CS 760: Machine Learning
Reinforcement Learning

University of Wisconsin-Madison

Outline

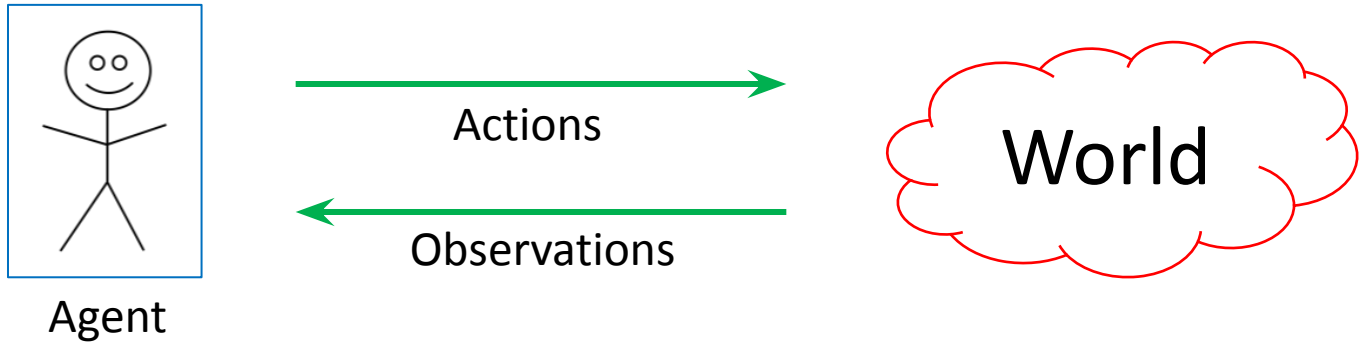
- **Review: Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration
- **Q Learning**
  - Q function, Q-learning, SARSA, approximation

Outline

- **Review: Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration
- **Q Learning**
  - Q function, Q-learning, SARSA, approximation

Review: General Model

We have an **agent** **interacting** with the **world**

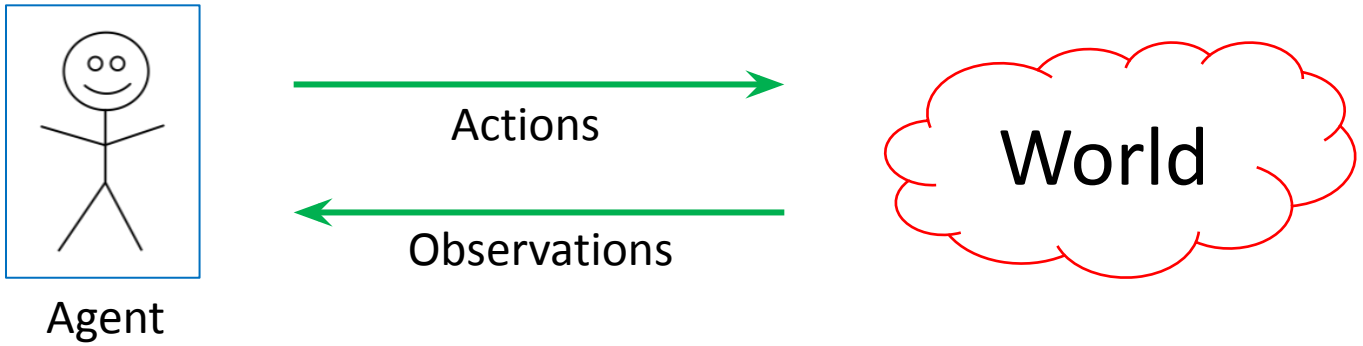


- Agent receives a reward based on state of the world
  - **Goal**: maximize reward / utility **($$$)**
  - Note: **data** consists of actions & observations
    - Compare to unsupervised learning and supervised learning

Building The Theoretical Model

Basic setup:

- Set of states S
- Set of actions A



- Information: at time t, observe state $s_t \in$ S. Get reward r_t
- Agent makes choice $a_t \in$ A. State changes to $s_{t+1,}$ continue

Goal: find a map from **states to actions** maximize rewards.

a “policy”

Markov Decision Process (MDP)

