

# Reinforcement Learning – Learning from Interaction

# Learning to Control

- So far looked at two models of **supervised** learning
  - Supervised:
    - Classification,
    - Regression, etc.
- Alternative: No labels – Unsupervised learning, Clustering, etc.
- How did you learn to cycle?
  - Neither of the above
  - Trial and error!
  - Falling down hurts!

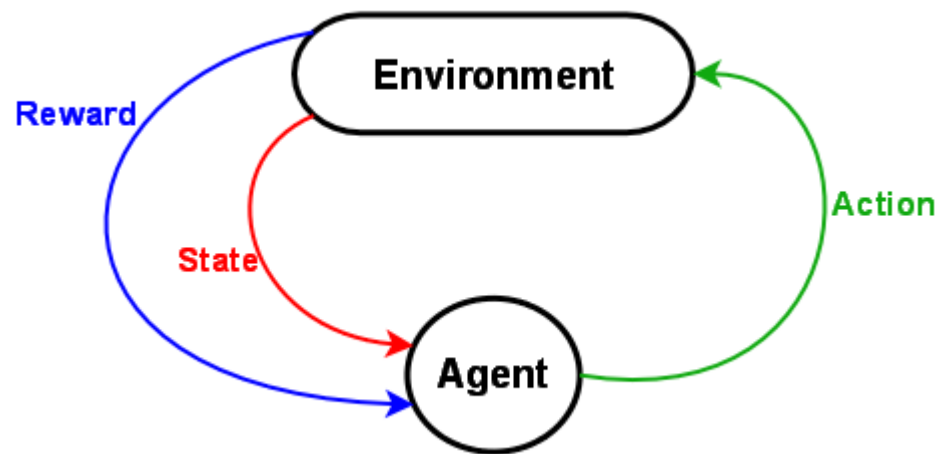
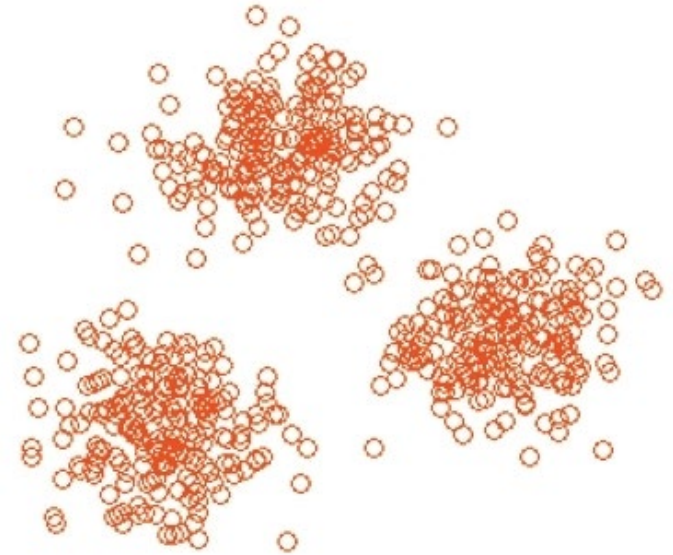
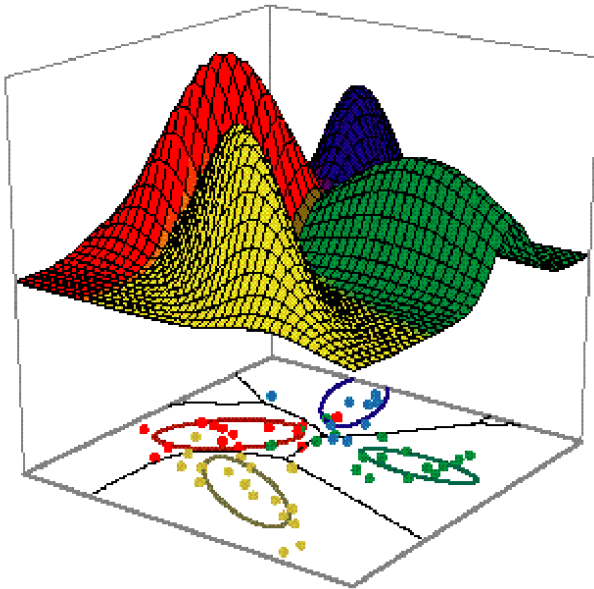
# 3 Learning Formalisms

- Learning algorithms can be classified based on the three types of feedback that the learner has access to:
1. **Supervised Learning**: One extreme → For every input, the learner is provided with a target. The environment tells the learner what its response is. The learner then compares its actual response to the target and adjusts its internal memory in such a way that it is more likely to produce the appropriate response the next time it receives the same input.
  2. **Unsupervised Learning**: On the other extreme → The learner receives no feedback from the world at all. Instead the learner's task is to re-represent the inputs in a more efficient way, as clusters or categories or using a reduced set of dimensions. Unsupervised learning is based on the similarities and differences among the input patterns. It does not result directly in differences in overt behavior because its "outputs" are really internal representations.

# 3 Learning Formalisms

- 3. Reinforcement Learning:** A third alternative, much closer to supervised than unsupervised learning → The learner receives feedback about the appropriateness of its response. For correct responses, RL resembles supervised learning: in both cases, the learner receives information that what it did is appropriate. However, the two forms of learning differ significantly for errors.
- Supervised learning lets the learner know exactly what it should have done.
  - Reinforcement Learning only says that the behavior was inappropriate and (usually) how inappropriate it was.

