CS 540 Introduction to Artificial Intelligence
Reinforcement Learning II

University of Wisconsin-Madison
Spring 2025

Announcements

- **Homework**:
  - HW9 due on **Wednesday April 23rd at 11:59 PM**
- **Final Exam:**
  - **May 7th 07:45 - 09:45 AM**
  - **Lecture 001 (MW 14:30 - 15:45): S429 Chemistry Building**
  - **Lecture 002 (TR 11:00 - 12:15): 1220 Microbial Sciences**
  - **Lecture 003 (TR 16:00 - 17:15): B10 Ingraham Hall**
  - Students with Mc Burney accommodations or alternate requests will be notified about the exam time and location.
  - More information coming soon!

- **Course evaluation**

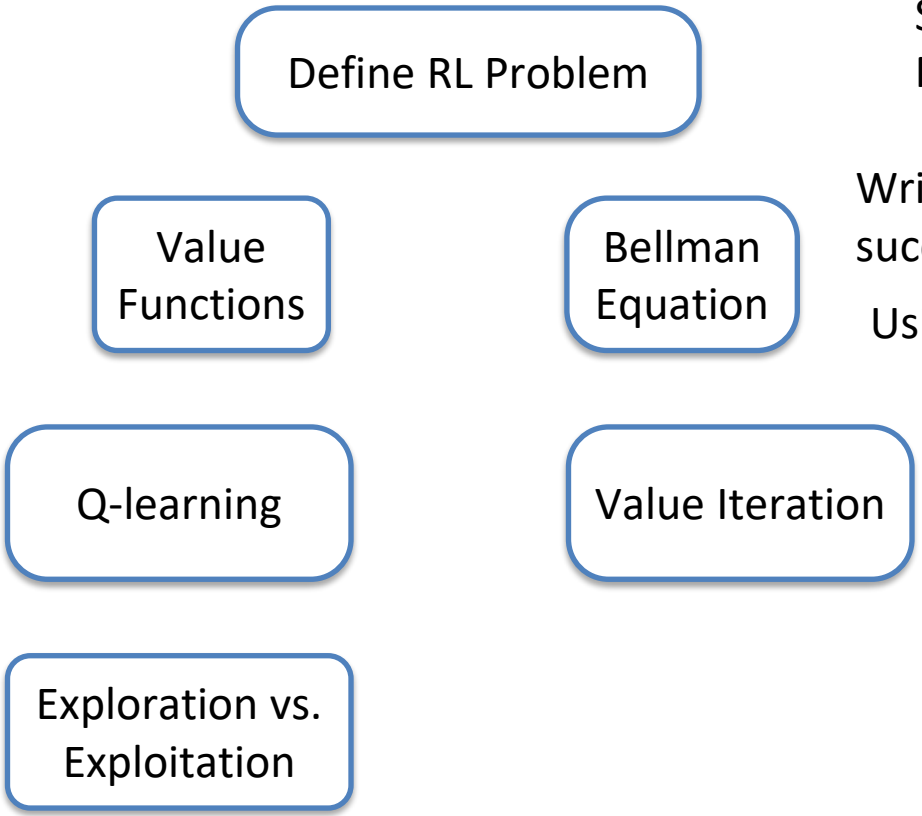
- Class roadmap:

Reinforcement Learning II
Advanced Search
Ethics and Trust in AI

Outline

- Review of reinforcement learning setting.
  - MDPs, value functions
- Bellman equations and dynamic programming
- From dynamic programming to Q-learning

Key Ideas in Reinforcement Learning



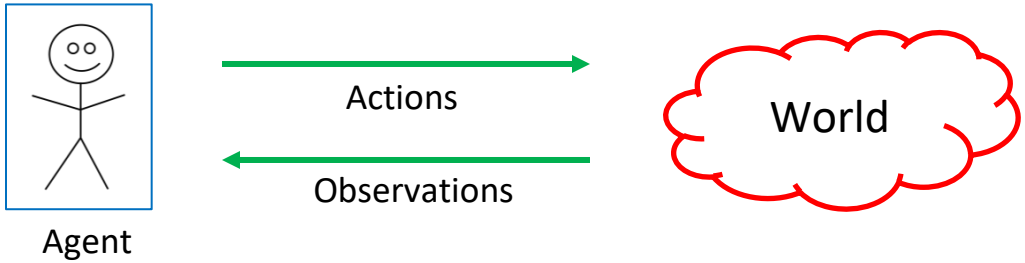
States, Actions, Transitions, Rewards, Markov property, discounting

Writing the value of one state in terms of successor states.