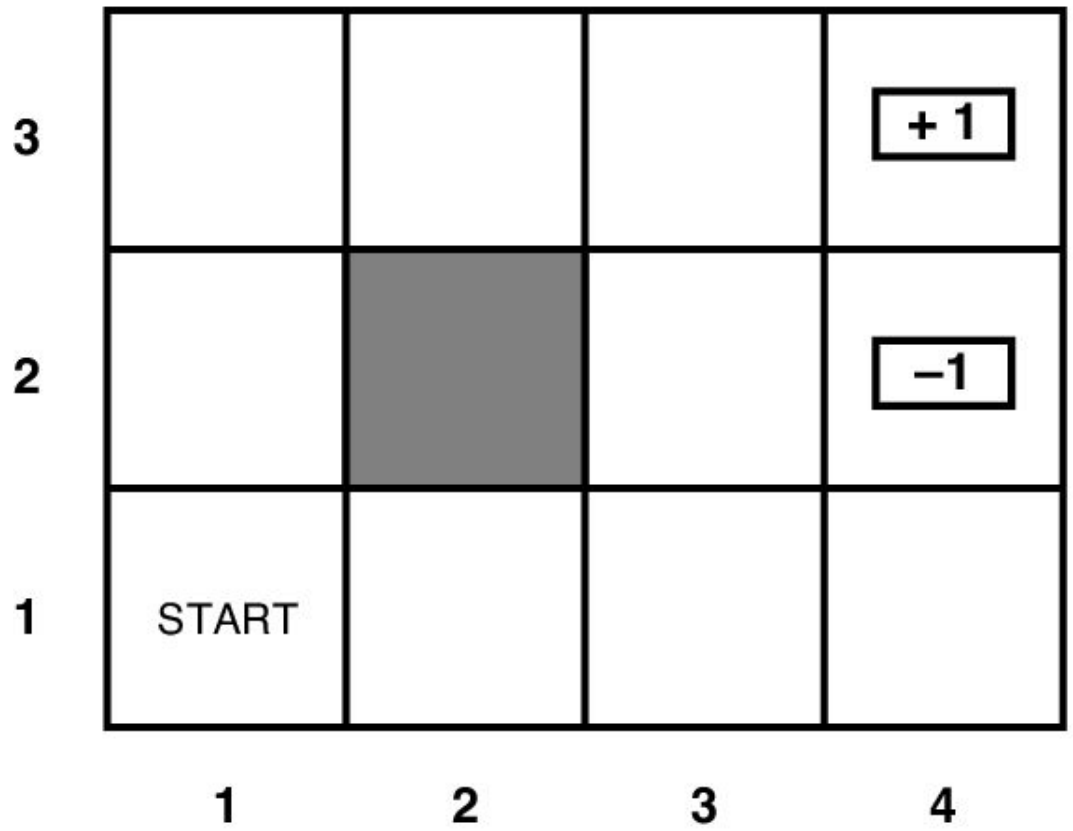
The formal mathematical model:

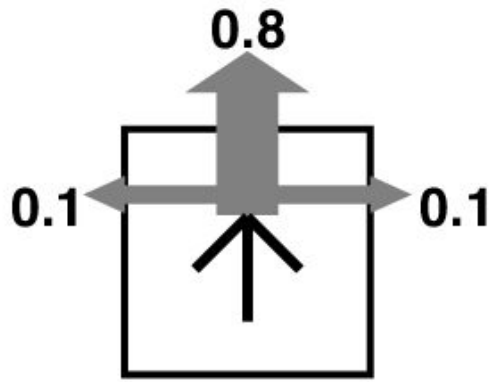
- **State set** S. Initial state s_0. **Action set** A
- **State transition model**: $P(s_{t+1}|s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy**: $\pi(s) : S \rightarrow A$ action to take at a particular state.

