Sequential Decision-Making Under Uncertainty:
Reinforcement Learning

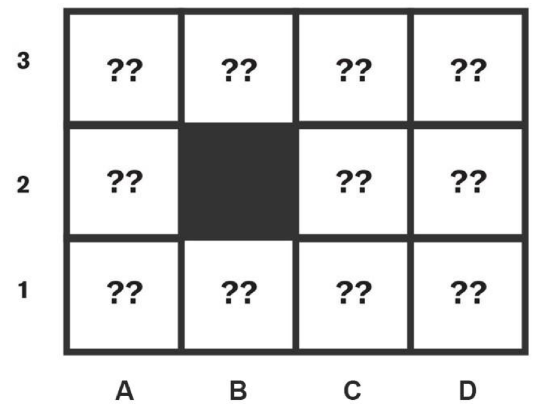
Siddharth Srivastava, Ph.D.
Assistant Professor
Arizona State University

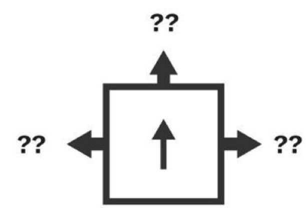
Ira A. Fulton Schools of Engineering
Arizona State University

What if the Agent Encounters a New Environment?

- S **set of states**
  - E.g., At(1,1)
- A **set of actions**
- T **transition model**
  - $P(s'|s,a) = T(s,a,s')$
- $R: S \rightarrow \mathbb{R}$

Unknown





Map not known to the agent
(or partially known)

- **Adapting to new situations is an essential component of intelligence**
- **How would we want an AI agent deal with it?**

Progression Towards RL

We considered two ideas

- Model learning,
- Monte Carlo policy evaluation (passive RL)
  - Recall: learns value function, but doesn't utilize reachability information

Next few slides:

- Further improvements, RL

Idea 3: Bootstrapped Monte Carlo

- Collect episode samples for the fixed policy as before (Monte Carlo evaluation)

- Now, bootstrap using the policy evaluation formula

$$V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s, \pi(s))[R(s) + \gamma V_k^{\pi}(s')]$$

- Since we don't know T, but we get R, we can **bootstrap** using current estimate of V:

Samples from s

$$\begin{aligned} V_{k+1}(s)[1] &= R(s) + \gamma \tilde{V}_k^{\pi}(s_1) \\ V_{k+1}(s)[2] &= R(s) + \gamma \tilde{V}_k^{\pi}(s_2) \\ V_{k+1}(s)[3] &= R(s) + \gamma \tilde{V}_k^{\pi}(s_3) \\ V_{k+1}(s)[4] &= R(s) + \gamma \tilde{V}_k^{\pi}(s_4) \end{aligned}$$

- $\tilde{V}^{\pi}{}_{k+1}(s) = \frac{1}{n}\sum_i V_{k+1}(s)[i]$

Analysis of Bootstrapped Monte Carlo

- Collect episode samples as before
- Since we don't know T, but we get R, we can **bootstrap** using current estimate of V
  - $\tilde{V}^{\pi}{}_{k+1}(s) = \frac{1}{n} \sum_i V_{k+1}(s)[i]$
- **Advantage**: This utilizes connections between states (bootstraps)
- **Limitation**: The agent can't utilize available data until the estimate $\tilde{V}_{k+1}$ is computed (at end of episode)

Idea 4: Temporal-Difference (TD) Learning

- **This is the central idea of Reinforcement Learning!**
- **Combines MC evaluations, bootstrapping, and online action**
  - Online Monte Carlo policy evaluation and learning
- **What does "online" stand for?**
  - Update your value function at each step rather than at end of episode
  - Policy is still fixed
- **Not immediately clear how…**

