

# **Introduction to Statistical Learning**

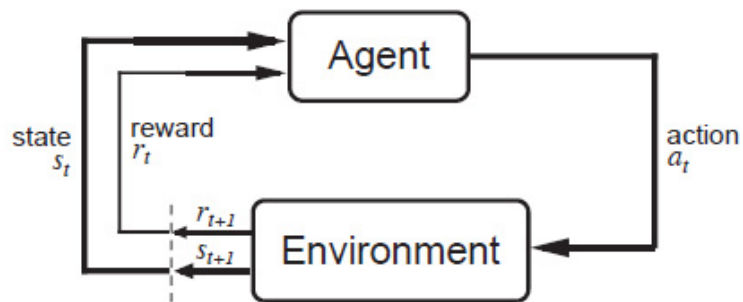
INF 552, Machine Learning for Data  
Informatics

University of Southern California

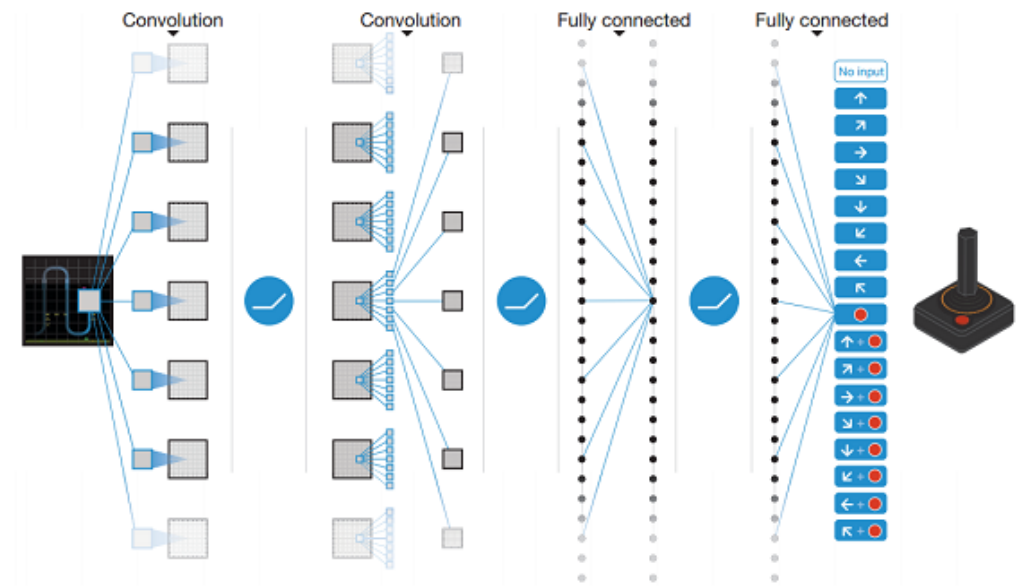
M. R. Rajati, PhD

# Lesson 13

## Reinforcement Learning



AlphaGo



# Overview

- Supervised Learning: Immediate feedback (labels provided for every input).
- Unsupervised Learning: No feedback (no labels provided).
- Reinforcement Learning: Delayed scalar feedback (a number called reward).

