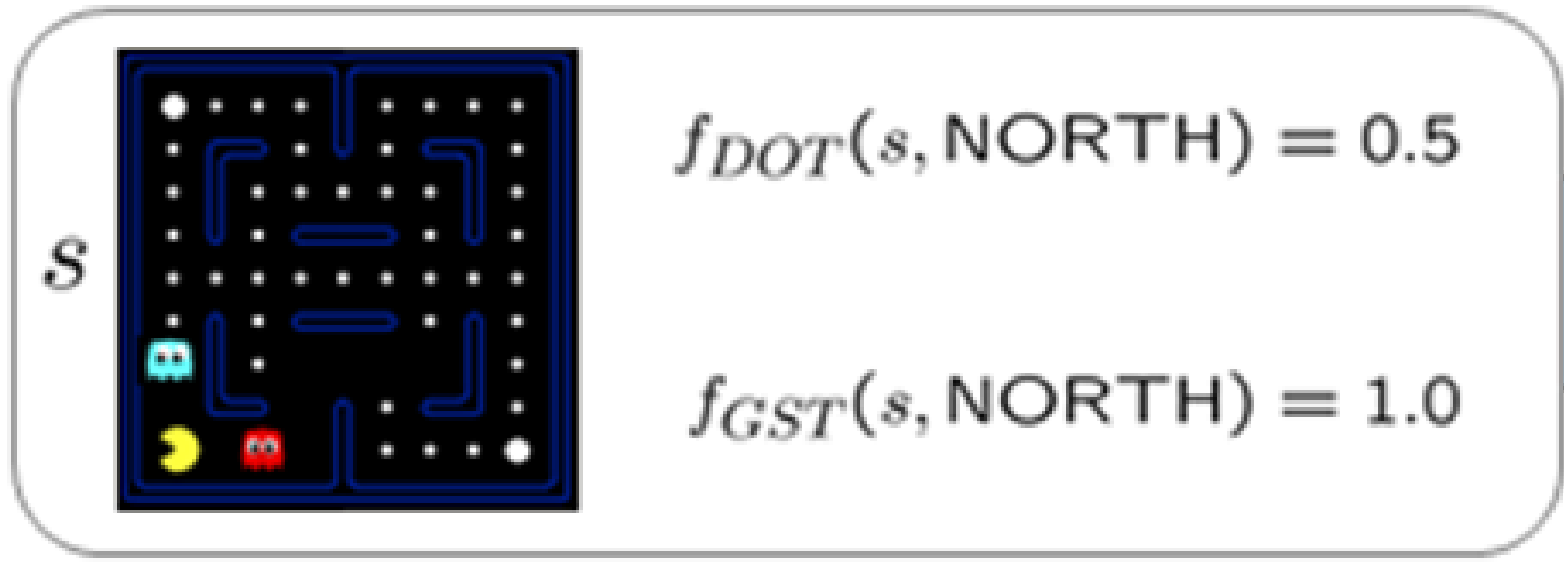
$$w_{DOT} \leftarrow 4.0 + \alpha [-501] 0.5$$

$$w_{GST} \leftarrow -1.0 + \alpha [-501] 1.0$$

$$Q(s, a) = 3.0f_{DOT}(s, a) - 3.0f_{GST}(s, a)$$

[Demo: approximate Q-learning pacman (L11D10)]

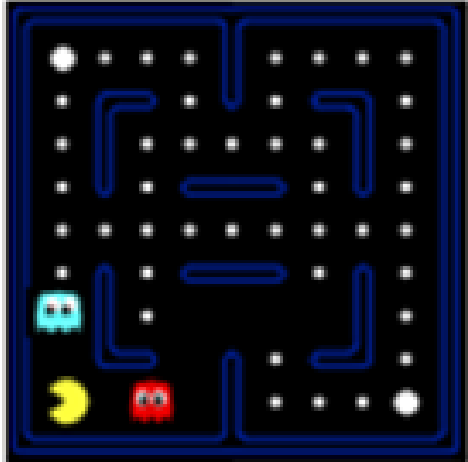
# Current State: S



- $f_{DOT}(s, a) = 1/(\text{distance to nearest dot after executing action } a \text{ in state } s)$
- $f_{GST}(s, a) = (\text{Distance to nearest ghost after executing action } a \text{ in state } s)$
- Might need different weights for each action in other situations.

# Current State: S

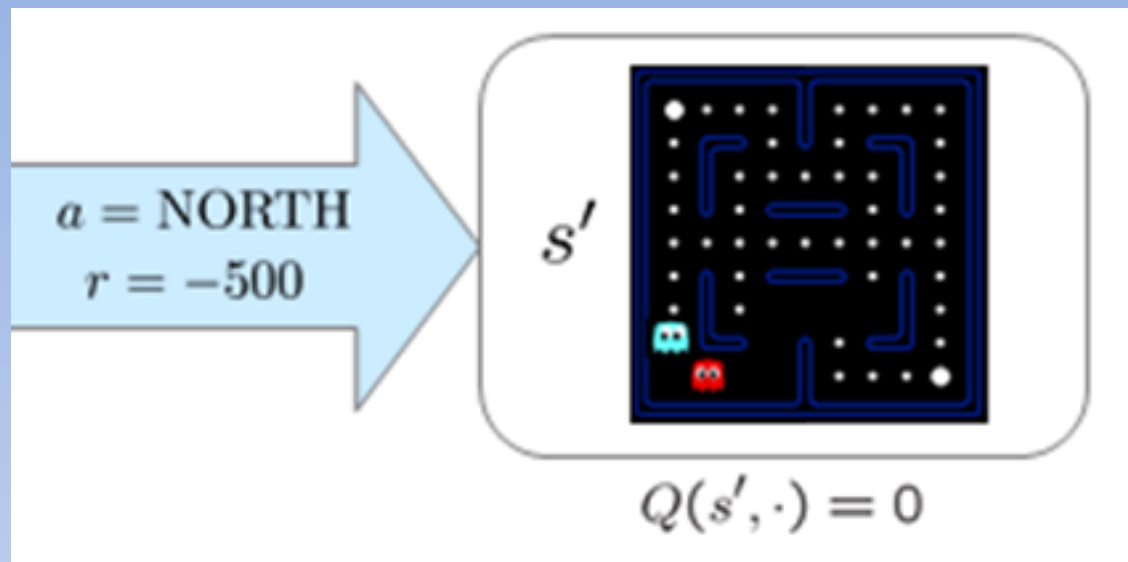
*S*



$$f_{DOT}(s, \text{NORTH}) = 0.5$$
$$f_{GST}(s, \text{NORTH}) = 1.0$$

$$Q(s, a) = 4.0f_{DOT}(s, a) - 1.0f_{GST}(s, a)$$

- $Q(s, \text{NORTH}) = 4 * 0.5 - 1 * 1.0 = 1.0$



$$\text{difference} = \left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$$