Using values to choose optimal actions.

Back to Our 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

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.

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

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

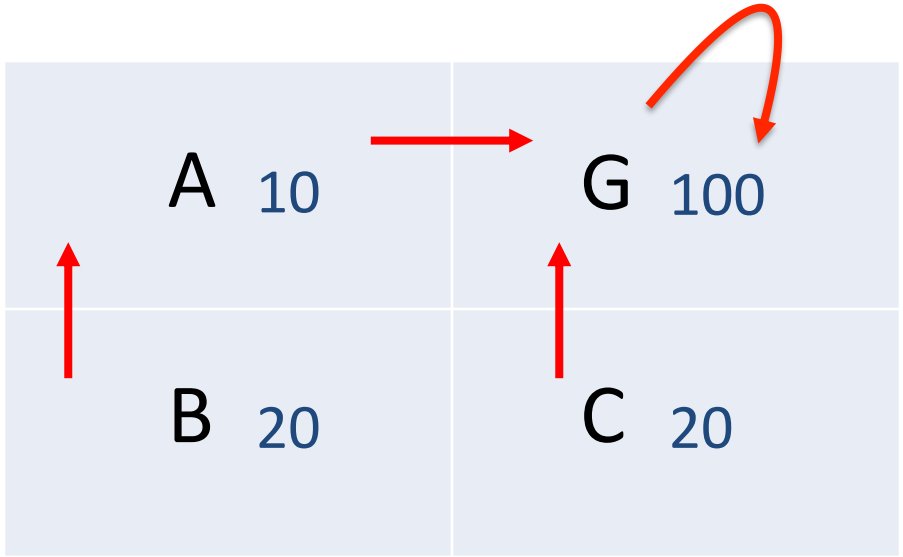
One issue: these are infinite series. **Convergence**?

- 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

Example



Deterministic transitions; $\gamma = 0.8$; policy shown with red arrows.

Values and Policies

- 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?

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

All the states we could go to

Transition probability

Expected rewards

Obtaining the Optimal Policy

Assume, we know the expected utility of an action.

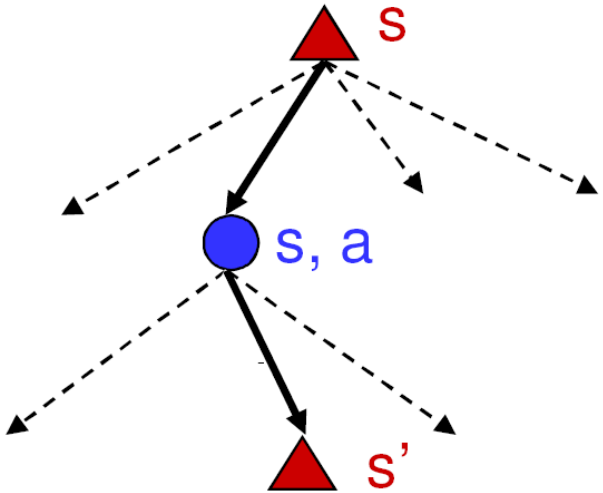
- So, to get the optimal policy, compute