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

Grid World Abstraction

Note: (i) Robot is unreliable (ii) Reach target fast

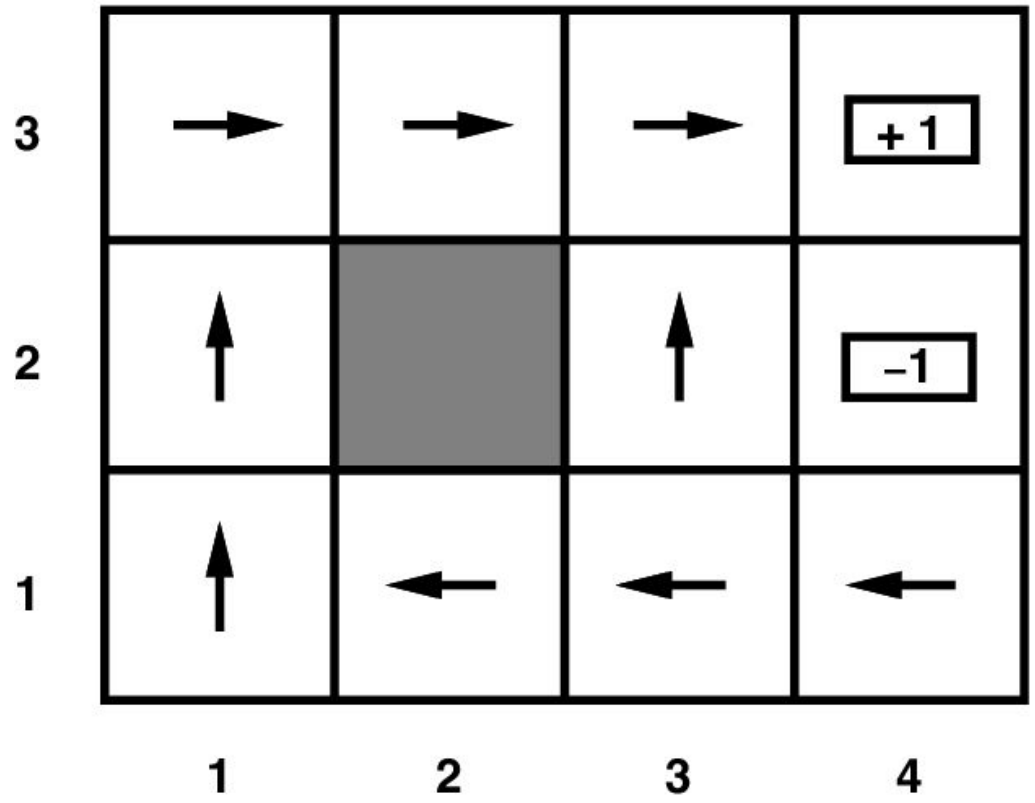


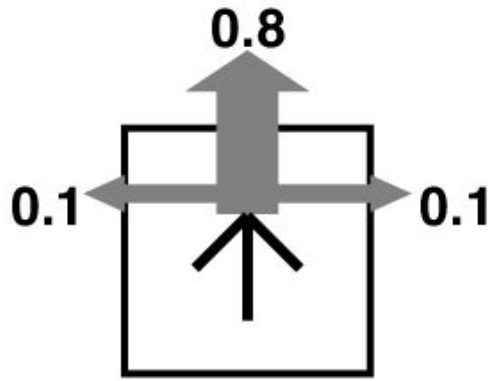


$r(s) = -0.04$ for every non-terminal state

Grid World Optimal Policy

Note: (i) Robot is unreliable (ii) Reach target fast





$r(s) = -0.04$ for every non-terminal state

Back to MDP Setup

The formal mathematical model:

- **State set** S. Initial state s_0. **Action set** A
- **State transition model**: $P(s_{t+1}|s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function: $r(s_t)$**
- **Policy**: $\pi(s) : S \rightarrow A$ action to take at a particular state.

How do we find the best policy?

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

Break & Quiz

Break & Quiz

Q 1.1 Consider an MDP with 2 states {*A, B*} and 2 actions: **"stay"** at current state and "**move**" to other state. Let **r** be the reward function such that **r**(*A*) = 1, **r**(*B*) = 0. Let γ be the discounting factor. What is the optimal policy $\pi(A)$ and $\pi(B)$? What are $V^*(A)$, $V^*(B)$?

- A. Stay, Stay, $1/(1-\gamma)$, 1
- B. Stay, Move, $1/(1-\gamma)$, $1/(1-\gamma)$
- C. Move, Move, $1/(1-\gamma)$, 1
- D. Stay, Move, $1/(1-\gamma)$, $\gamma/(1-\gamma)$

Break & Quiz

Q 1.1 Consider an MDP with 2 states {*A, B*} and 2 actions: **"stay"** at current state and "**move**" to other state. Let **r** be the reward function such that **r**(*A*) = 1, **r**(*B*) = 0. Let γ be the discounting factor. What is the optimal policy $\pi(A)$ and $\pi(B)$? What are $V^*(A)$, $V^*(B)$?