- sample =  $-500 + Q(s', \cdot) = -500 + 0 = -500$
- $Q(s, \text{NORTH}) = 4.0 * 0.5 - 1.0 * 1.0 = 1.0$
- difference = -501

difference = -501



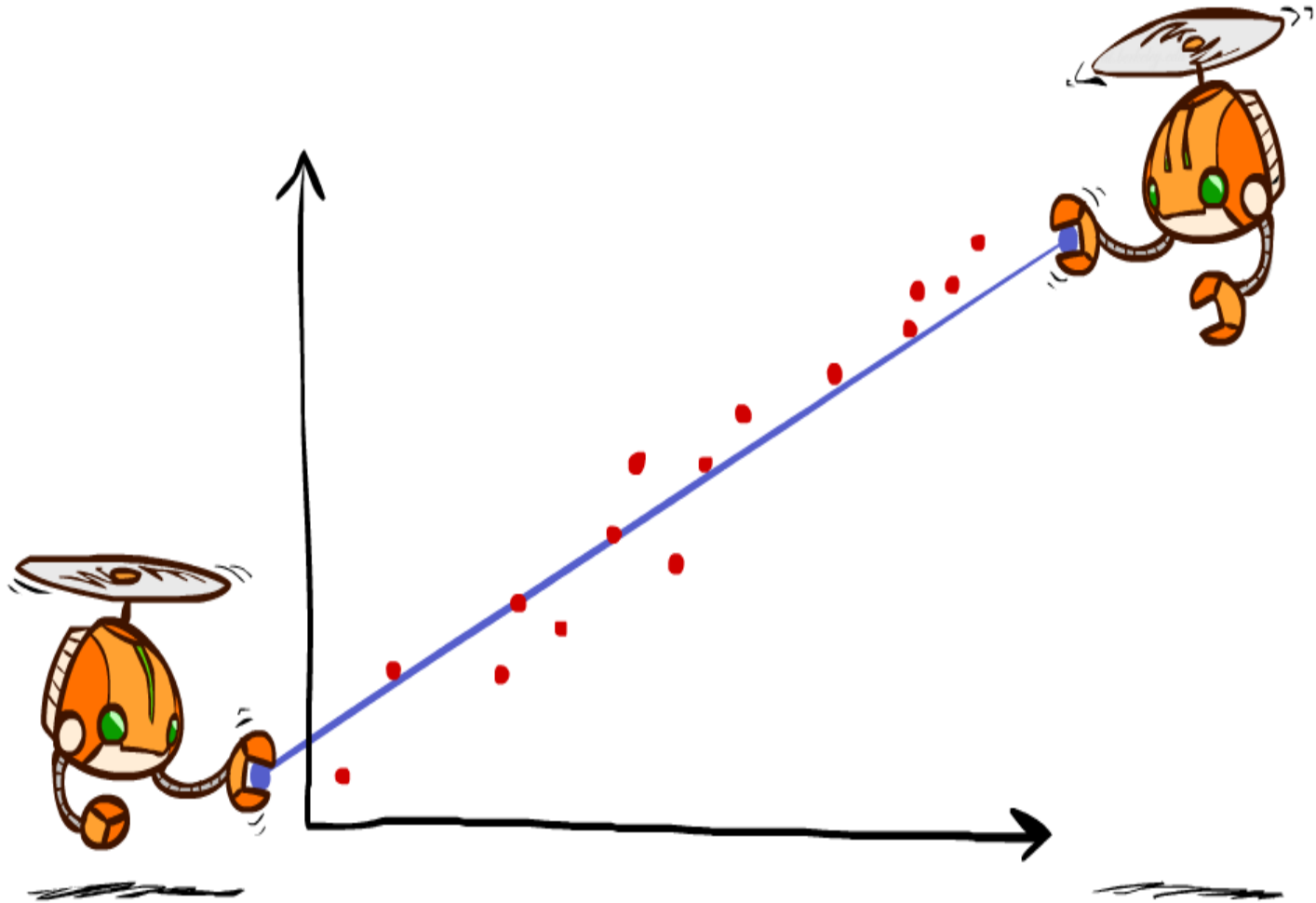
$$w_{DOT} \leftarrow 4.0 + \alpha [-501] 0.5$$