# 3 Learning Formalisms

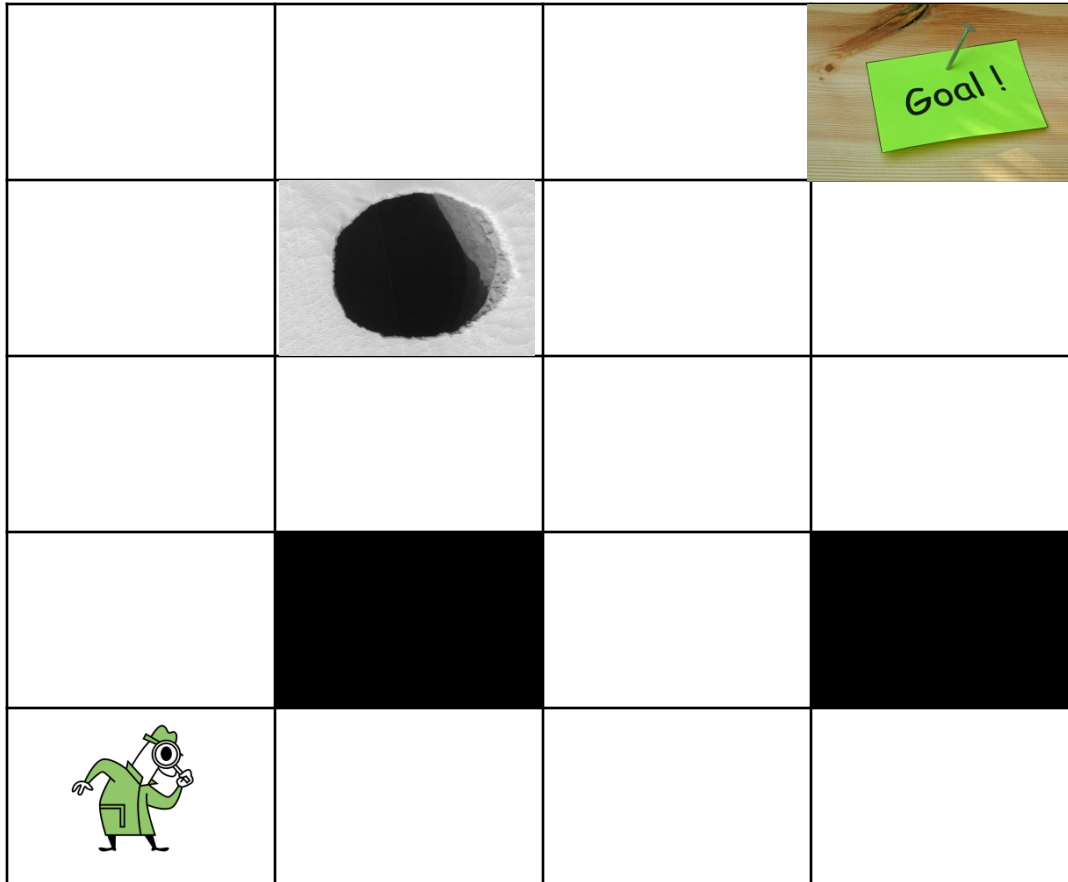


# Reinforcement Learning

- Consider learning to choose actions, like:
  - Robot learning to dock on battery charger
  - Learning to choose actions to optimize factory output
  - Learning to play Backgammon
- Note several problem characteristics:
  - Delayed reward
  - Opportunity for active exploration
  - Possibility that state only partially observable
  - Possible need to learn multiple tasks with same sensors/effector
  - Probabilistic Action Outcomes

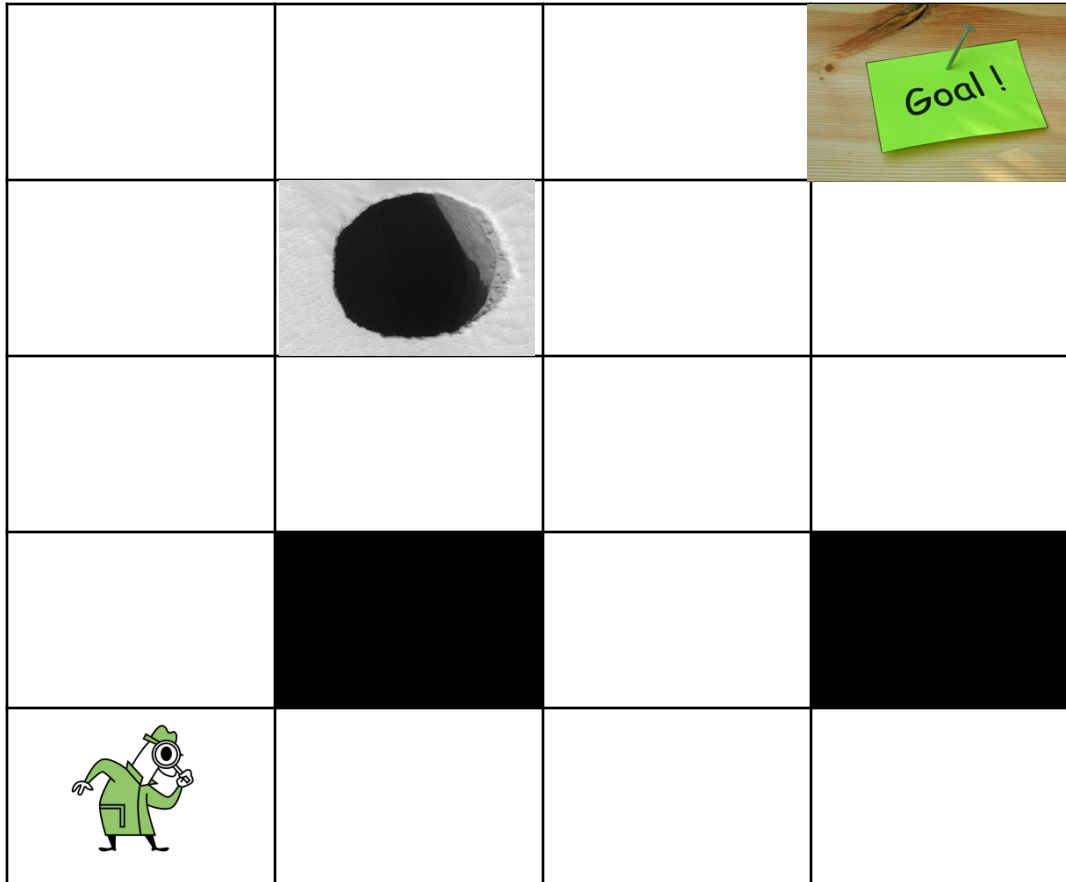
# Example domain

(Design this for your next Homework)