- A. Stay, Stay, $1/(1-\gamma)$, 1
- B. Stay, Move, $1/(1-\gamma)$, $1/(1-\gamma)$
- C. Move, Move, $1/(1-\gamma)$, 1
- **D. Stay, Move, $1/(1-\gamma)$, $\gamma/(1-\gamma)$**

Want to stay at A, or, if at B, move to A.

Starting at A, sequence A,A,A,… rewards 1, γ, γ^2,…. Start at B, sequence B,A,A,… rewards 0, γ, γ^2,….

Sums to $1/(1-\gamma)$ and $\gamma/(1-\gamma)$.

Outline

- **Review: Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration
- **Q Learning**
  - Q function, Q-learning, SARSA, approximation

Defining the Optimal Policy

For policy π, **expected utility** over all possible state sequences from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{starting from } s_0}} P(\text{sequence})U(\text{sequence})$$

Called the **value function** (for π, s_0)

Discounting Rewards

One issue: these are infinite series. **Convergence**?

- Solution

$$U(s_0, s_1 \ldots) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \ldots = \sum_{t \geq 0} \gamma^t r(s_t)$$

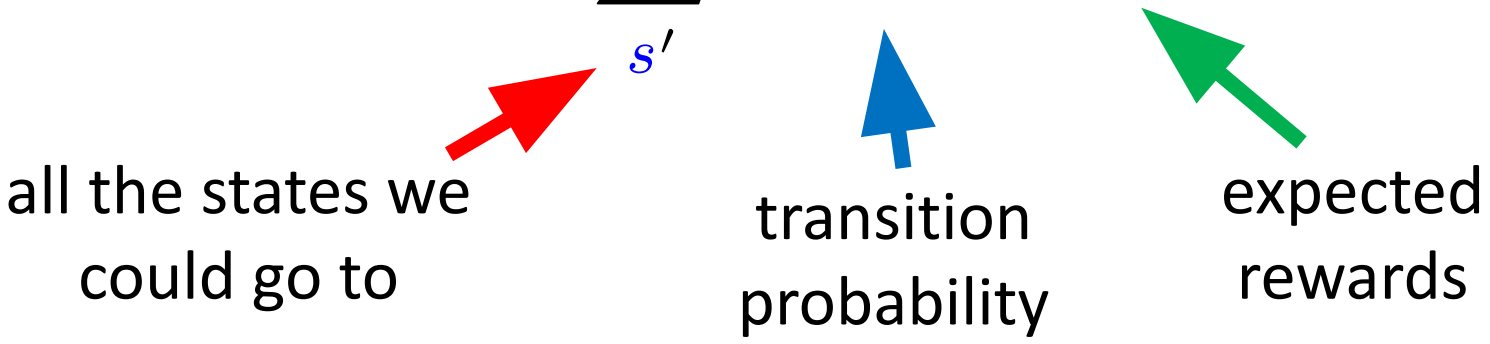
- Discount factor γ between 0 and 1
  - Set according to how important **present** is vs. **future**
  - Note: has to be less than 1 for convergence

From Value to Policy

Now that $V^{\pi}(s_0)$ is defined what a should we take?

- First, set V*(s) to be expected utility for **optimal** policy from s
- What's the expected utility of an action?
  - Specifically, action a in state s?

$$\sum_{s'} P(s'|s,a)V^*(s')$$



Obtaining the Optimal Policy

We know the expected utility of an action.