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

All the states we could go to

Transition probability

Expected rewards

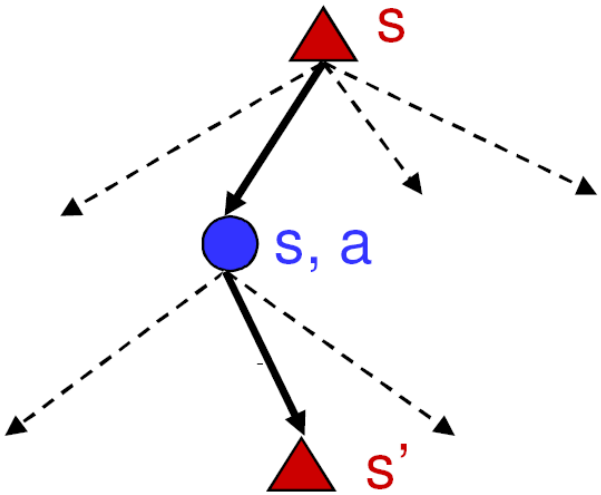


Credit L. Lazbenik

Bellman Equations

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

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



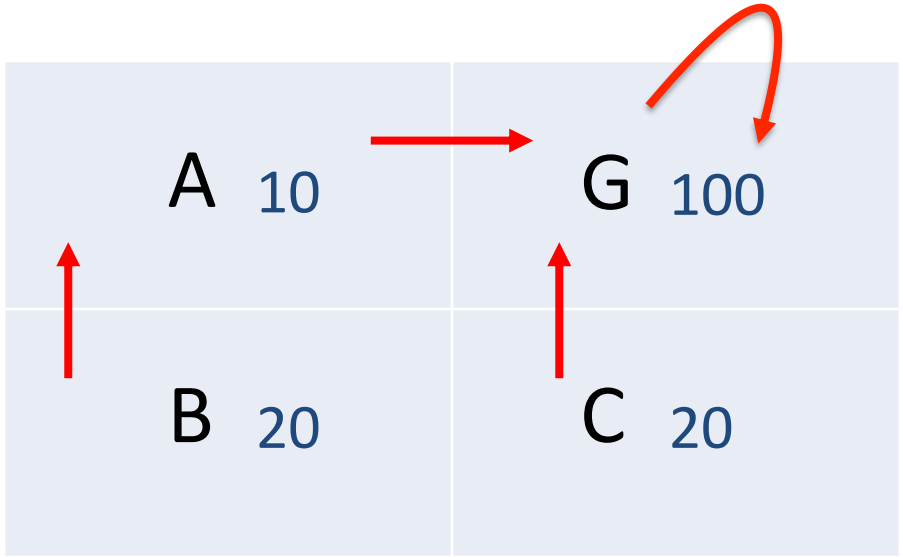
Credit L. Lazbenik

Current state reward

Discounted expected future **rewards**

Richard Bellman: Inventor of dynamic programming.

Example



Deterministic transitions; $\gamma = 0.8$; policy shown with red arrows.

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:

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

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

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

Q-Learning

- Our **next** reinforcement learning algorithm.
- Does not require knowing *r* or *P*. Learn from data of the form:$\{(s_t, a_t, r_t, s_{t+1})\}$.
- Learns an action-value function *Q**(*s*,*a*) that tells us the expected value of taking *a* in state *s*.
  - Note: $V^*(s) = \underset{a}{max} Q^*(s, a)$.
- Optimal policy is formed as $\pi^*(s) = \arg\underset{a}{max} Q^*(s, a)$

The Q*(s,a) function

- Starting from state s, perform (perhaps suboptimal) action a. THEN follow the optimal policy

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

- Equivalent to

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

Q-Learning Iteration

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

- Iterative procedure

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

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

Note: We are using a policy to take actions; based on the estimated Q!

Q-Learning

Estimate Q*(*s*,*a*) from data $\{(s_t, a_t, r_t, s_{t+1})\}$:

1. Initialize Q(.,.) arbitrarily (eg all zeros)
    1. Except terminal states $Q(s_{terminal},.)=0$
2. Iterate over data until Q(.,.) converges:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$

Learning rate

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 prevent you from discovering the true optimal strategy

Q-Learning: ε-Greedy Behavior Policy

Getting data with both **exploration and exploitation**

- With probability ε, take a random action; else the action with the highest (current) 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 Algorithm

Input: step size α, exploration probability ϵ

1. set $Q(s,a) = 0$ for all s, a.
2. For each episode:
3. Get initial state s.
4. While (s not a terminal state):
5. Perform a = ϵ-greedy(Q, s), receive r, s'
6. $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$
7. $s \leftarrow s'$
8. End While
9. End For

Explore: take action to see what happens.

