CS 760: Machine Learning
Reinforcement Learning II

Guest Lecturer: Adam Labiosa

University of Wisconsin-Madison

November 30, 2023

Announcements

- Homework 7 due December 7 at 9:30 am.
- Final exam: December 18 from 2:45 - 4:45 pm in the Social Sciences building.
- Course evaluations available until 12/13.

Lecture Goals

At the end of today's lecture, you will be able to:

1. Implement fundamental dynamic programming approaches to reinforcement learning.

2. Implement the q-learning algorithm.

Markov Decision Processes

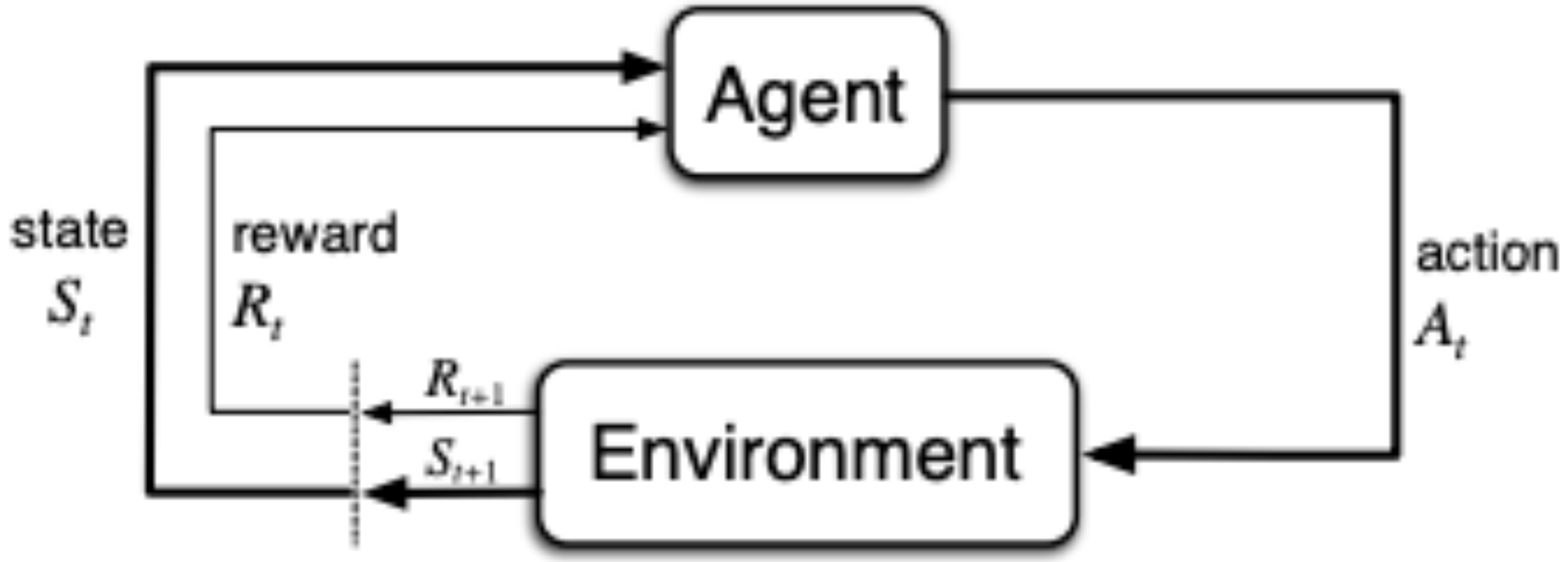
RL problems are formalized as Markov decision processes, $\langle \mathcal{S}, \mathcal{A}, r, p \rangle$:

- States: $s \in \mathcal{S}$
- Actions: $a \in \mathcal{A}$
- Rewards: $R \sim r(s, a)$
- State transitions: $S \sim p(\cdot \mid s, a)$
  - **Markov property:** next state only depends on current state and action taken.
- Goal: Find a policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, that maximizes cumulative reward.