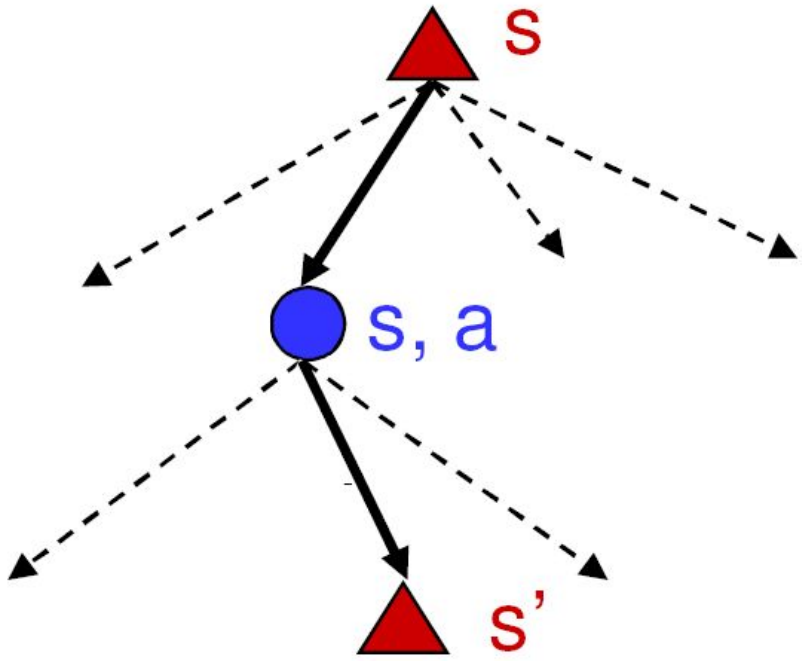
- So, to get the optimal policy, compute

$$\pi^*(s) = \text{argmax}_a \sum_{s'} P(s'|s,a)V^*(s')$$

all the states we could go to

transition probability

expected rewards



Credit L. Lazbenik

Slight Problem…

Now we can get the optimal policy by doing

$$\pi^*(s) = \text{argmax}_a \sum_{s'} P(s'|s,a)V^*(s')$$

- So we need to know $V^*(s)$.
  - But it was defined in terms of the optimal policy!
  - So we need some other approach to get $V^*(s)$.
  - Need some other **property** of the value function!

Bellman Equation

Let's walk over one step for the value function:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

current state reward

discounted expected future **rewards**

How do we derive the Bellman equation?

Start from the definition of the value of the **optimal** policy:

$$V^*(s) = r(s) + \gamma \max_a \sum_{\substack{\text{sequences} \\ \text{starting} \\ \text{from } s}} P(\text{sequence}|a) U(\text{sequence}|a)$$

$$= r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) \left(r(s') + \gamma \max_{a'} \sum_{\substack{\text{sequences} \\ \text{starting} \\ \text{from } s'}} P(\text{sequence}|a') U(\text{sequence}|a') \right)$$

$$= r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

Value Iteration

Q: how do we find $V^*(s)$?

- Why do we want it? Can use it to get the best policy
- Know: reward $r(s)$, transition probability $P(s'|s,a)$
- Also know $V^*(s)$ satisfies Bellman equation (recursion above)

A: Use the property. Start with $V_0(s)=0$. Then, update

$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$$

Policy Iteration

With value iteration, we estimate V*

- Then get policy (i.e., indirect estimate of policy)
- Could also try to get policies directly

- This is **policy iteration.** Basic idea:
  - Start with random policy π
  - Use it to compute value function V^{π} (for that policy)
  - Improve the policy: obtain π'

Policy Iteration: Algorithm

Policy iteration. Algorithm

- Start with random policy π
- Use it to compute value function V^{π} : a set of linear equations

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

- Improve the policy: obtain π'

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

- Repeat

Outline

- **Review: Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration
- **Q Learning**
  - Q function, Q-learning, SARSA, approximation

Q-Learning (**model-free** RL)

What if we don't know transition probability P($s'|s,a$)?

- Need a way to learn to act without it.
- **Q-learning**: get an action-utility function Q(s,a) that tells us the value of doing a in state s
- Note: $V^*(s) = \max_a Q(s,a)$
- Now, we can just do $\pi^*(s) = \arg\max_a Q(s,a)$
  - But need to estimate Q!

The Q function

Definition

$$Q_M^\pi(s,a) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t) \;\Big|\; \pi, s_0 = s, a_0 = a\Big]$$

Note: the first action is *a*, then follow π forever.

Bellman optimality equations for Q*(s,a)

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q^*(s',a')\right]$$

Q-Learning Iteration

How do we get Q(*s*,*a*)?

- Similar iterative procedure

learning rate

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

Idea: combine old value and new estimate of future value.

Note: We are using a policy π to take actions $a_t = \pi(s_t)$; this policy is based on Q!

Exploration Vs. Exploitation

General question!