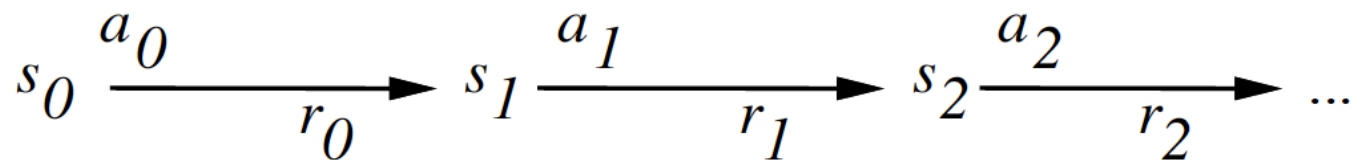
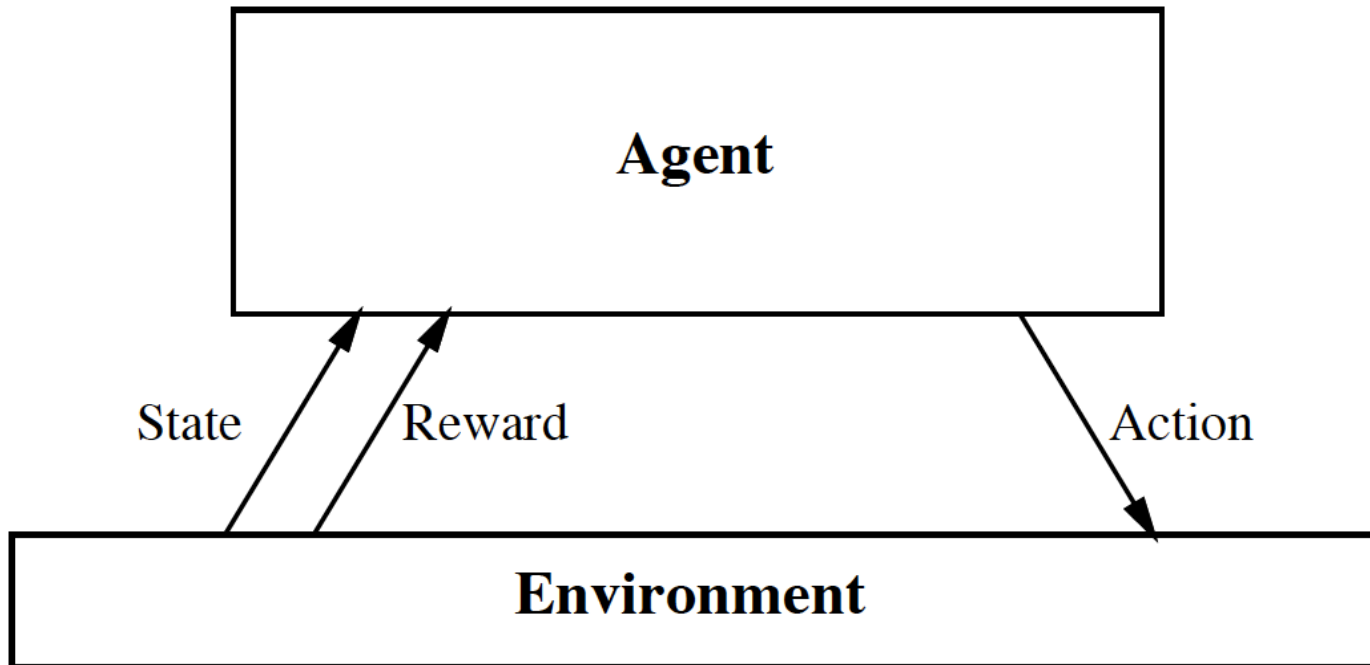
# Overview

- RL deals with agents that must sense & act upon their environment.
- This combines classical agent-based AI and machine learning techniques.  
It is a very comprehensive problem setting.

# Overview

- Examples:
  - A robot cleaning my room and recharging its battery
  - Robot-soccer
  - How to invest in shares
  - Modeling the economy through rational agents
  - Learning how to fly a helicopter
  - Scheduling planes to their destinations
  - and so on

# The Big Picture



Your action influences the state of the world which determines its reward

# Complications

- The outcome of your actions may be uncertain
- You may not be able to perfectly sense the state of the world
- The reward may be stochastic.
- Reward is delayed (i.e. finding food in a maze)

# Complications

- You may have no clue (model) of how rewards are being paid off.
- The world may change while you try to learn it
- How much time do you need to explore uncharted territory before you exploit what you have learned?



# The Task

- To learn an optimal *policy* that maps states of the world to actions of the agent.  
I.e., if this patch of room is dirty, I clean it. If my battery is empty, I recharge it.