$$w_{GST} \leftarrow -1.0 + \alpha [-501] 1.0$$

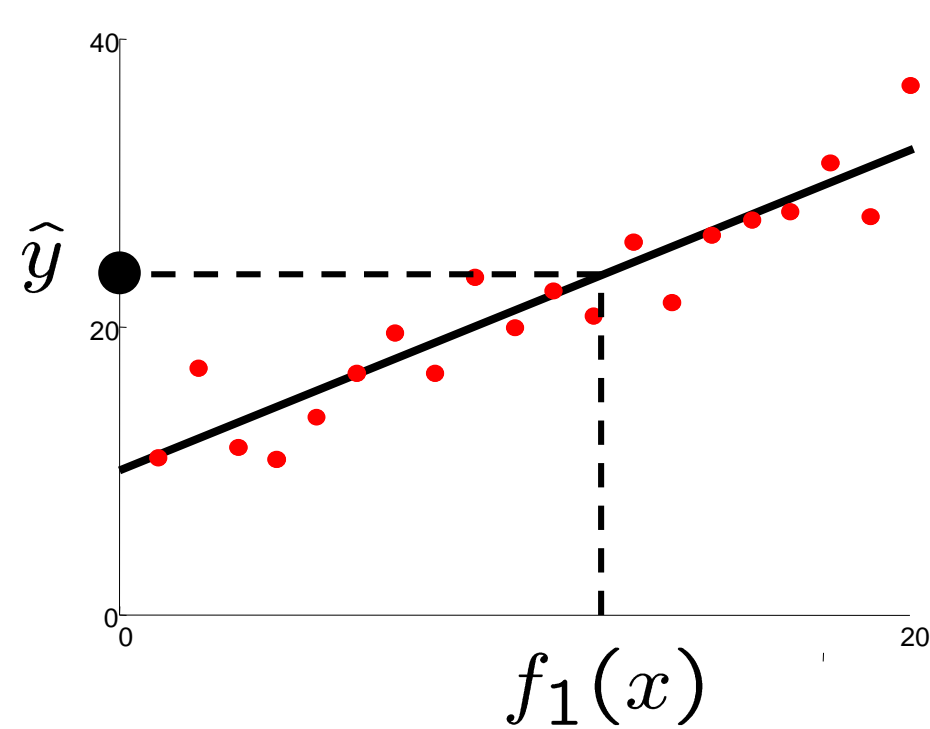
$$Q(s, a) = 3.0 f_{DOT}(s, a) - 3.0 f_{GST}(s, a)$$



# Q-Learning and Least Squares

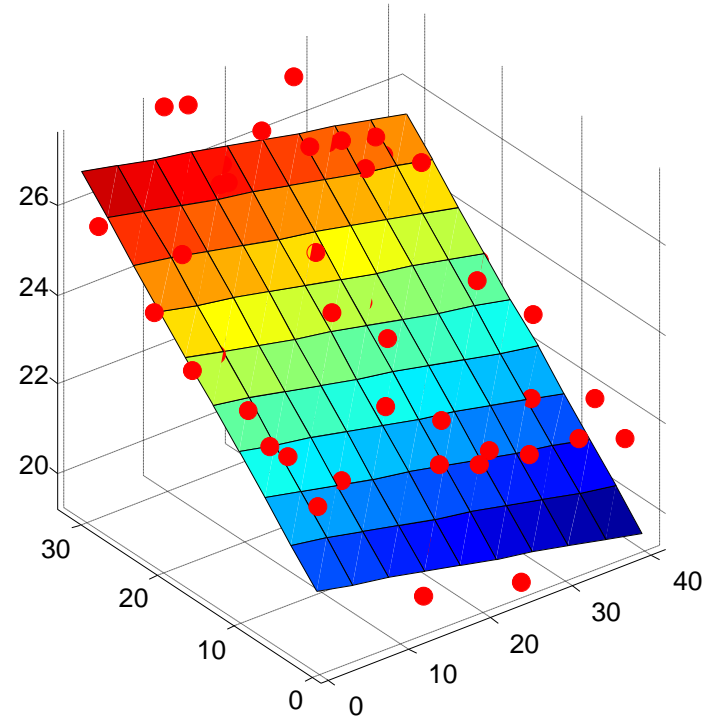


# Linear Approximation: Regression\*



Prediction:

$$\hat{y} = w_0 + w_1 f_1(x)$$

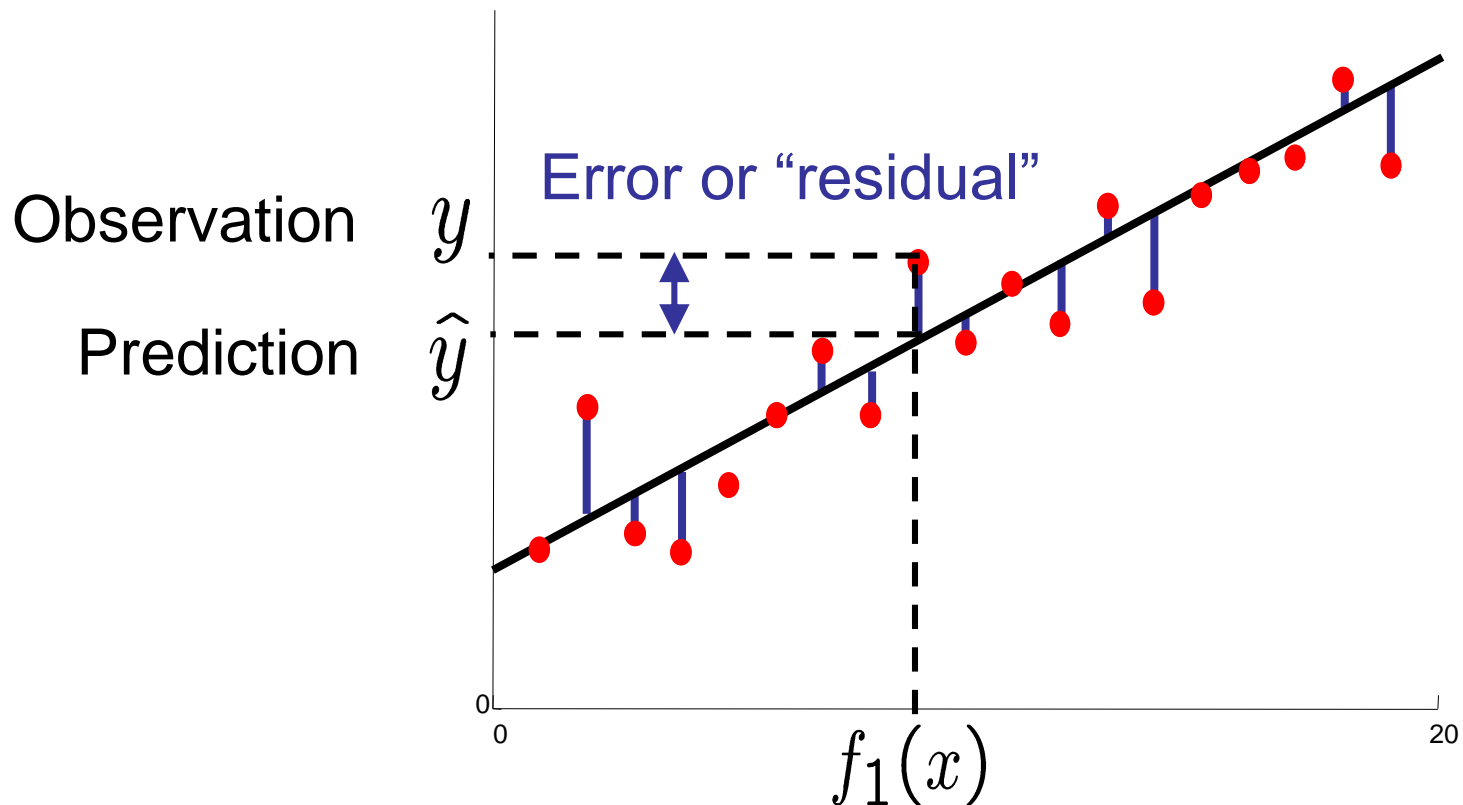


Prediction:

$$\hat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$$

# Optimization: Least Squares\*

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left( y_i - \sum_k w_k f_k(x_i) \right)^2$$