- **Exploration:** take an action with unknown consequences
  - **Pros**:
    - Get a more accurate model of the environment
    - Discover higher-reward states than the ones found so far
  - **Cons**:
    - When exploring, not maximizing your utility
    - Something bad might happen
- **Exploitation:** go with the best strategy found so far
  - **Pros**:
    - Maximize reward as reflected in the current utility estimates
    - Avoid bad stuff
  - **Cons**:
    - Might also prevent you from discovering the true optimal strategy

Q-Learning: Epsilon-Greedy Policy

How to **explore**?

- With some 0<ε<1 probability, take a random action at each state, or else the action with highest Q(*s*,*a*) value.

$$a = \begin{cases} \text{argmax}_{a \in A} Q(s, a) & \text{uniform}(0, 1) > \epsilon \\ \text{random } a \in A & \text{otherwise} \end{cases}$$

Q-Learning: SARSA

An alternative:

- Just use the next action, no max over actions:

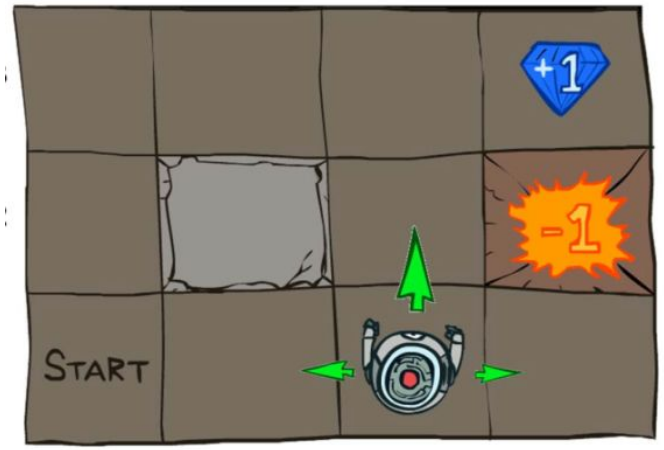
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- Called state–action–reward–state–action (**SARSA**)
- Can use with epsilon-greedy policy

Q-Learning Details

Note: if we have a **terminal** state, the process ends

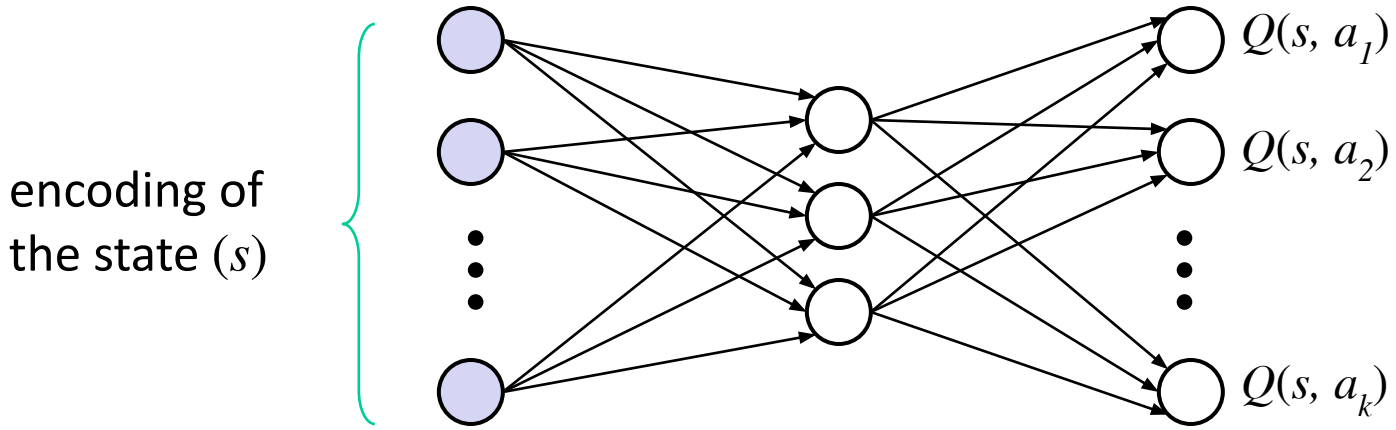
- An **episode**: a sequence of states ending at a terminal state
- Want to run on many episodes
- Slightly different Q-update for terminal states



Q-Learning – Compact Representations

Q-table can be quite large… might not even fit memory

- Solution: use some other representation for a more compact version. Ex: neural networks.