# Example domain

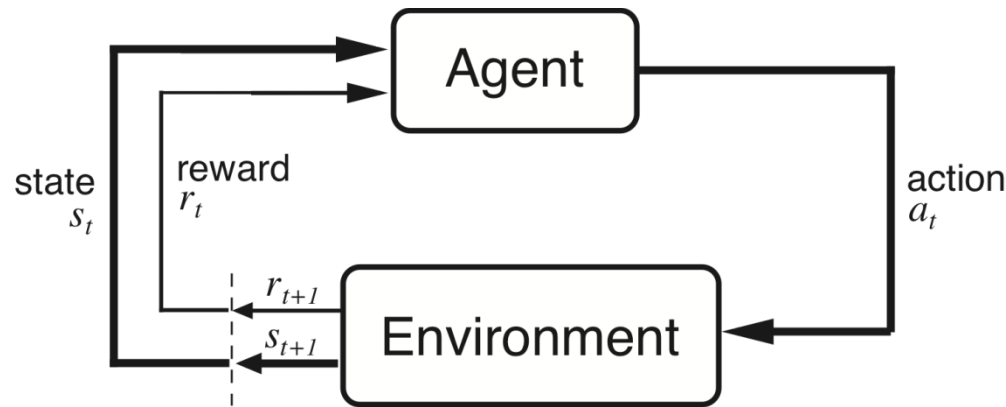
(Design this for your next Homework)



Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓



# The Agent-Environment Interface



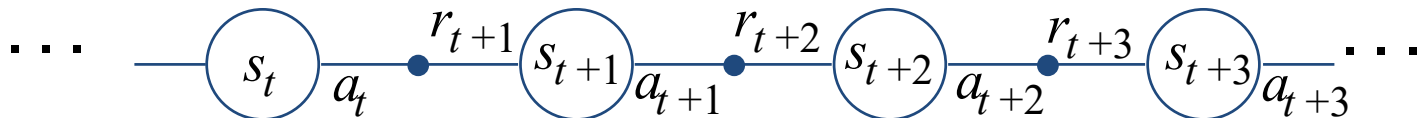
Agent and environment interact at discrete time steps:  $t = 0, 1, 2, K$

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

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

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

and resulting next state:  $s_{t+1}$



# What does the agent learn?

Straight line plan:

$a_1, a_2, a_3, \dots$

No guarantee! The actions have probabilistic effects.  
For example, when picking up an object, there is a non-zero chance of dropping it.

Policy:  $\pi(s) = a$

Map from states to actions.

Agent tries to learn the optimal policy. But optimal in terms of what?

# Returns

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

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?

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

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

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

**Non-Episodic tasks:** No episodes. Infinite game. e.g. a fire engine trying putting out fires as they arise.

# Returns

**Finite Horizon:** Fixed T

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

**Infinite Horizon:** T is infinite

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

Problem: All policies have infinite returns.

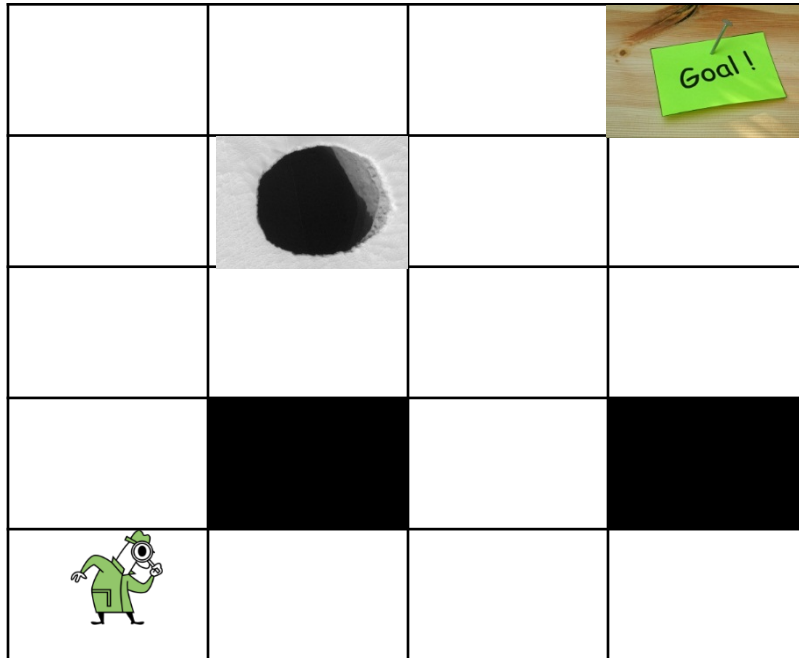
Solution: Discounted reward

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} \dots$$

Optimality Criterion: Find  $\pi$  that maximizes  $E[R_t]$

Note that the goal has changed from maximizing reward to expected reward

# Example domain - Rewards



## Example – Reward 1

-1 for every step taken  
+10 for reaching the goal  
-50 for falling in the pit

## Example – Reward 2

0 for every step taken  
+10 for reaching the goal  
-50 for falling in the pit

What is the difference in optimal policy between Reward 1 scheme and Reward 2 scheme?

# Markov Decision Process

$$M = \langle S, A, P, R \rangle$$

- ◆ S: Set of fully observable states
- ◆ A: Set of actions available to the agent
- ◆ P: Probabilistic state transition function  $P(s' | s, a)$  describing the probability of reaching state  $s'$  if the agent takes action  $a$  in state  $s$ .
- ◆ R: Reward function  $R(s)$  describing the immediate reward the agent achieves for being in a state.

**Goal:** Find  $\pi$  that maximizes the total expected discounted reward

$$\operatorname{argmax}_{\pi} E_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right]$$