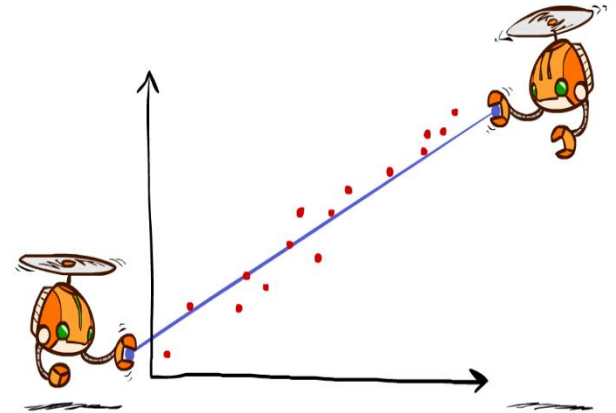
# Minimizing Error\*

Imagine we had only one point  $x$ , with features  $f(x)$ , target value  $y$ , and weights  $w$ :

$$\text{error}(w) = \frac{1}{2} \left( y - \sum_k w_k f_k(x) \right)^2$$

$$\frac{\partial \text{error}(w)}{\partial w_m} = - \left( y - \sum_k w_k f_k(x) \right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left( y - \sum_k w_k f_k(x) \right) f_m(x)$$



Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[ \underset{\text{“target”}}{r + \gamma \max_a Q(s', a')} - \underset{\text{“prediction”}}{Q(s, a)} \right] f_m(s, a)$$

## QUIZ 2: FEATURE-BASED REPRESENTATIONS (9/9 points)

Consider the following feature based representation of the Q-function:

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a)$$

with

$$f_1(s, a) = 1 / (\text{distance to nearest dot after having executed action } a \text{ in state } s)$$

$$f_2(s, a) = (\text{distance to nearest ghost after having executed action } a \text{ in state } s)$$

### Part 1

Assume  $w_1 = 1$ ,  $w_2 = 10$ . For the state  $s$  shown below, find the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

- $Q(s, \text{West}) = ?$
- $Q(s, \text{South}) = ?$
- Based on this approximate Q-function, which action would be chosen?



## QUIZ 2: FEATURE-BASED REPRESENTATIONS (9/9 points)

Consider the following feature based representation of the Q-function:

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a)$$

with

$$f_1(s, a) = 1/(\text{distance to nearest dot after having executed action } a \text{ in state } s)$$

$$f_2(s, a) = (\text{distance to nearest ghost after having executed action } a \text{ in state } s)$$

### Part 1

Assume  $w_1 = 1$ ,  $w_2 = 10$ . For the state  $s$  shown below, find the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

- $Q(s, \text{West}) =$ 
  - $1*(1/1) + 10*(3 \text{ moves away})$
- $Q(s, \text{South}) =$ 
  - $1*(1/1) + 10*(1 \text{ move away})$
- Based on this approximate Q-function, which action would be chosen?



## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

- Q-Values:
  - $Q(s', \text{West}) = ?$
  - $Q(s', \text{East}) = ?$
- Now compute update with  $\alpha = 0.5$ :
  - Sample =
  - Difference =
  - $w_1 =$
  - $w_2 =$

Now let's compute the update to the weights. Let  $\alpha = 0.5$ .

$$\text{difference} = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) =$$

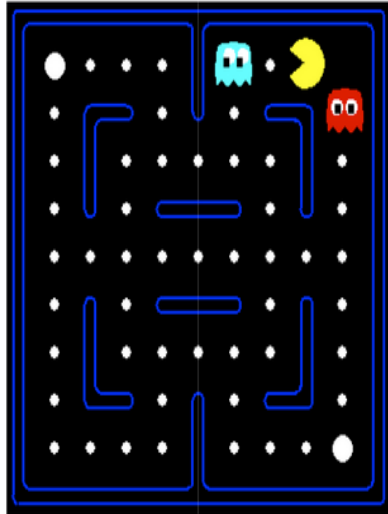
$$w_1 \leftarrow w_1 + \alpha (\text{difference}) f_1(s, a) =$$

$$w_2 \leftarrow w_2 + \alpha (\text{difference}) f_2(s, a) =$$

# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



$Q(s', West) =$

$Q(s', East) =$

What is the sample value (assuming  $\gamma = 1$ )?

The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$Q(s', West) = 1 * (1/1) + 10 * (1 \text{ move away})$

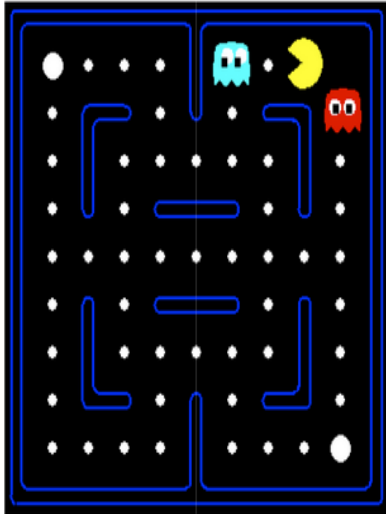
$Q(s', East) = 1 * (1/1) + 10 * (1 \text{ move away})$



# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$$Q(s', \text{West}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$Q(s', \text{East}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$Q(s', \text{West}) =$$



$$Q(s', \text{East}) =$$

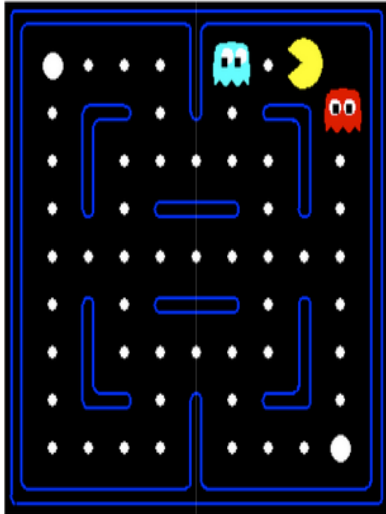


What is the sample value (assuming  $\gamma = 1$ )?

# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



$$Q(s', West) =$$



$$Q(s', East) =$$



What is the sample value (assuming  $\gamma = 1$ )?

The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$$Q(s', West) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$Q(s', East) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$\text{Sample} = 9 \text{ (Reward)} + 1 \text{ } (\gamma) * 11 \text{ } (\max_a Q(s', a'))$$

### Learn $Q(s,a)$ values as you go

- Receive a sample  $(s, a, s', r)$
- Consider your old estimate:  $Q(s, a)$
- Consider your new sample estimate:

$$\text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

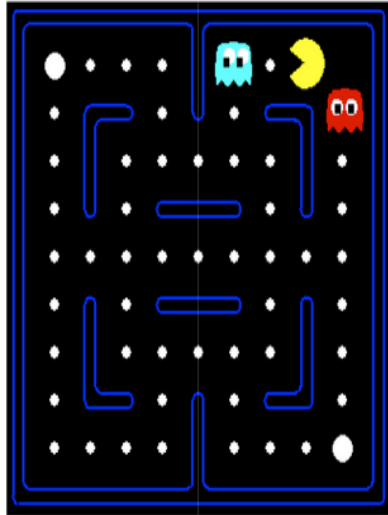
- Incorporate the new estimate into a running average:

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + (\alpha) [\text{sample}]$$

# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$$Q(s', \text{West}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$Q(s', \text{East}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$\text{Sample} = 9 \text{ (Reward)} + 1 \text{ } (\gamma) * 11 \text{ (}\max_a Q(s', a')\text{)}$$

$$Q(s', \text{West}) =$$



$$Q(s', \text{East}) =$$



What is the sample value (assuming  $\gamma = 1$ )?



### Learn $Q(s,a)$ values as you go

- Receive a sample  $(s, a, s', r)$
- Consider your old estimate:  $Q(s, a)$
- Consider your new sample estimate:

$$\text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

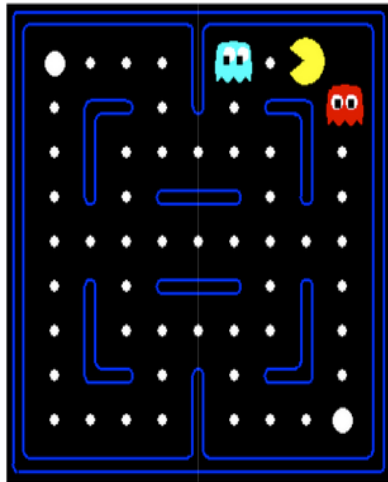
- Incorporate the new estimate into a running average:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) [\text{sample}]$$

# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$$Q(s', \text{West}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$Q(s', \text{East}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$\text{Sample} = 9 \text{ (Reward)} + 1 \text{ } (\gamma) * 11$$

$$(\max_a Q(s', a')) = 20$$

$$\text{Difference} = 20 \text{ (Sample)} - 31 \text{ (} Q(s, \text{west}) \text{)} = -11$$

## Part 3

Now let's compute the update to the weights. Let  $\alpha = 0.5$ .

$$\text{difference} = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) =$$

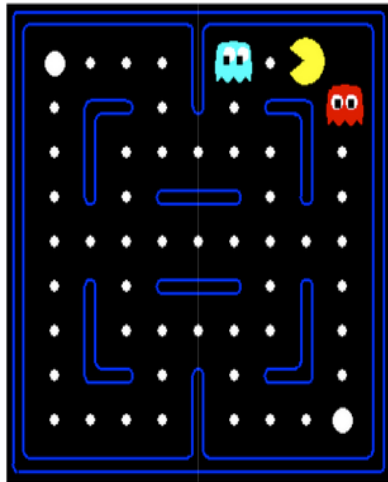
$$w_1 \leftarrow w_1 + \alpha (\text{difference}) f_1(s, a) =$$

$$w_2 \leftarrow w_2 + \alpha (\text{difference}) f_2(s, a) =$$

# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$$Q(s', \text{West}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$Q(s', \text{East}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$\text{Sample} = 9 \text{ (Reward)} + 1 \text{ } (\gamma) * 11$$

$$(\max_a Q(s', a)) = 20$$

$$\text{Difference} = 20 \text{ (Sample)} - 31 \text{ (} Q(s, \text{west}) \text{)} = -11$$

## Part 3

Now let's compute the update to the weights. Let  $\alpha = 0.5$ .

$$\text{difference} = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) =$$



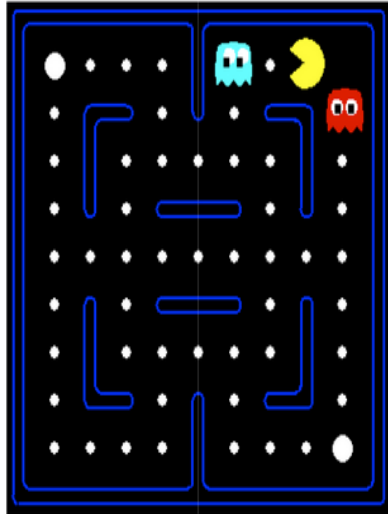
$$w_1 \leftarrow w_1 + \alpha (\text{difference}) f_1(s, a) =$$

$$w_2 \leftarrow w_2 + \alpha (\text{difference}) f_2(s, a) =$$

# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$$Q(s', \text{West}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$Q(s', \text{East}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$\text{Sample} = 9 \text{ (Reward)} + 1 \text{ } (\gamma) * 11 \text{ (max}_a Q(s', a')) = 20$$

$$\text{Difference} = 20 \text{ (Sample)} - 31 \text{ (Q(s, west))} = -11$$

$$w_1 \leftarrow 1 \text{ (} w_1 \text{)} + 0.5 \text{ (} \alpha \text{)} * 1 * (-11 \text{ diff}) * (f_1(s, \text{west})) = 1 - 5.5 = -4.5$$