Today's lecture, we will assume that p and r are known to us.

Data in Reinforcement Learning

Agent learns from the sequence of data seen while acting in task Markov decision process:

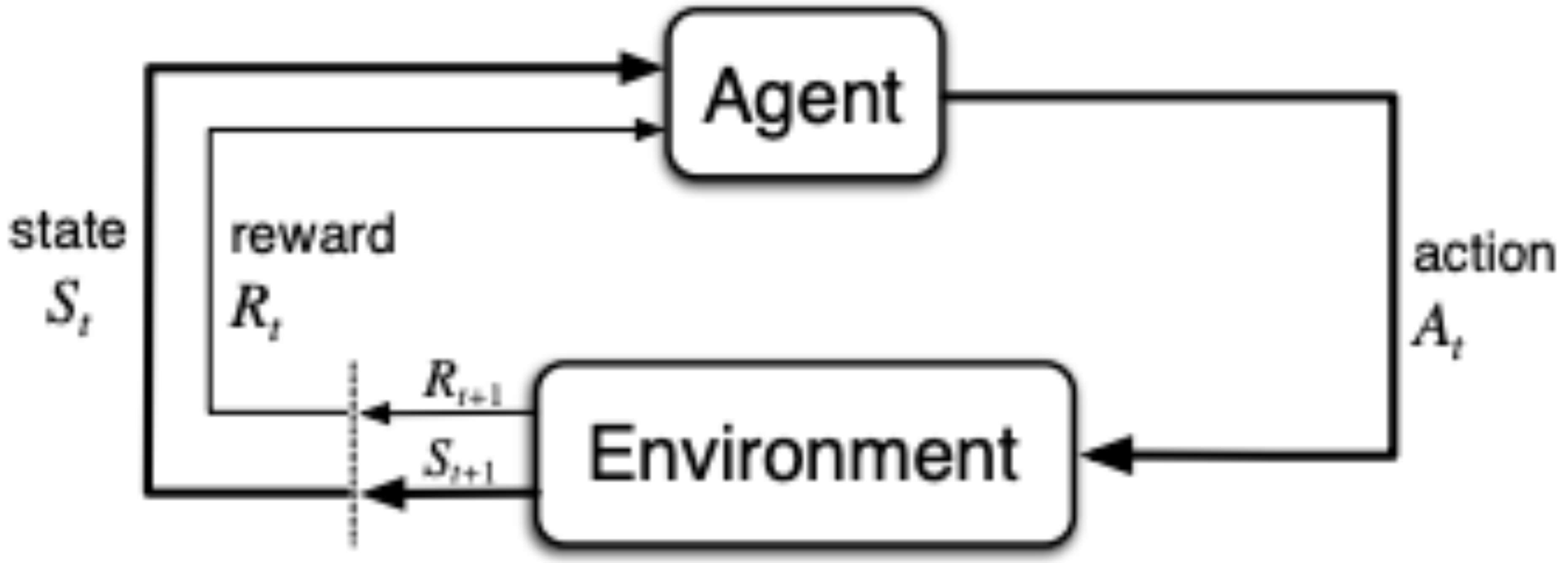


$$\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots$$

$$S_{t+1}, R_{t+1} \sim p(\cdot \mid S_t, A_t)$$

$$A_{t+1} \leftarrow \pi(S_{t+1})$$

Reinforcement Learning



Agent's objective is to find policy, π, so as to maximize the expected cumulative discounted reward from each state:

$$v_\pi(s) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t R_t \,|\, S_0 = s, A_t \leftarrow \pi(S_t), S_{t+1}, R_{t+1} \sim p(\cdot \,|\, S_t, A_t)]$$

$$= \mathbf{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \,|\, S_0 = s, A_t \leftarrow \pi(S_t), S_{t+1}, R_{t+1} \sim p(\cdot \,|\, S_t, A_t)]$$

For brevity, $\mathbf{E}_\pi$ will be used for $\mathbf{E}[\ldots \,|\, A_t \leftarrow \pi(S_t), S_{t+1}, R_{t+1} \sim p(\cdot \,|\, S_t, A_t)]$

Policies

- The agent's decision making rule.
- Formally, a function outputting the conditional probability of selecting an action in a particular state: $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$.
- A deterministic policy is a function mapping states to actions: $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

Returns and Episodes

- Episodes are subsequences of interaction that begin in some initial state and end in a special terminal state.