# The Markov Property

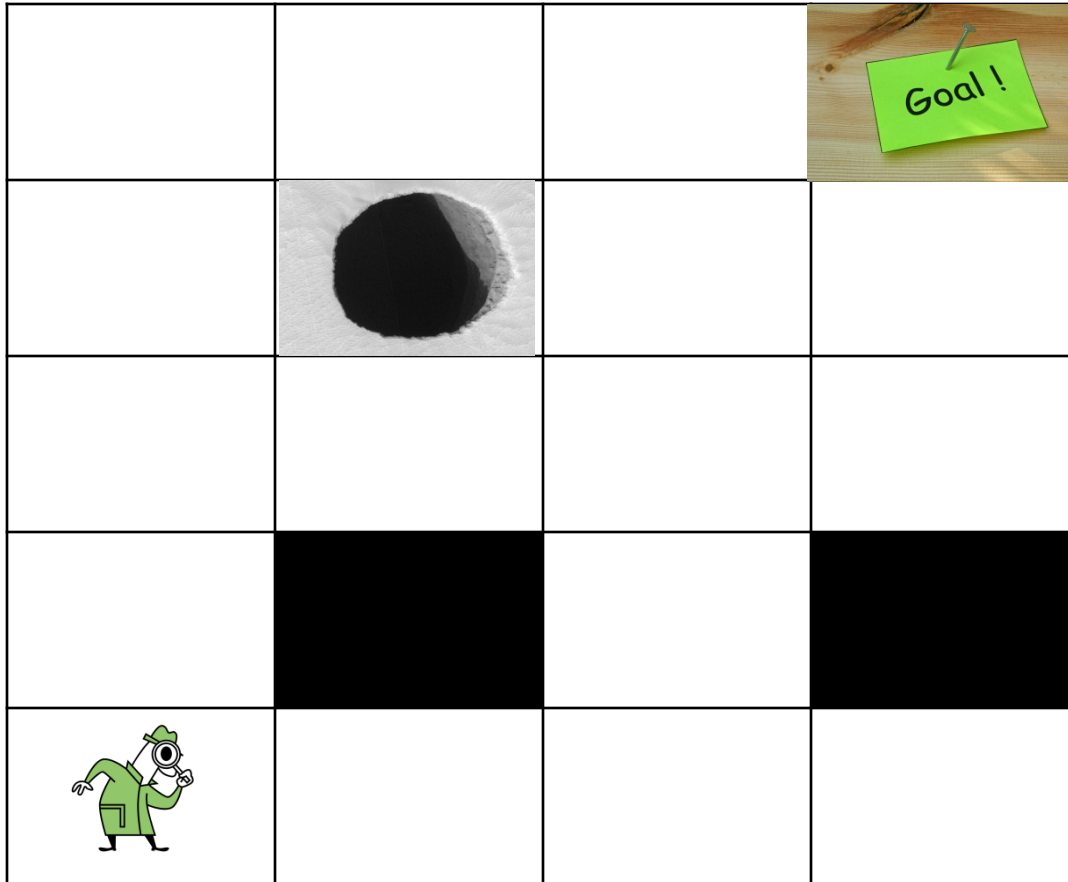
- “the state” at step  $t$ , means whatever information is available to the agent at step  $t$  about its environment. – Snapshot of the world
- The state can include immediate “sensations”, highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state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

## **Markov Property:**

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

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


# Example domain



Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓



# Example domain

			+10
	-50		
			

Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓

# 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  $\pi$**

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

- Informally the value of a state indicates how much better it is to be in that state than other states, when following the policy
- The **value of a state-action pair** is the expected return starting from that state, taking that action, and thereafter following  $\pi$ :

**State-action value function for policy  $\pi$**

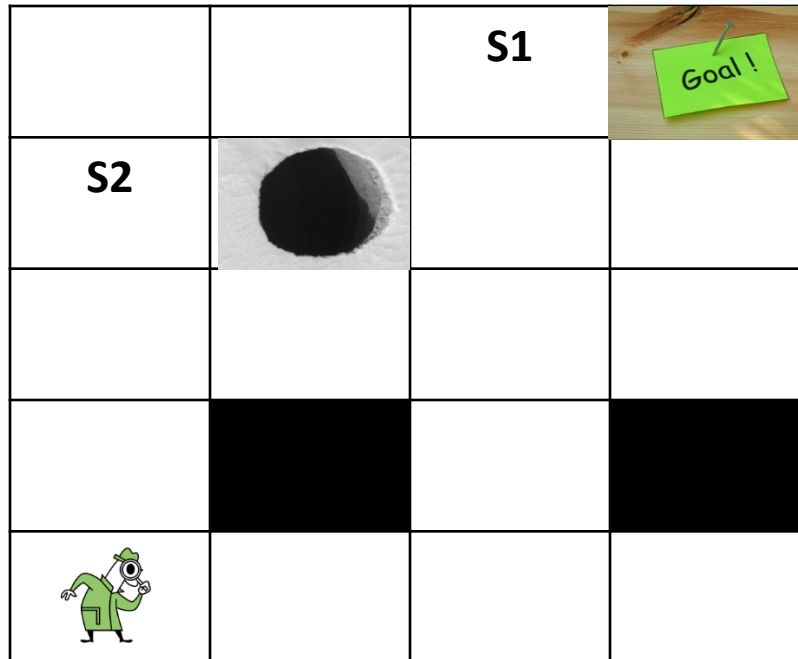
$$Q_{\pi}(s, a) = E_{\pi} \{R_t | s_t = s, a_t = a\} = E_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a \right\}$$

# Example domain – Values

Which of the states will have a higher value? S1 or S2?

Which action will have a higher value in S1?  $\longrightarrow$  or  $\downarrow$  ?

Which action will have a higher value in S2?  $\longrightarrow$  or  $\uparrow$  ?



# Bellman Equation for a Policy $\pi$

The basic idea:

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

So:

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

Or, without the expectation operator:

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s')$$

# Example domain

What is the value of this policy?

↓	↓	↓	↓
↓	↓	↓	↓
↓	↓	↓	↓
↓		↓	
↓	↓	↓	↓

Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓

# Example domain

What is the value of this policy?

0	-45	0	10
0	-50	0	0
0	0	0	0
0		0	
0	0	0	0

Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓

# 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  $\pi^*$ .
- Optimal policies share the same **optimal state-value function**:

$$V^*(s) = \max_{\pi} V^\pi(s) \quad \text{for all } s \in S$$

# Example domain – One optimal policy

↓	→	→	↑
↓	→	→	↑
→	→	↑	↑
↑		↑	
↑	→	↑	←

Actions	Effects
→	0.6 → 0.4 ↓
←	1.0 ←
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓

We will see more policies later. In our homework, you will learn the optimal policy for this domain



# Bellman Optimality Equation

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

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} E \{ r_t + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \} \\ &= \max_{a \in A(s)} \left[ R(s) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right] \end{aligned}$$

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

# Bellman Optimality Equation

Similarly, the optimal value of a state-action pair is the expected return for taking that action and thereafter following the optimal policy

$$\begin{aligned} Q^*(s, a) &= E \left\{ r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a \right\} \\ &= R(s) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q^*(s', a') \end{aligned}$$

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

# Dynamic Programming

- DP is the solution method of choice for MDPs
  - Require complete knowledge of system dynamics (P and R)
  - Expensive and often not practical
  - Curse of dimensionality
  - Guaranteed to converge!
- RL methods: online approximate dynamic programming
  - No knowledge of P and R
  - Sample trajectories through state space
  - Some theoretical convergence analysis available

# Policy Evaluation

**Policy Evaluation:** for a given policy  $\pi$ , compute the state-value function  $V^\pi$

Recall:

$$\begin{aligned}\text{State value function: } V^\pi(s) &= E_\pi \{R_t | s_t = s\} \\ &= E_\pi \{r_t + \gamma V^\pi(s_{t+1}) | s_t = s\}\end{aligned}$$

$$\text{Bellman Equation: } V^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s')$$

- System of  $|S|$  simultaneous linear equations
- Solve iteratively

# Policy Improvement

Suppose we have computed  $V^\pi$  for a policy  $\pi$ .