## Part 3

Now let's compute the update to the weights. Let  $\alpha = 0.5$ .

$$\text{difference} = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) =$$



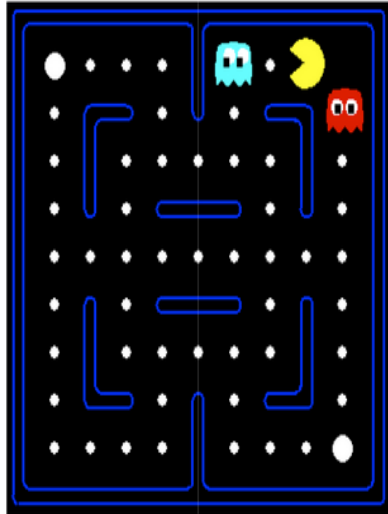
$$w_1 \leftarrow w_1 + \alpha (\text{difference}) f_1(s, a) =$$

$$w_2 \leftarrow w_2 + \alpha (\text{difference}) f_2(s, a) =$$

# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$$Q(s', \text{West}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$Q(s', \text{East}) = 1 * (1/1) + 10 * (1 \text{ move away})$$

$$\text{Sample} = 9 \text{ (Reward)} + 1 \text{ } (\gamma) * 11 \text{ (max}_a Q(s', a')) = 20$$

$$\text{Difference} = 20 \text{ (Sample)} - 31 \text{ (Q(s, west))} = -11$$

$$w_1 \leftarrow 1 \text{ (} w_1 \text{)} + 0.5 \text{ (} \alpha \text{)} * 1 * (-11 \text{ diff}) * (f_1(s, \text{west})) = 1 - 5.5 = -4.5$$

## Part 3

Now let's compute the update to the weights. Let  $\alpha = 0.5$ .

$$\text{difference} = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) =$$



$$w_1 \leftarrow w_1 + \alpha (\text{difference}) f_1(s, a) =$$

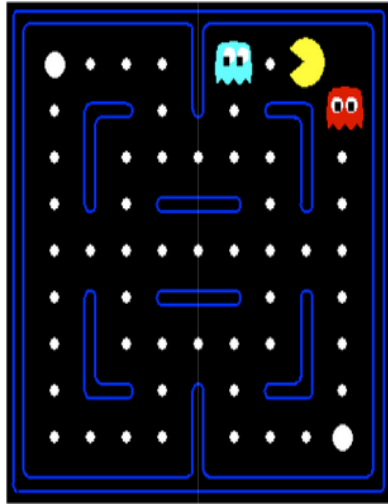


$$w_2 \leftarrow w_2 + \alpha (\text{difference}) f_2(s, a) =$$

# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

Sample = 9 (Reward) + 1 ( $\gamma$ ) \* 11 ( $\max_a Q(s', a')$ ) = 20

Difference = 20 (Sample) - 31 ( $Q(s, \text{west})$ ) = -11

$w_1 \leftarrow 1 (w_1) + 0.5 (\alpha) * 1 * (-11 \text{ diff}) * (1 f_1(s, \text{west})) = 1 - 5.5 = -4.5$

$w_2 \leftarrow 10 (w_1) + 0.5 (\alpha) * 1 * (-11 \text{ diff}) * (3 f_1(s, \text{west})) =$

$10 + 0.5 * (-11 * 3) = 10 + 0.5 * (-33) = 10 - 16.5 = -6.5$

## Part 3

Now let's compute the update to the weights. Let  $\alpha = 0.5$ .

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

-11



$w_1 \leftarrow w_1 + \alpha (\text{difference}) f_1(s, a) =$

-4.5



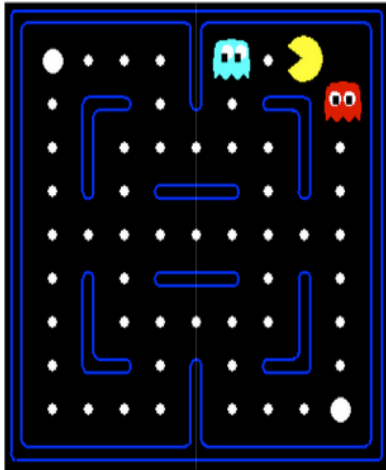
$w_2 \leftarrow w_2 + \alpha (\text{difference}) f_2(s, a) =$



# Question

## Part 2

Assume Pac-Man moves West. This results in the state  $s'$  shown below.



The reward for this transition is  $r = +10 - 1 = 9$  (+10: for food pellet eating, -1 for time passed). Fill in following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$$\text{Sample} = 9 \text{ (Reward)} + 1 \text{ } (\gamma) * 11 \text{ (max}_a Q(s', a')) = 20$$

$$\text{Difference} = 20 \text{ (Sample)} - 31 \text{ (Q(s, west))} = -11$$

$$w_1 \leftarrow 1 \text{ (} w_1 \text{)} + 0.5 \text{ (} \alpha \text{)} * 1 * (-11 \text{ diff}) * (1 \text{ } f_1(s, \text{west})) = 1 - 5.5 = -4.5$$

$$w_2 \leftarrow 10 \text{ (} w_1 \text{)} + 0.5 \text{ (} \alpha \text{)} * 1 * (-11 \text{ diff}) * (3 \text{ } f_1(s, \text{west})) =$$

$$10 + 0.5 * (-11 * 3) = 10 + 0.5 * (-33) = 10 - 16.5 = -6.5$$

## Part 3

Now let's compute the update to the weights. Let  $\alpha = 0.5$ .