Idea 4: Temporal-Difference (TD) Learning

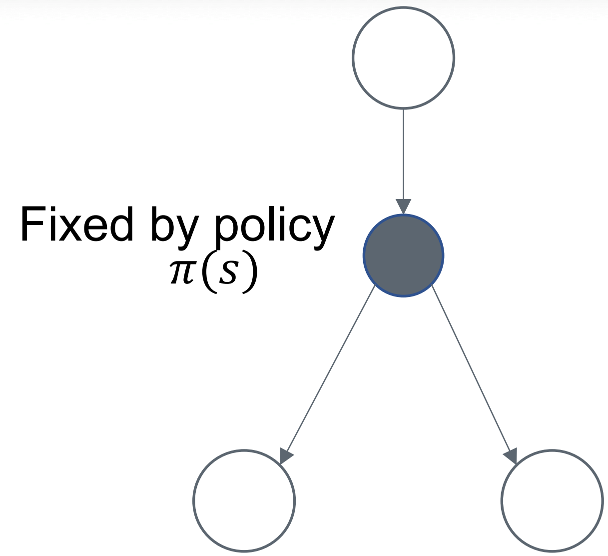
- **This is the central idea of Reinforcement Learning!**
- **Combines bootstrapping, MC evaluations, and online action**
  - Online Monte Carlo policy evaluation and learning
- **What does "online" stand for?**
  - Update your value function at each step rather than at end of episode
  - Policy is still fixed
- **Not immediately clear how...**
  - Need to update value function: $V^\pi(s) \leftarrow V^\pi(s) + ?$

TD Learning

- Current Estimate: $\tilde{V}^{\pi}(s)$

- Update this with a new sample as follows:

$$\tilde{V}^{\pi}(s) \leftarrow \tilde{V}^{\pi}(s) + \alpha\left(V(s)[i] - \tilde{V}^{\pi}(s)\right)$$
$$= (1-\alpha)\tilde{V}^{\pi}(s) + \alpha \cdot V(s)[i]$$



- **α is called the learning rate or step size**

TD Learning converges to the true policy values if α decays to zero, all states visited infinitely often

Analysis of TD Value and Q Learning

- **A similar rule can be derived for updating the Q function rather than the V function**

- **Advantages of TD learning**
  - Online update improves estimates as you go
  - Especially good for tasks with long episodes, or continuing tasks that don’t have natural ends
  - Bootstraps using information about state connectivity

- **Limitations: doesn’t improve the policy**
  - Online Monte Carlo Policy Evaluation (not learning policies yet)

Extracting Actions using TD Evaluation

- Suppose we learned V using TD learning
- How can we determine which action to use?
- We know about extracting the policy from the value function:

$$\pi(s) = argmax_a Q(s,a)$$

$$Q(s,a) = \sum_{s'} P(s'|s,a)[R(s) + \gamma V(s')]$$

- This gives us an opportunity to do a policy update – much like policy iteration
- But this doesn't quite help here (why?)

Learning the Q-Function using TD

Q-value iteration:

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s) + \gamma V_k(s')]$$

$$= \sum_{s'} P(s'|s,a)[R(s) + \gamma \max_{a'} Q_k(s',a')]$$

Next state and action

We can adapt this to use samples like we did for evaluating the V function for a given policy

– i^{th} sample:

$$Q(s,a)[i] = R(s)[i] + \gamma \tilde{Q}(s',a')$$

$s, a, R(s), s', a'$ as seen in the sample

TD updates for Q:

$$\tilde{Q}(s,a) \leftarrow (1-\alpha)\tilde{Q}(s,a) + \alpha Q(s,a)[i]$$

Learning Q-Function Using TD: SARSA

$$\tilde{Q}(s,a) \leftarrow (1-\alpha)\tilde{Q}(s,a) + \alpha Q(s,a)[i]$$

- This approach uses s, a, R, s', a' tuples from samples
  - It's aptly named "SARSA"
- TD learning for V and for Q does on-policy estimation
  - Estimate V and Q of the policy while executing it
- Q-learning: a powerful variation that learns a better policy

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

[Sutton & Barto, RL, 1st ed.]