- **The initial state of one episode is independent of interaction in the preceding episode.**

- The return from step t is: $G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$

- Recursive definition: $G_t = R_{t+1} + \gamma G_{t+1}$.

Value functions

- Many RL algorithms use **value functions** to aid in long-term credit assignment.
- Two types of value function: state-value and action-value functions.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \,|\, S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s]$$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \,|\, S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a]$$

Bellman Equation

- Bellman equation expresses state-value, $v_\pi(s)$, in terms of expected reward and state-values at next time-step.

$$v_\pi(s) = \mathbf{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s]$$

- From the definition of expectation:

$$v_\pi(s) = \sum_a \pi(a \,|\, s) \sum_{s'} \sum_r p(s', r \,|\, s, a)[r + \gamma v_\pi(s')]$$

Optimality

- Agent's objective: find policy that maximizes $v_\pi(s)$ for all s.
- The optimal policy — policy that has maximal value in all states. $\pi^\star \geq \pi$ if $v_{\pi^\star}(s) \geq v_\pi(s)$ for all states and possible policies.
- Possibly multiple but always at least one deterministic optimal policy in a finite MDP.
- $\pi^\star(s) = \arg\max_a q_\star(s, a)$ $\qquad$ $q_\star(s, a) = \mathbf{E}[R_{t+1} + \gamma v_\star(S_{t+1}) \mid S_t = s, A_t = a]$

Value of taking action a and then acting optimally for all future time-steps.

Optimal Value Functions

- Like all policies, the optimal policy has value functions:
  - $v_{\pi^\star}(s) = \mathbb{E}[R_{t+1} + \gamma v_{\pi^\star}(S_{t+1}) \,|\, S_t = s]$
  - $q_{\pi^\star}(s, a) = \mathbb{E}[R_{t+1} + \gamma v_{\pi^\star}(S_{t+1}) \,|\, S_t = s, A_t = a]$
- The optimal policy is greedy with respect to the action-values, i.e., $\pi^\star(s) = \arg\max_a q_{\pi^\star}(s, a)$

Bellman Optimality Equation

$$v_\star(s) = \mathbf{E}_{\pi^\star}[q_\star(s, A)]$$

Exercise from last time: state-value is expected action-value.

$$= \sum_a \pi^\star(a \mid s) q_\star(s, a)$$

Definition of expectation.

$$= \max_a q_\star(s, a)$$

Optimal policy is greedy w.r.t $q_\star$

$$= \max_a \mathbf{E}_{\pi^\star}[G_t \mid S_t = s, A_t = a]$$

Definition of action-value .

$$= \max_a \mathbf{E}_{\pi^\star}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]$$

Recursive definition of return.

$$= \max_a \mathbf{E}_{\pi^\star}[R_{t+1} + \gamma v_\star(S_{t+1}) \mid S_t = s, A_t = a]$$

Definition of state-value.

$$v_\star(s) = \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_\star(s')]$$

Definition of expectation.

Dynamic Programming in RL

- Compute value functions and then use to find policies.
- Dynamic programming methods turn Bellman equations into value function updates.
- Bellman equation for policy value becomes the **policy evaluation** update:

$$v_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_k(s')]$$

$$\lim_{k \to \infty} v_k(s) = v_\pi(s)$$

- Bellman optimality equation becomes the value iteration update:

$$v_{k+1}(s) \leftarrow \max_a \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_k(s')]$$

$$\lim_{k \to \infty} v_k(s) = v_\star(s)$$

Policy Evaluation

- Given a policy, compute its state- or action-value function.

$$v_{k+1}(s) \leftarrow \sum_{a} \pi(a \mid s) \sum_{s'} \sum_{r} p(s', r \mid s, a)[r + \gamma v_k(s')]$$

$$q_{k+1}(s, a) \leftarrow \sum_{s'} \sum_{r} p(s', r \mid s, a)[r + \gamma \sum_{a'} q_k(s', a')]$$

- When to stop making updates?
- Do these updates converge?
  - Yes, update is a **contraction mapping** with fixed points v_π and q_π respectively.
  - Convergence proof for value-iteration.