$$\text{difference} = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) =$$



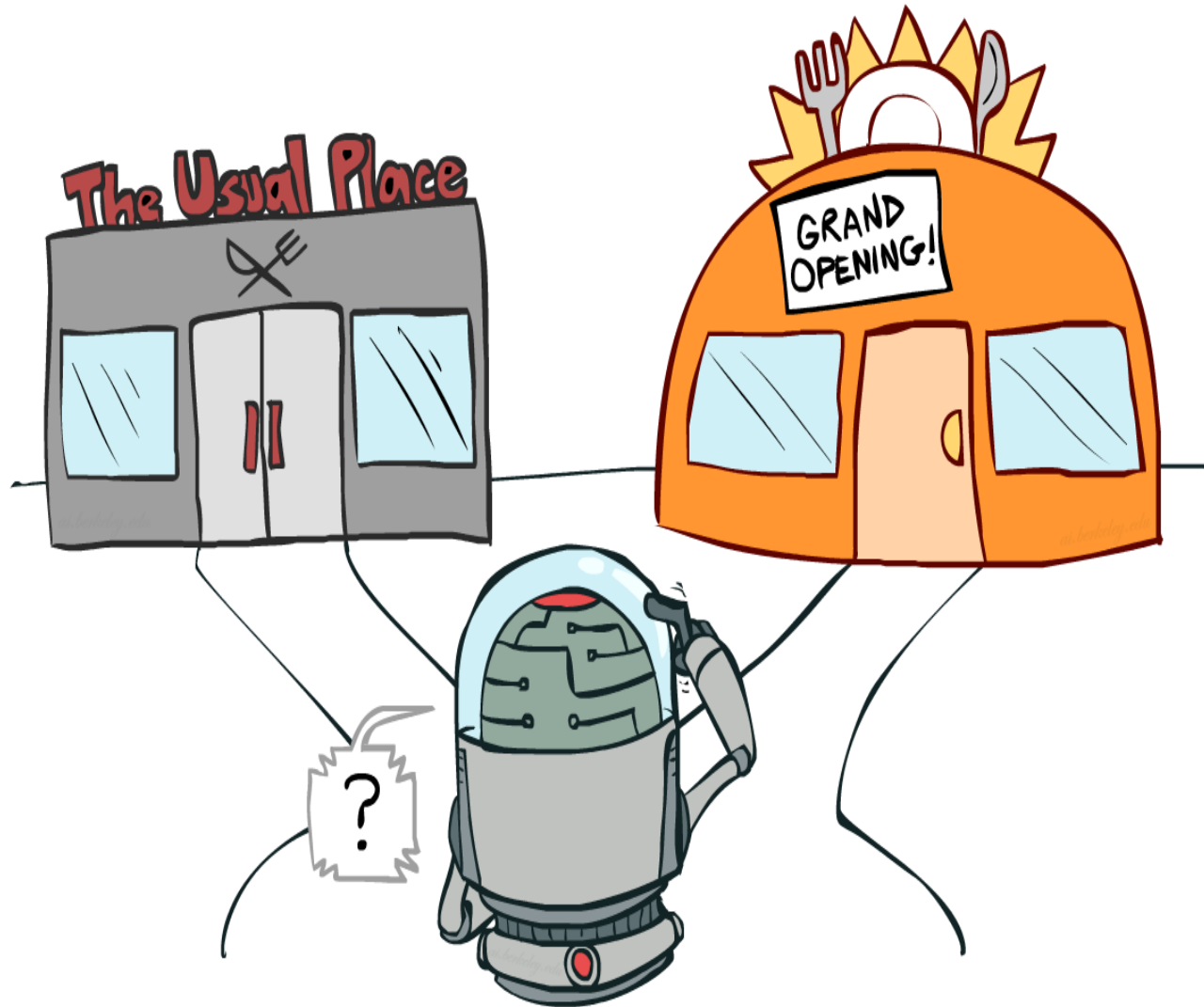
$$w_1 \leftarrow w_1 + \alpha (\text{difference}) f_1(s, a) =$$



$$w_2 \leftarrow w_2 + \alpha (\text{difference}) f_2(s, a) =$$



# Exploration vs. Exploitation



# How to Explore?

---

- Several schemes for forcing exploration
  - Simplest: random actions ( $\epsilon$ -greedy)
    - Every time step, flip a coin
    - With (small) probability  $\epsilon$ , act randomly
    - With (large) probability  $1-\epsilon$ , act on current policy
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: lower  $\epsilon$  over time
    - Another solution: exploration functions

# Exploration Functions



- When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

- Exploration function

- Takes a value estimate  $u$  and a visit count  $n$ , and returns an optimistic utility, e.g.

$$f(u, n) = u + k/n$$

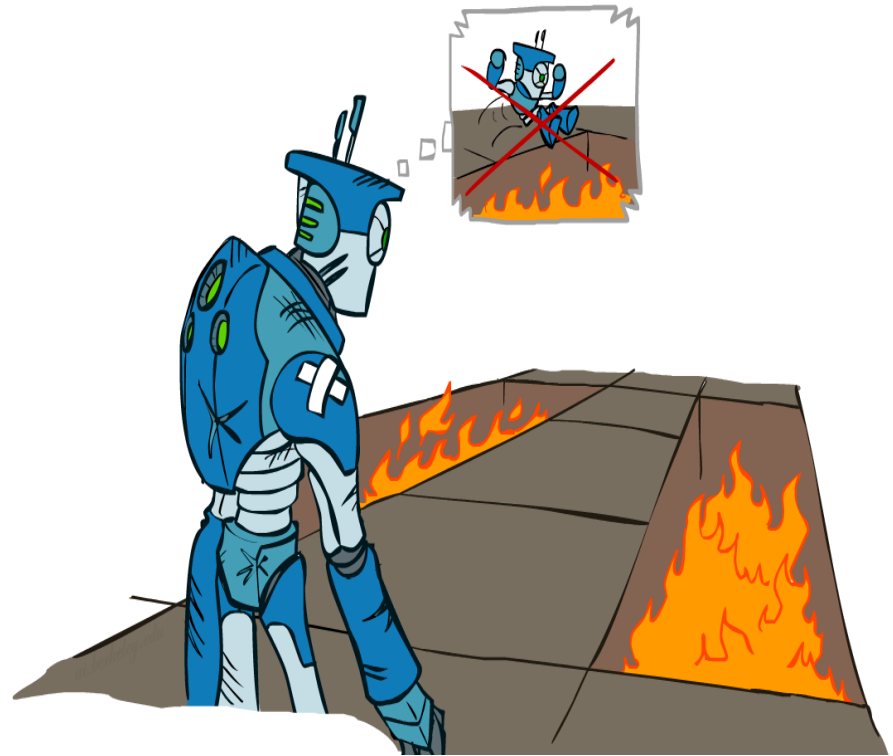
Regular Q-Update:  $Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} Q(s', a')$

Modified Q-Update:  $Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$

- Note: this propagates the “bonus” back to states that lead to unknown states as well!