Q-Learning

- **SARSA learned the Q value of the policy being followed:**

$$\tilde{Q}(s,a) \leftarrow (1-\alpha)\tilde{Q}(s,a) + \alpha \cdot [R(s) + \gamma \tilde{Q}(s',a')]$$

- **Q-learning changes things to go 1-step greedy over the current Q estimate**

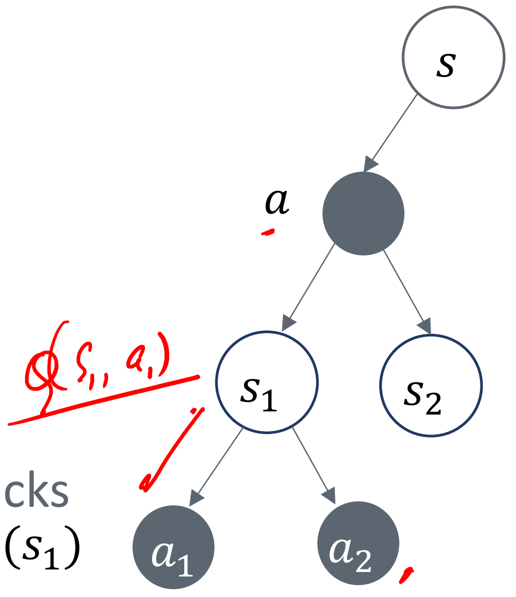
$$\tilde{Q}(s,a) \leftarrow (1-\alpha)\tilde{Q}(s,a) + \alpha \cdot [R(s) + \gamma \max_{a'} \tilde{Q}(s',a')]$$

- **This is also called off-policy control**

- **Why is this useful?**

SARSA: $\tilde{Q}(s,a) \leftarrow (1-\alpha)\tilde{Q}(s,a) + \alpha \cdot [R(s) + \gamma \tilde{Q}(s',a')]$

Q-learning: $\tilde{Q}(s,a) \leftarrow (1-\alpha)\tilde{Q}(s,a) + \alpha \cdot [R(s) + \gamma \, \underline{max_{a'}} \; \tilde{Q}(s',a')]$