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

- What is it that the agent tries to optimize?  
Answer: the **total future discounted reward**:

# The Task

- What is it that the agent tries to optimize?

Answer: the **total future discounted reward**:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad 0 \leq \gamma < 1$$

Note: immediate reward is worth more than future reward.

What would happen to mouse in a maze with  $\gamma = 0$  ?

# Value Function

- Let's say we have access to the optimal value function that computes the total future discounted reward  $V^*(s)$
- What would be the optimal policy  $\pi^*(s)$ ?
- Answer: we choose the action that maximizes:

$$\pi^*(s) = \arg \max_a \left[ r(s, a) + \gamma V^*(\delta(s, a)) \right]$$

# Value Function

- We assume that we know what the reward will be if we perform action “ $a$ ” in state “ $s$ ”:

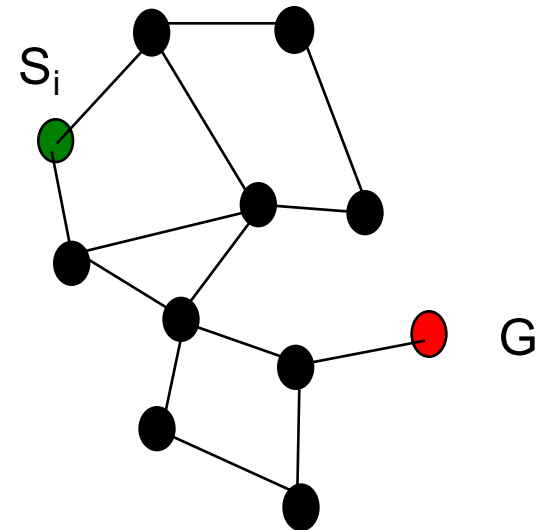
$$r(s, a)$$

- We also assume we know what the next state of the world will be if we perform action “ $a$ ” in state “ $s$ ”:

$$s_{t+1} = \delta(s_t, a)$$

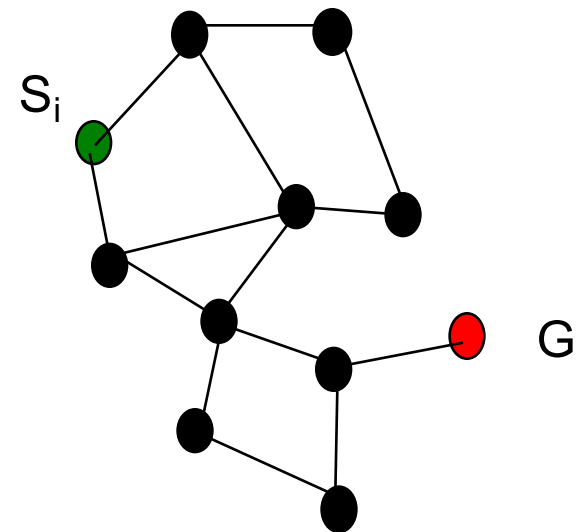
# Example I

- Consider some complicated graph, and we would like to find the shortest path from a node  $S_i$  to a goal node  $G$ .
- Traversing an edge will cost you “length edge” dollars.



# Example I

- The value function encodes the total remaining distance to the goal node from any node  $s$ , i.e.  
 $V(s) = \text{“}1 / \textit{distance}\text{”}$  to goal from  $s$ .
- If you know  $V(s)$ , the problem is trivial. You simply choose the node that has highest  $V(s)$ .