For a given state  $s$ , would it be better to do an action  $a \neq \pi(s)$  ?

The value of performing action  $a$  in state  $s$  is:

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

It is better to switch to action  $a$  for state  $s$  if and only if

$$Q^\pi(s, a) > V^\pi(s)$$

# Policy Improvement Cont.

Do this for all states to get a new policy  $\pi'$  that is **greedy** with respect to  $V^\pi$  :

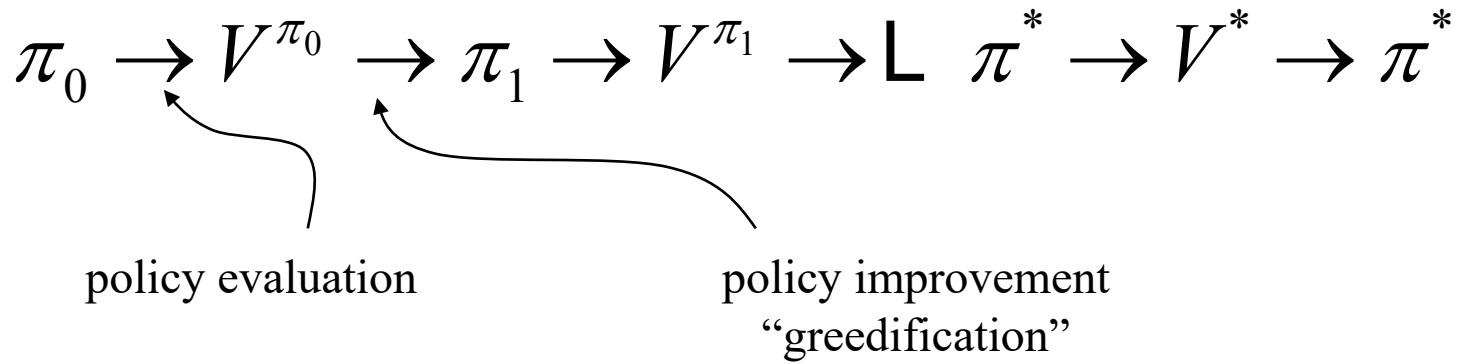
$$\pi'(s) = \operatorname{argmax}_a Q^\pi(s, a) = \operatorname{argmax}_a \left[ R(s) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s') \right]$$

Then  $V^{\pi'} \geq V^\pi$

Stop when  $\pi = \pi'$

Then  $\pi = \pi' = \pi^*$

# Policy Iteration



# Example domain

What is the value of this policy?

↓	↓	↓	↓
↓	↓	↓	↓
↓	↓	↓	↓
↓		↓	
↓	↓	↓	↓

Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓



# Example domain

What is the value of this policy?

0	-45	0	10
0	-50	0	0
0	0	0	0
0		0	
0	0	0	0

Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓

# Example domain – Improved policy

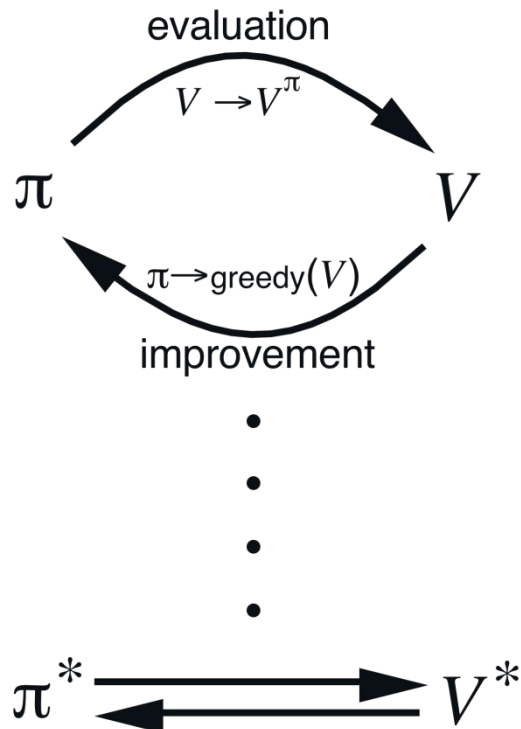
↑	←	→	↑
↑	↓	↓	↑
↑	↓	↑	↑
↑		↑	
↑	↑	↑	↑

Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓

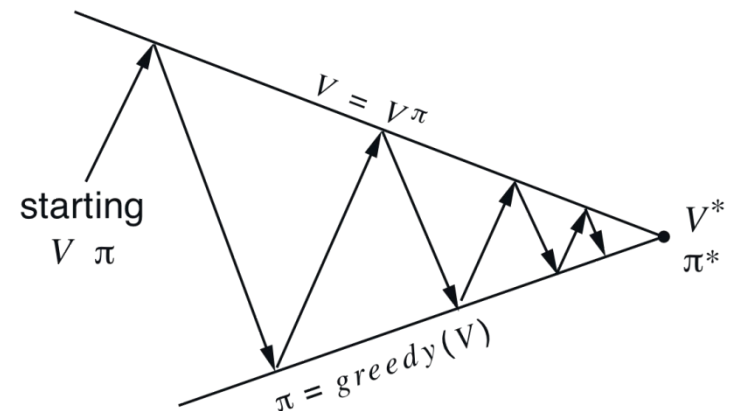
# Generalized Policy Iteration

**Generalized Policy Iteration (GPI):**

any interaction of policy evaluation and policy improvement,  
independent of their granularity.