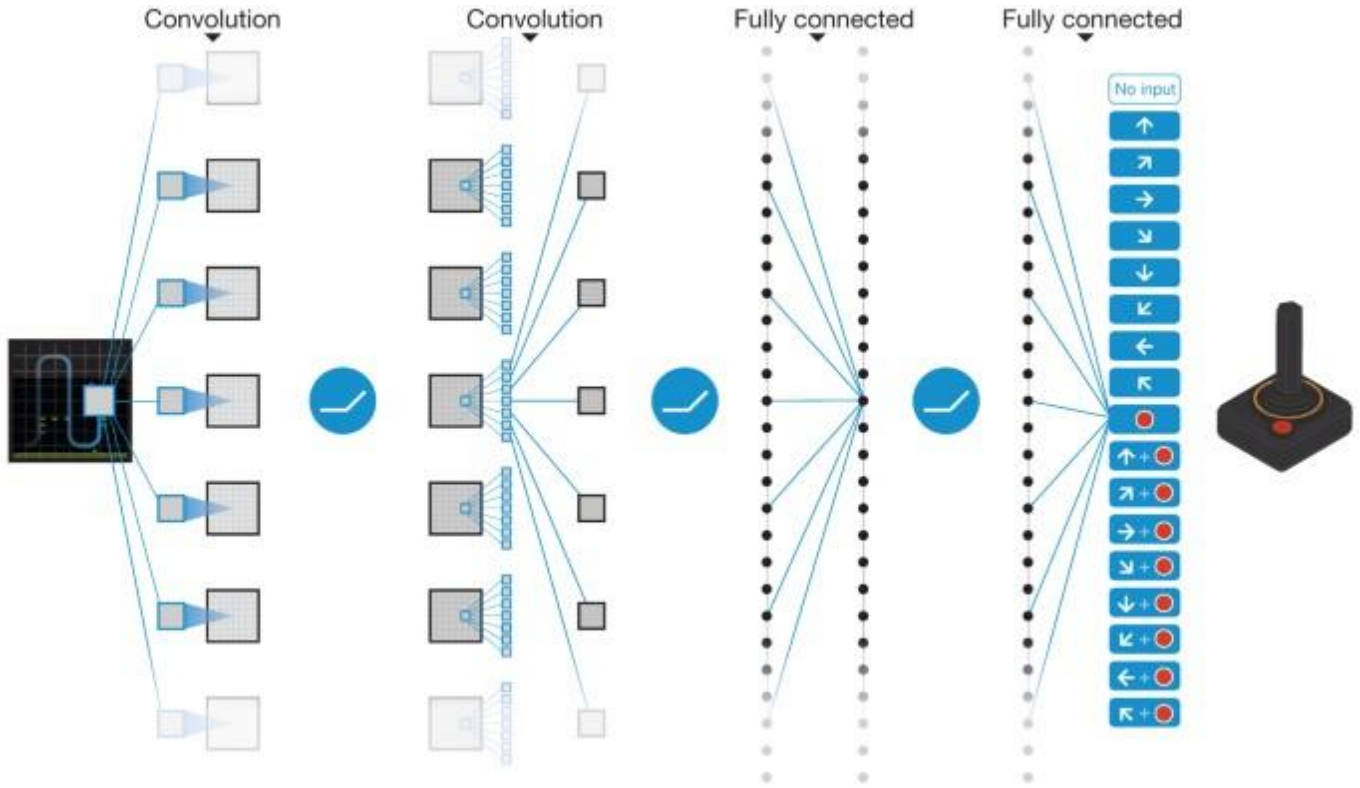


each input unit encodes a property of the state (e.g., a sensor value)

or could have <u>one net</u> for <u>each</u> possible action

Deep Q-Learning

How do we get Q(*s*,*a*)?



Mnih et al, "Human-level control through deep reinforcement learning"

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- A. Visit every state and try every action
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Prioritize exploitation over exploration.

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- **A. Visit every state and try every action**
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Prioritize exploitation over exploration.

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- **A. Visit every state and try every action**
- B. Perform at least 20,000 iterations. (No: this is dependent on the particular problem, not a general constant).
- C. Re-start with different random initial table values. (No: this is not necessary in general).
- D. Prioritize exploitation over exploration. (No: insufficient exploration means potentially unupdated state action pairs).

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Misha Khodak, Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, Fred Sala