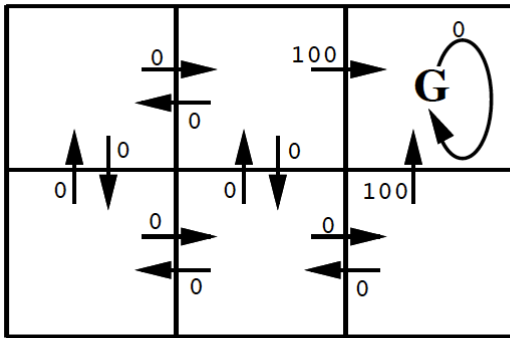


# Example II

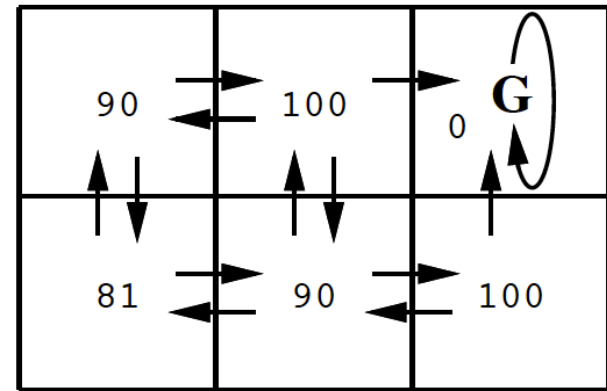
- A simple deterministic world.
- Each grid square represents a distinct state, each arrow a distinct action.
- The immediate reward function,  $r(s, a)$  gives reward 100 for actions entering the goal state  $G$ , and zero otherwise. Values of  $V^*(s)$  follow from  $r(s, a)$ , and the discount factor  $\gamma = 0.9$ .
- An optimal policy is also shown.

# Example II

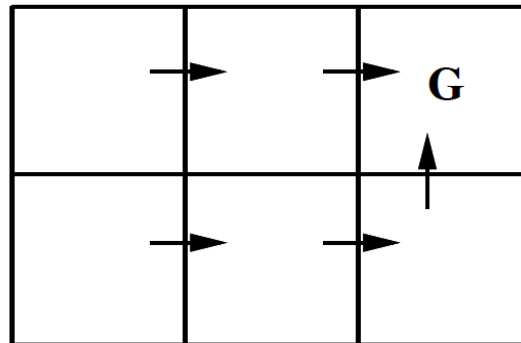
Find your way to the goal.



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



$V^*(s)$  values



One optimal policy

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad 0 \leq \gamma < 1$$



# Q-Function

- One approach to RL is then to try to estimate  $V^*(s)$ .

*Bellman Equation:*

$$V^*(s) \leftarrow \max_a \left[ r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

- However, this approach requires you to know  $r(s,a)$  and  $\delta(s,a)$ .
- This is unrealistic in many real problems. What is the reward if a robot is exploring mars and decides to take a right turn?

# Q-Function

- Fortunately we can circumvent this problem by exploring and experiencing how the world reacts to our actions. We need to *learn*  $r$  &  $\delta$ .

# Q-Function

- We want a function that directly learns good state-action pairs, i.e. what action should I take in this state. We call this  $Q(s,a)$ .

# Q-Function

- Let us define the evaluation function  $Q(s, a)$  so that its value is the maximum discounted cumulative reward that can be achieved starting from state  $s$  and applying action  $a$  as the first action.
- In other words, the value of  $Q$  is the reward received immediately upon executing action  $a$  from state  $s$ , plus the value (discounted by  $\gamma$ ) of following the optimal policy thereafter.

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

# Q-Function

- Why is this rewrite important? Because it shows that if the agent learns the  $Q$  function instead of the  $V^*$  function, it will be able to select optimal actions even when it has no knowledge of the functions  $r$  and  $\delta$ .

# Q-Function

- It need only consider each available action  $a$  in its current state  $s$  and choose the action that maximizes  $Q(s, a)$ .