Policy Evaluation Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Policy Improvement (Control)

- We have $v_\pi(s)$ for the current policy π. How can we improve π?
- Alternate:
  - Run policy evaluation updates to find v_π.
  - Set $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]$
  - Why does this update to π lead to an improved policy?

Policy Improvement Theorem

- Suppose for π that there exists s, a such that $q_\pi(s, a) \geq v_\pi(s)$.
- Let $\pi'(s) = a$ and $\pi'(\tilde{s}) = \pi(\tilde{s})$ for all other states.
- What is true about π'? Why?
  - As good as or better than π, i.e., $v_{\pi'}(s) \geq v_\pi(s), \forall s$
- If π is sub-optimal, does there exist s, a such that $q_\pi(s, a) \geq v_\pi(s)$?
  - Yes, this follows from Bellman Optimality. Must be at least one state where π is not greedy w.r.t. its action-value function.
  - Optimal value function: $v_\star(s) = \max_a q_\star(s, a) \forall s$

Policy Iteration

- First, evaluate π to obtain v_π.
- Then, update π to π' such that $\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + v_\pi(s')]$
- Policy improvement theorem guarantees that $v_{\pi'}(s) \geq v_\pi(s) \forall s$.
- Can converge quickly in practice (in terms of policy updates).

Policy Iteration Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Value Iteration

- What's wrong with policy iteration?
  - Policy evaluation must converge between policy updates.
  - We don't need the exact action-values — just which action has maximal action-value.
- Value iteration combines policy evaluation and iteration in one step:

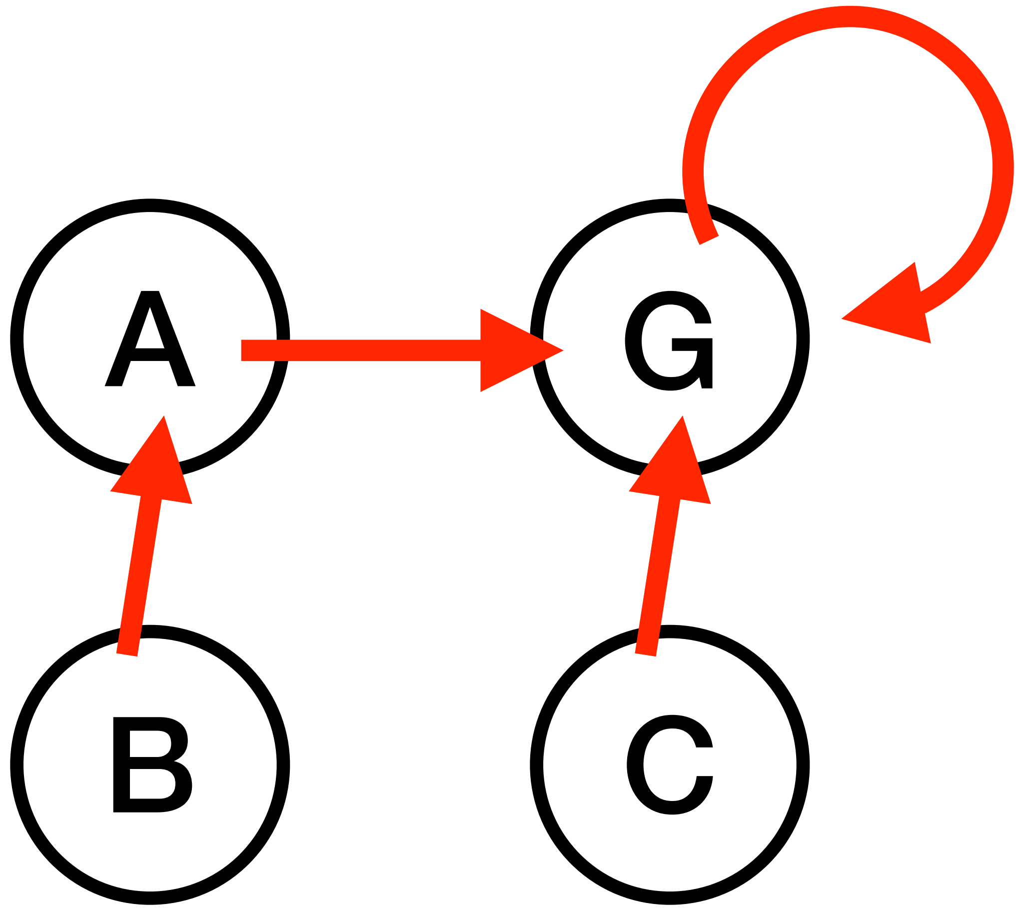
$$v_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$$