SARSA always picks $\pi(s_1)$

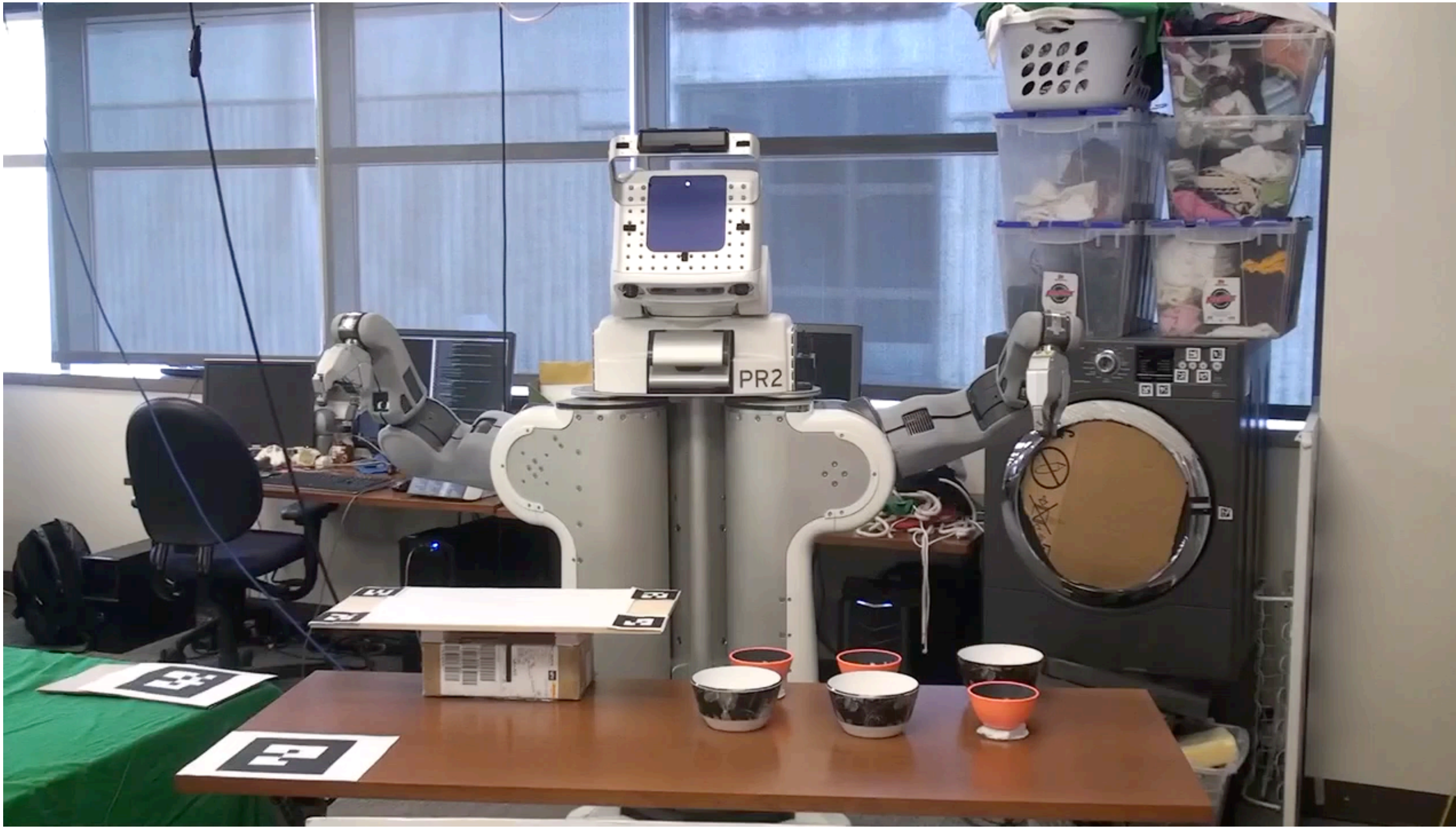
Q-learning picks the better one among a_1, a_2
This makes Q-learning "learn" off policy
As a result, Q-learning can be used to learn optimal policies while following an arbitrary policy!*

Analysis of Q-Learning

- **Q-learning converges to the optimal Q function**
  - Even if the policy being followed is not optimal
  - This is great: your self driving car could learn from you and do better!*
- ***Assumptions:**
  - Must keep visiting all pairs (s,a) infinitely often
  - Decay α appropriately over time
- **Q-learning is one of the most popular RL approaches**
  - Online Monte Carlo policy evaluation and learning

Summary

- **SDM/Planning: mathematical unifying framework in AI!**
  - Get the agent to do what you want
  - Without having to program it
- **In the process, need to utilize everything you learned in the course:**
  - Learning (models + policies)
  - Perception (special case of reasoning)
  - Search, dynamic programming, Monte Carlo methods


PR2

Directions for Branching Off

Sequential decision making is a rich, active area of research. Here are a few pointers in case you wish to delve deeper!

- Relational representations for MDPs and RL
- Hierarchical planning and learning
- Deep neural networks as function approximators for V and Q, as heuristic learners
- Planning in situations with partial observability
  - Partially observable MDPs

Becoming an AI Expert

A few general principles for designing AI applications

- Use efficient representations
- Exploit structure of the problem (facilitated by good representation)
- If prior knowledge is available, use it!

General principles for evaluating methods

- Keep an open mind
- But, be critical! Distinguish fact from opinion (even yours)
- Look for assumptions made and guarantees provided
  - When is the approach going to work? When will it fail?

Design responsibly!