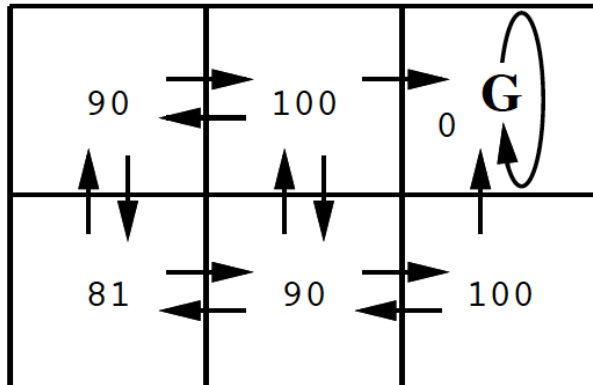
$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q(s, a)$$

$$V^*(s) = \max_a Q(s, a)$$

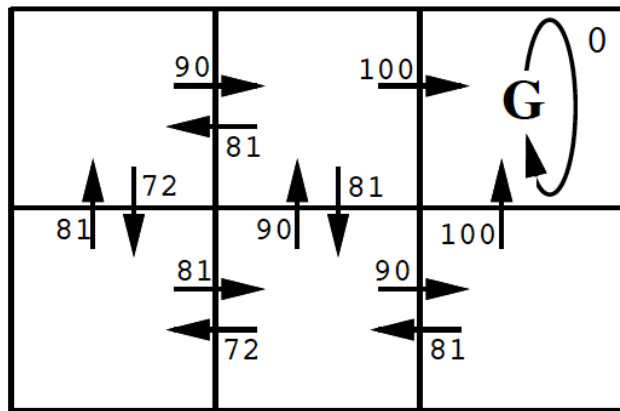
# Example

- To illustrate, the figure in the next slide shows the  $Q$  values for every state and action in the simple grid world.
- The  $Q$  value for each state-action transition equals the  $r$  value for this transition plus the  $V^*$  value for the resulting state discounted by  $\gamma$ .
- The optimal policy shown in the figure corresponds to selecting actions with maximal  $Q$  values.

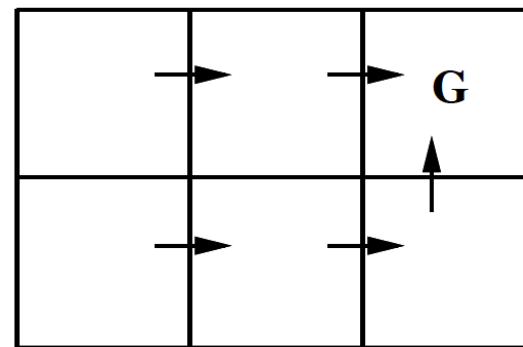
# Example II



$V^*(s)$  values



$Q(s, a)$  values



One optimal policy

Check that

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

$$V^*(s) = \max_a Q(s, a)$$



# Q-Learning

- Learning the  $Q$  function corresponds to learning the optimal policy. How can  $Q$  be learned?
- The key problem is finding a reliable way to estimate training values for  $Q$ , given only a sequence of immediate rewards  $r$  spread out over time.

# Q-Learning

- This can be accomplished through iterative approximation.
- To see how, notice the close relationship between  $Q$  and  $V^*$ ,

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

# Q-Learning

which allows rewriting the Q function as:

$$\begin{aligned} Q(s, a) &\equiv r(s, a) + \gamma V^*(\delta(s, a)) \\ &= r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \end{aligned}$$

This still depends on  $r(s, a)$  and  $\delta(s, a)$ ;  
**however,**

# Q-Learning

$$\begin{aligned} Q(s,a) &\equiv r(s,a) + \gamma V^*(\delta(s,a)) \\ &= r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a') \end{aligned}$$

this recursive definition of  $Q$  provides the basis for algorithms that iteratively approximate  $Q$ .

$\hat{Q}$  refers to the learner's estimate, or hypothesis, of the actual  $Q$  function.

# Q-Learning

$$\begin{aligned} Q(s, a) &\equiv r(s, a) + \gamma V^*(\delta(s, a)) \\ &= r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \end{aligned}$$

- Imagine the robot is exploring its environment, trying new actions as it goes.
- At every step it receives some reward “ $r$ ”, and it observes the environment change into a new state  $s'$  for action  $a$ .
- How can we use these observations,  $(s, a, s', r)$  to learn a model?