Update action-value based on result.

Q-Learning: SARSA

An alternative update rule:

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

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

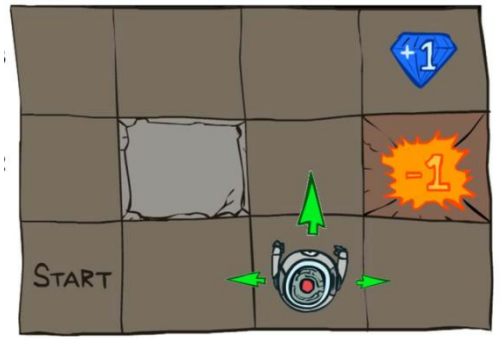
Learning rate

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

Q-Learning Details

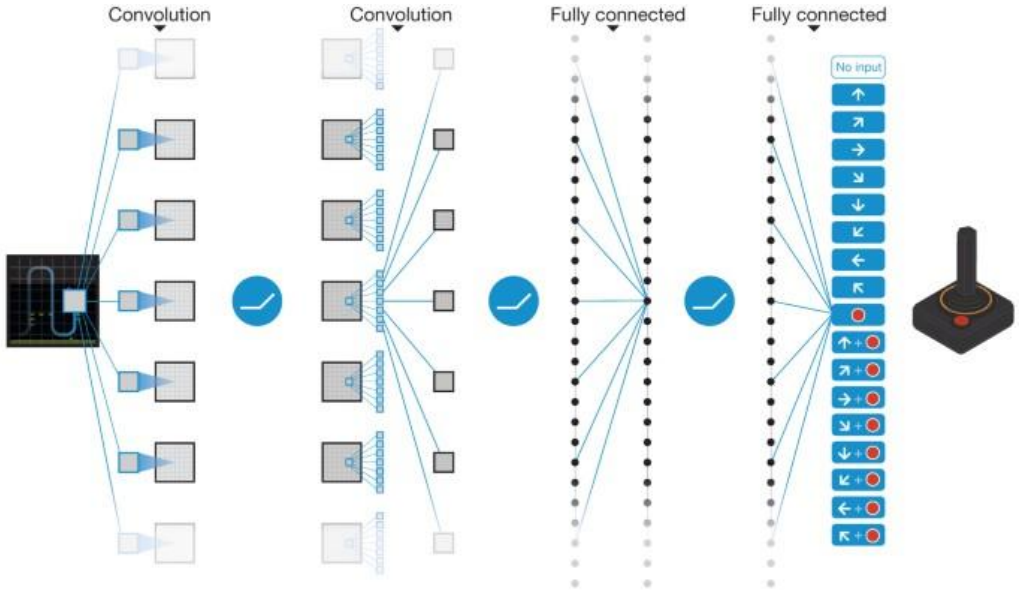
Note: if we have a **terminal** state, the process ends

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



Deep Q-Learning

How do we get Q(*s*,*a*) with a large number of states?



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

Deep Q-Learning

How do we get Q(*s*,*a*) with a large number of states?

- Deep Q-learning uses a neural network to approximate Q(s,a)
  - Let $Q_\theta : S \times A \to \mathbb{R}$ be the neural network with weights and biases denoted θ.
- Training is similar to supervised regression:
  - (s, a) as input and $y = r(s) + \gamma \max_{a'} Q_\theta(s', a')$ as label.
  - Note that output of the neural network is used in the label.
  - Loss function: $\mathcal{L}(\theta) = (y - Q_\theta(s, a))^2$

Break & Quiz

Q 2.2 For Q learning to converge to the true Q function, we must

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

Summary of RL

- Reinforcement learning setup
- Mathematical formulation: MDP
- Value functions & the Bellman equation
- Value iteration
- Q-learning