A geometric metaphor for convergence of GPI:



# Value Iteration

Recall the **Bellman optimality equation**:

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} E \left\{ r_t + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\} \\ &= \max_{a \in A(s)} \left[ R(s) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right] \end{aligned}$$

We can convert it to an **full value iteration backup**:

$$V_{t+1}(s) \leftarrow \max_{a \in A(s)} \left[ R(s) + \gamma \sum_{s'} P(s'|s, a) V_t(s') \right]$$

Iterate until “convergence”

# Example domain

Optimal Value Function: Value of the optimal policy

11.33	10.16	37.73	51.18
12.64	-28.24	28.16	37.73
14.09	18.29	21.79	28.16
11.82		18.29	
9.91	11.82	14.09	12.64

Actions	Effects
→	0.6 → 0.4 ↓
←	← 1.0
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓

# Example domain – Optimal policy

↓	→	→	↑
↓	→	→	↑
→	→	↑	↑
↑		↑	
↑	→	↑	←

Actions	Effects
→	0.6 → 0.4 ↓
←	1.0 ←
↑	0.6 ↑ 0.4 ←
↓	1.0 ↓

# What is RL?

Imagine playing a new game whose rules you don't know; after a hundred or so moves, your opponent announces, "You lose".

-Russell and Norvig

Introduction to Artificial Intelligence

# What is different?

- Agent placed in an environment and must learn to behave optimally in it
- Assume that the world behaves like an MDP, except:
  - Agent can act but does **not know** the transition model
  - Agent observes its current state and its reward but does **not know** the reward function
- Goal: learn an optimal policy
- Challenge: How do you get to know the models (reward function and transition probabilities)
- Solution: Learn them as you explore the environment



# Why is RL difficult?

- Actions have non-deterministic effects – which are initially unknown and must be learned
- Rewards / punishments can be infrequent
  - Often at the end of long sequences of actions
  - How do we determine what action(s) were really responsible for reward or punishment? (credit assignment problem)
- World is large and complex

# Passive vs. Active learning

- Passive learning
  - The agent acts based on a fixed policy  $\pi$  and tries to learn how good the policy is by observing the world go by
  - Analogous to policy evaluation in policy iteration
- Active learning
  - The agent attempts to find an optimal (or at least good) policy by exploring different actions in the world
  - Analogous to solving the underlying MDP

# Model-Based vs. Model-Free RL

- *Model based approach to RL:*
  - learn the MDP model ( $T$  and  $R$ ), or an approximation of it
  - use it to find the optimal policy
- *Model free approach to RL:*
  - derive the optimal policy without explicitly learning the model

We will consider both types of approaches

# Passive Learning

## Adaptive Dynamic Programming

- Basically it learns the transition model  $\mathbf{T}$  and the reward function  $\mathbf{R}$  from the training sequences
- Based on the learned MDP ( $\mathbf{T}$  and  $\mathbf{R}$ ) we can perform policy evaluation (which is part of policy iteration previously taught)

# ADP

- ADP is a model based approach
  - ▶ Follow the policy for awhile
  - ▶ Estimate transition model based on observations
  - ▶ Learn reward function
  - ▶ Use estimated model to compute utility of policy

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s, a, s') V^{\pi}(s')$$

learned



- How can we estimate transition model  $T(s, a, s')$ ?
  - ▶ Simply the fraction of times we see  $s'$  after taking  $a$  in state  $s$ .

# Approach 3:

## Temporal Difference Learning

- Instead of calculating the exact utility for a state can we approximate it and possibly make it less computationally expensive?
- Yes we can! Using Temporal Difference (TD) learning

$$V_{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V_{\pi}(s')$$

- Instead of doing this sum over all successors, only adjust the utility of the state based on the successor observed in the trial.
- It does not estimate the transition model – model free

# Why is this correct? – Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers
  - E.g. to estimate the mean of a r.v. from a sequence of samples.

$$\begin{aligned}\hat{X}_{n+1} &= \frac{1}{n+1} \sum_{i=1}^{n+1} x_i = \frac{1}{n} \sum_{i=1}^n x_i + \frac{1}{n+1} \left( x_{n+1} - \frac{1}{n} \sum_{i=1}^n x_i \right) \\ &= \hat{X}_n + \frac{1}{n+1} (x_{n+1} - \hat{X}_n)\end{aligned}$$

average of n+1 samples

learning rate

sample n+1

- Given a new sample  $x(n+1)$ , the new mean is the old estimate (for  $n$  samples) plus the weighted difference between the new sample and old estimate

# Temporal Difference Learning (TD)

- TD update for transition from  $s$  to  $s'$ :

$$V^\pi(s) = V^\pi(s) + \alpha(R(s) + \gamma V^\pi(s') - V^\pi(s))$$

learning rate

New (noisy) sample of utility  
based on next state

- So the update is maintaining a “mean” of the (noisy) utility samples
- If the learning rate decreases with the number of samples (e.g.  $1/n$ ) then the utility estimates will eventually converge to true values!

$$V^\pi(s) = R(s) + \gamma \sum_{s'} T(s, a, s') V^\pi(s')$$



# Temporal Difference Update

When we move from state  $s$  to  $s'$ , we apply the following update rule:

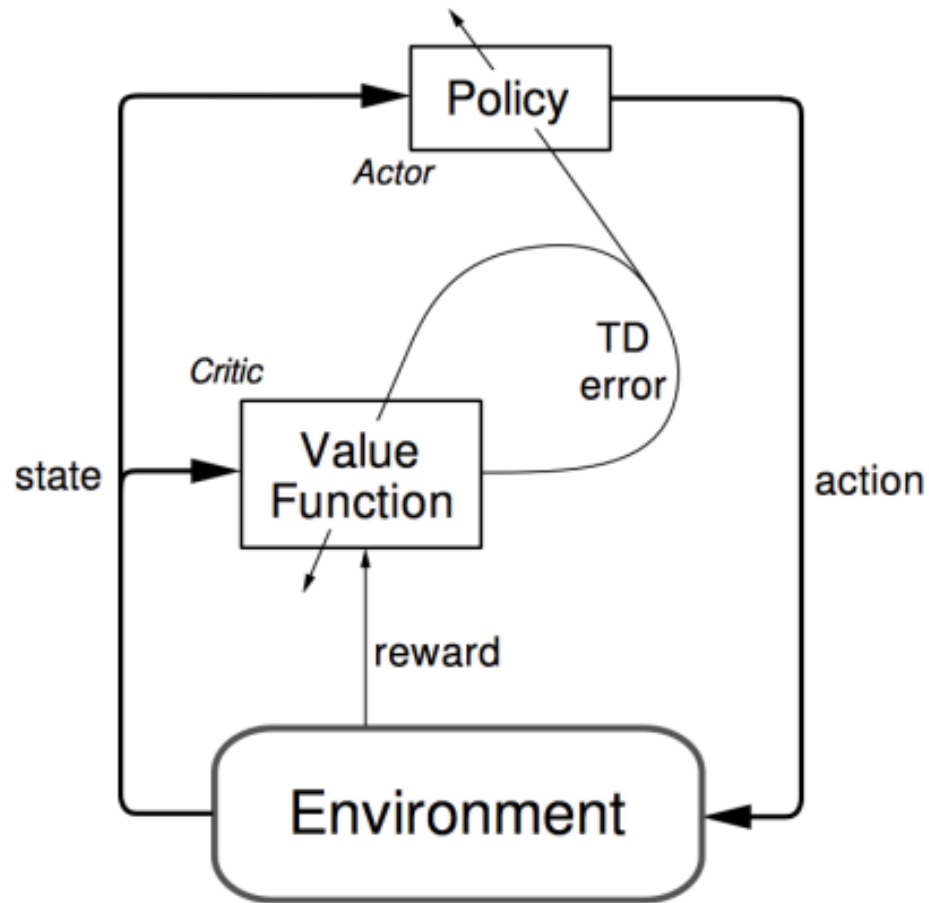
$$V^\pi(s) = V^\pi(s) + \alpha(R(s) + \gamma V^\pi(s') - V^\pi(s))$$

This is similar to one step of value iteration

We call this equation a “backup”

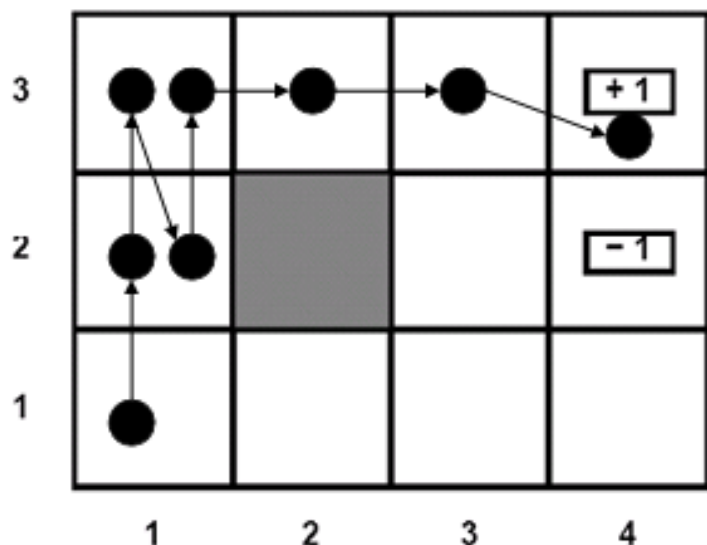
TD uses the observed states instead of the sum over all states that ADP uses.

# Temporal Difference Learning



# Passive learning

- Here we assume that the agent executes a fixed policy  $\pi$
- The goal is to evaluate how good  $\pi$  is, based on some sequence of trials performed by the agent – i.e., compute  $V^{\pi}(s)$



Learning  $V^{\pi}(s)$  does not lead to a optimal policy, why?

- the models are incomplete/inaccurate
- the agent has only tried limited actions, we cannot gain a good overall understanding of  $T$
- This is why we need active learning

# Goal of Active Learning

- Let's first assume that we still have access to some sequence of trials performed by the agent
  - The agent is not following any specific policy
  - We can assume for now that the sequences should include a thorough exploration of the space
  - We will talk about how to get such sequences later
- The goal is to learn an optimal policy from such sequences

# Naïve Approach

1. Act Randomly for a (long) time
  - ▲ Or systematically explore all possible actions
2. Learn
  - ▲ Transition function
  - ▲ Reward function
3. Use value iteration, policy iteration, ...
4. Follow resulting policy thereafter.

**Will this work?** Yes (if we do step 1 long enough and there are no “dead-ends”)

**Any problems?** We will act randomly for a long time before exploiting what we know.

# Revision of Naïve Approach

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work? No. Can get stuck in local minima.

What can be done?

# Exploration versus Exploitation

- Two reasons to take an action in RL
  - ▲ **Exploitation**: To try to get reward. We exploit our current knowledge to get a payoff.
  - ▲ **Exploration**: Get more information about the world. How do we know if there is not a pot of gold around the corner.
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic intuition behind most approaches:
  - ▲ Explore more when knowledge is weak
  - ▲ Exploit more as we gain knowledge

# Active RL Exploration

- Several strategies exist for exploration vs exploitation trade-off
  - Greedy in the limit of infinite exploration – Choose the best action with probability  $p$  and explore with  $1-p$ . Increase  $p$  with time.
  - Boltzmann exploration: Choose an action with probability proportional to the  $q$  value of that action

$$\Pr(a | s) = \frac{\exp(Q(s, a) / T)}{\sum_{a' \in A} \exp(Q(s, a') / T)}$$

- Optimistic exploration: Agent always acts greedily according to a model that assumes all “unexplored” states are maximally rewarding



# Q-Learning

- Instead of learning the optimal value function  $V$ , directly learn the optimal Q function.
  - Recall  $Q(s,a)$  is the expected value of taking action  $a$  in state  $s$  and then following the optimal policy thereafter
- The optimal Q-function satisfies  $V(s) = \max_{a'} Q(s, a')$  which gives:

$$\begin{aligned} Q(s, a) &= R(s) + \gamma \sum_{s'} T(s, a, s') V(s') \\ &= R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s, a') \end{aligned}$$

- Given the Q function we can act optimally by selecting action greedily according to  $Q(s,a)$  without a model

How can we learn the Q-function directly?

# Q-learning – Model-free Learning

Bellman constraints on optimal Q-function:

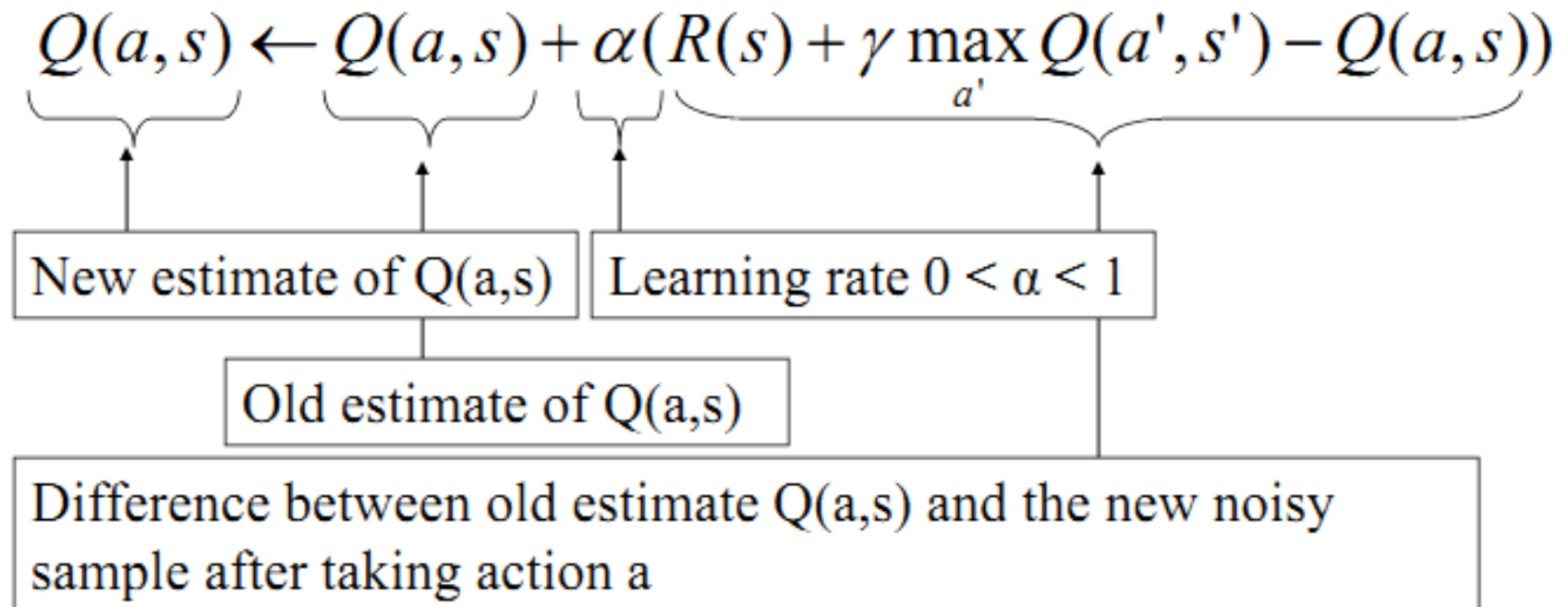
$$Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

- We can perform updates after each action just like in TD.
  - ▶ After taking **action a** in **state s** and reaching **state s'** do:  
(note that we directly observe reward  $R(s)$ )

$$Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \underbrace{\gamma \max_{a'} Q(s', a')}_{\text{(noisy) sample of Q-value based on next state}} - Q(s, a))$$

(noisy) sample of Q-value  
based on next state

# Q-learning: Model-free learning



# Q-learning: Estimating the Policy

Q-Update: After moving from state  $s$  to state  $s'$  using action  $a$ :

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

Note that  $T(s, a, s')$  does not appear anywhere!

Further, once we converge, the optimal policy can be computed without  $T$ .


This is a completely model-free learning algorithm.

# Q-learning

1. Start with initial Q-function (e.g. all zeros)
  2. Take action from **explore/exploit policy** giving new state  $s'$  (should converge to greedy policy, i.e. GLIE)
  3. Perform TD update
$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

$Q(s, a)$  is current estimate of optimal Q-function.
  4. Goto 2
- Does not require model since we learn Q directly!
  - Uses explicit  $|S| \times |A|$  table to represent Q
  - Explore/exploit policy directly uses Q-values
    - ▲ E.g. use Boltzmann exploration.

# Q-values after 5000 iterations

			+10
	-50		
			

$$\varepsilon = 0.5$$

$$\alpha = 0.1$$

$$\gamma = 0.9$$

Reduced  $\varepsilon = \frac{\varepsilon}{1 + \varepsilon}$  after every

10 iterations

65.6 65.6	72.8 72.9	80.9 80.9	99.9 99.9
58.9 -0.4	72.9 90	99.9 99.9	
65.6 59	22.8 20.5	80.9 9	90 21.6
51 -28	65.6 20.5	72.9 80.9	72.4 80.9
59 53.1	19 52	72.9 65.6	80.9 72.9
47 58.3	59 59	72.9 65.6	72.9 72.9
53 47.8		65.6 59	
43 53.1		59 53.1	
47.8 43	47.8 43	59 47.8	47.8 53.1
43 43	47.8 47.8	53 53	47.8 47.8

# (Some) Important Issues in Real World RL

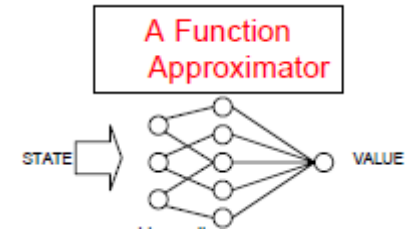
- Large State spaces
- Large number of objects
- Continuous states and continuous action spaces
- Partial Observability

# Large State Spaces

- When a problem has a large state space we can not longer represent the  $V$  or  $Q$  functions as explicit tables
- Even if we had enough memory
  - Never enough training data!
  - Learning takes too long
- What to do??



# Solution: Function Approximation



- Never enough training data!
  - Must **generalize** what is learned from one situation to other “similar” new situations
- Idea:
  - Instead of using large table to represent  $V$  or  $Q$ , use a parameterized function
    - The number of parameters should be small compared to number of states (generally exponentially fewer parameters)
  - Learn parameters from experience
  - When we update the parameters based on observations in one state, then our  $V$  or  $Q$  estimate will also change for other similar states
    - I.e. the parameterization facilitates generalization of experience
  - Examples are Linear functions, Neural networks etc.