# Q-Learning

- The learner represents its hypothesis  $\hat{Q}$  by a large table with a separate entry for each state-action pair.
- The table entry for the pair  $(s, a)$  stores the value for  $\hat{Q}(s,a)$ , learner's current hypothesis about the actual but unknown value  $Q(s,a)$ .
- The table can be initially filled with random values (though it is easier to understand the algorithm if one assumes initial values of zero).

# Q-Learning

- The agent repeatedly observes its current state  $s$ , chooses some action  $a$ , executes this action, then observes the resulting reward  $r = r(s, a)$  and the new state  $s' = \delta(s, a)$ .

# Q-Learning

- It then updates the table entry for  $\hat{Q}(s,a)$  following each such transition, according to the rule:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a') \quad s' = s_{t+1}$$



# Q-Learning

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a') \quad s' = s_{t+1}$$

- This equation continually estimates  $Q$  at state  $s$  consistent with an estimate of  $Q$  at state  $s'$ , one step in the future: temporal difference (TD) learning.
- Note that  $s'$  is closer to goal, and hence more “reliable”, but still an estimate itself.

# Q-Learning Summary

*Q* learning algorithm

For each  $s, a$  initialize the table entry  $\hat{Q}(s, a)$  to zero.

Observe the current state  $s$

Do forever:

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

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

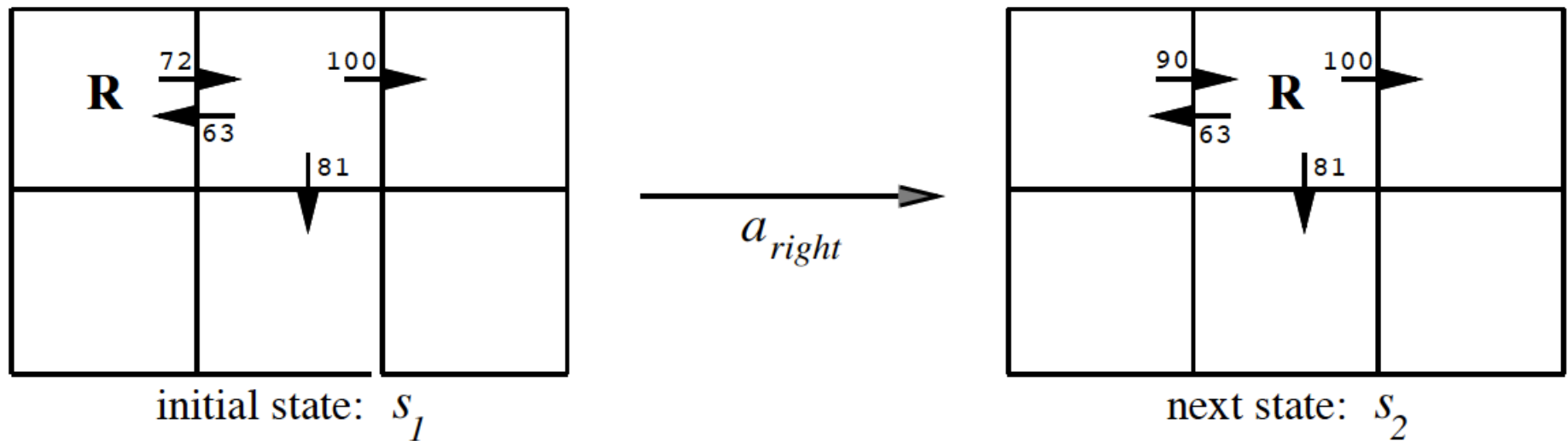
- $s \leftarrow s'$

# Q-Learning

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$
$$s' = s_{t+1}$$

- We do an update after each state-action pair. I.e., we are learning online!
- We are learning useful things about explored state-action pairs. These are typically most useful because they are likely to be encountered again.
- Under suitable conditions, these updates can actually be proved to converge to the real answer.

# Example: Q-Learning



$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{66, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

Q-learning propagates Q-estimates 1-step backwards