Value Iteration Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Quiz

Consider the following MDP which has deterministic transitions and $\gamma = 0.8$. The policy's action is shown with a red arrow. What is $v_\pi(B)$ in this MDP?



Two approaches:

1. Compute reward total for entire (infinite) sequence).
2. Compute $v_\pi(G)$ then $v_\pi(A)$ and then $v_\pi(B)$.

r(B) = 20; r(A) = 10; r(C) = 20; r(G) = 100

Q-learning

- Value iteration is not a learning method.
  - Requires knowledge of transitions and rewards to compute updates.
- Ideally, compute updates without this knowledge.
- Consider the agent is in state s and takes action a and then receives reward r and transitions to state s'; (s, a, s', r) is called a transition.
- Q-learning: initialize $q(s, a) = 0$ for all states and actions and then for each transition seen update:

$$q(s, a) \leftarrow (1 - \alpha)q(s, a) + \alpha(r + \gamma \max_{a'} q(s', a'))$$

Why is Q-learning reasonable?

- Consider a modified version of value iteration:

$$q_{k+1}(s,a) \leftarrow \sum_{s',r} p(s',r \,|\, s,a)[r + \gamma \max_{a'} q_k(s',a')]$$

- Without p and r we cannot compute right hand side **but** can approximate it after experiencing a reward and resulting next state.

$$q_{k+1}(s,a) \approx r + \gamma \max_{a'} q_k(s',a')$$

- With only a single reward and next state the update is noisy.
    - Moves q towards $q_\star$ in expectation but any single update has error.
    - Use a step-size parameter, α, to control:

$$q_{k+1}(s,a) \leftarrow (1-\alpha)q_k(s,a) + \alpha(r + \gamma \max_{a'} q_k(s',a'))$$

Q-learning Pseudocode

- Parameters: step-size α
- Initialize $q(s, a)$ arbitrarily for all states and actions except terminal states have $q(\texttt{terminal}, a) = 0$ for all a.
- Loop for each episode:
  - Initialize s
  - Loop for each step of episode until s is terminal:
    - Choose a from s using an exploration policy (more on this later).
    - Take action a and observe r and s'
    - $q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma \max_{a'} q(s', a') - q(s, a))$ Equivalent to update on previous slide
    - $s \leftarrow s'$

Summary

- Estimating value functions allow us to compute optimal policies.
- Policy Evaluation: find value function for a fixed policy.
- Policy Iteration: compute optimal policy by iterating 1) policy evaluation and 2) greedy policy improvement.
- Value Iteration: directly compute optimal value function.
- Q-learning: a learning method based based off of value iteration.

Thanks Everyone!

Slides adapted from Advanced Topics in RL and based on Chapter 4 of Reinforcement Learning: An Introduction.