# Regret

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret



# Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate  $V$  /  $Q$  best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
  - We'll see this distinction between modeling and prediction again later in the course
- Solution: learn policies that maximize rewards, not the values that predict them
- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

# Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
- Better methods exploit lookahead structure, sample wisely, change multiple parameters...

# Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct
- Step 1: Learn empirical MDP model
  - Count outcomes  $s'$  for each  $s, a$
  - Normalize to give an estimate of  $\hat{T}(s, a, s')$
  - Discover each  $\hat{R}(s, a, s')$  when we experience  $(s, a, s')$
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before



# Passive Learning Agent

- State-based Representation
- Fully observable environment
- Fixed policy  $\pi$
- In state  $s$  agent always executes action  $\pi(s)$

```

function PASSIVE-ADP-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal  $r'$ 
  persistent:  $\pi$ , a fixed policy
               mdp, an MDP with model  $P$ , rewards  $R$ , discount  $\gamma$ 
                $U$ , a table of utilities, initially empty
                $N_{sa}$ , a table of frequencies for state–action pairs, initially zero
                $N_{s'|sa}$ , a table of outcome frequencies given state–action pairs, initially zero
                $s, a$ , the previous state and action, initially null

  if  $s'$  is new then  $U[s'] \leftarrow r'$ ;  $R[s'] \leftarrow r'$ 
  if  $s$  is not null then
    increment  $N_{sa}[s, a]$  and  $N_{s'|sa}[s', s, a]$ 
    for each  $t$  such that  $N_{s'|sa}[t, s, a]$  is nonzero do
       $P(t | s, a) \leftarrow N_{s'|sa}[t, s, a] / N_{sa}[s, a]$ 
     $U \leftarrow \text{POLICY-EVALUATION}(\pi, U, \text{mdp})$ 
  if  $s'.\text{TERMINAL?}$  then  $s, a \leftarrow \text{null}$  else  $s, a \leftarrow s', \pi[s']$ 
  return  $a$ 

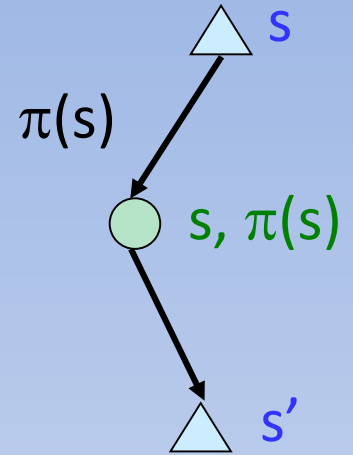
```

**Figure 21.2** A passive reinforcement learning agent based on adaptive dynamic programming. The POLICY-EVALUATION function solves the fixed-policy Bellman equations, as described on page 657.

# Direct-Evaluation

- Goal: Compute values for each state under  $\pi$
- Idea: Average together observed sample values
  - Act according to  $\pi$
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

# Temporal-Difference Learning



$$V_{k+1}^{\pi}(s) = \frac{1}{n} \sum_i \text{sample}_i$$

$$\text{sample} = R(s, \pi(s), s') + \gamma V_k^{\pi}(s')$$

$$V_{k+1}^{\pi}(s) \leftarrow (1 - \alpha)V_k^{\pi}(s) + \alpha * \text{sample}$$

$$V_{k+1}^{\pi}(s) \leftarrow V_k^{\pi}(s) + \alpha(\text{sample} - V_k^{\pi}(s))$$

# Temporal Difference Learning

```
function PASSIVE-TD-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal  $r'$ 
  persistent:  $\pi$ , a fixed policy
                $U$ , a table of utilities, initially empty
                $N_s$ , a table of frequencies for states, initially zero
                $s, a, r$ , the previous state, action, and reward, initially null

  if  $s'$  is new then  $U[s'] \leftarrow r'$ 
  if  $s$  is not null then
    increment  $N_s[s]$ 
     $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$ 
  if  $s'.\text{TERMINAL?}$  then  $s, a, r \leftarrow \text{null}$  else  $s, a, r \leftarrow s', \pi[s'], r'$ 
  return  $a$ 
```

**Figure 21.4** A passive reinforcement learning agent that learns utility estimates using temporal differences. The step-size function  $\alpha(n)$  is chosen to ensure convergence, as described in the text.

# Now w/ Model-Free Learning

## Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_a Q_k(s, a)]$$

- Learn  $Q(s,a)$  values as you go
  - Receive a sample  $(s,a,s',r)$
  - Consider your old estimate:
  - Consider your new sample estimate:

$$\text{sample} = R(s, a, s') + \gamma \max_a Q_k(s, a)$$

- Incorporate the new estimate into a running average:

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha(\text{sample})$$
$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha(\text{sample} - Q_k(s, a))$$

```

function Q-LEARNING-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal  $r'$ 
  persistent:  $Q$ , a table of action values indexed by state and action, initially zero
                $N_{sa}$ , a table of frequencies for state–action pairs, initially zero
                $s, a, r$ , the previous state, action, and reward, initially null

  if TERMINAL?( $s$ ) then  $Q[s, \text{None}] \leftarrow r'$ 
  if  $s$  is not null then
    increment  $N_{sa}[s, a]$ 
     $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$ 
     $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$ 
  return  $a$ 

```

**Figure 21.8** An exploratory Q-learning agent. It is an active learner that learns the value  $Q(s, a)$  of each action in each situation. It uses the same exploration function  $f$  as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.

# Approximate Q-Learning

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{difference}] \quad \text{Exact Q's}$$

$$w_i \leftarrow w_i + \alpha [\text{difference}] f_i(s, a) \quad \text{Approximate Q's}$$

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares