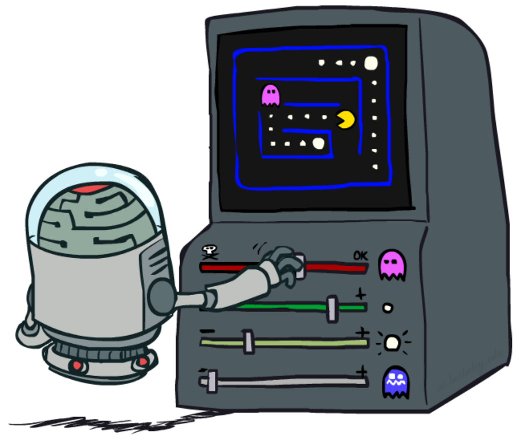
CSE 473: Artificial Intelligence

Reinforcement Learning



Dan Weld/ University of Washington

[Many slides taken from Dan Klein and Pieter Abbeel / CS188 Intro to AI at UC Berkeley – materials available at http://ai.berkeley.edu.]

Three Key Ideas for RL

- Model-based *vs* model-free learning
  - What function is being learned?
- Approximating the Value Function
  - Smaller → easier to learn & better generalization
- Exploration-exploitation tradeoff

Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment & learn model, T=P(**s**'|**s**,**a**) and R(**s**,**a**), (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - often works well when state-space is manageable
- Model-free approach:
  - don't learn a model; learn value function or policy directly
  - weaker theoretical results
  - often works better when state space is large

23

Two main reinforcement learning approaches

- Model-based approaches:

  Learn T + R

  $|S|^2|A| + |S||A|$ parameters (40,400)

- Model-free approach:

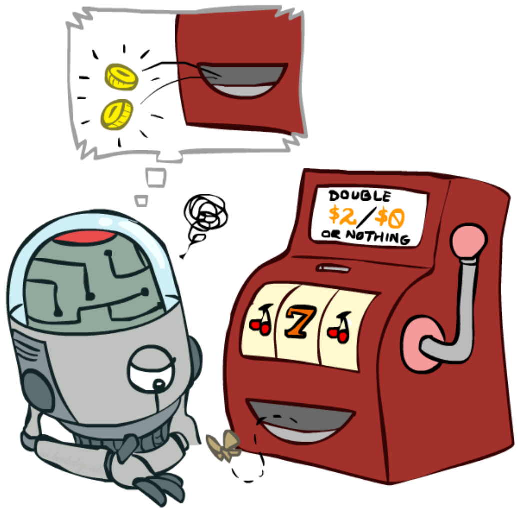
  Learn Q

  $|S||A|$ parameters (400)

Suppose 100 states, 4 actions

24

Model-Free Learning



Nothing is Free in Life!

- What exactly is Free???
  - No model of T
  - No model of R
  - (Instead, just model Q)

26

Reminder: Q-Value Iteration

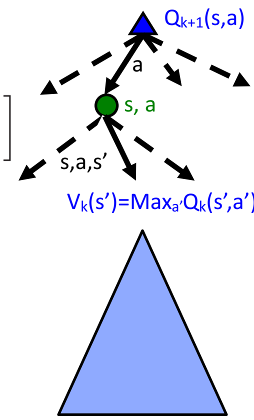
- **Forall s, a**
  - **Initialize Q_0(s, a) = 0** *no time steps left means an expected reward of zero*
- **K = 0**
- **Repeat** *do Bellman backups*

  For every (s,a) pair:

  $$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

  K += 1
- **Until convergence** *I.e., Q values*

This is easy….



Puzzle: Q-Learning

- **Forall s, a**
  - **Initialize Q_0(s, a) = 0** *no time steps left means an expected reward of zero*
- **K = 0**
- **Repeat** *do Bellman backups*

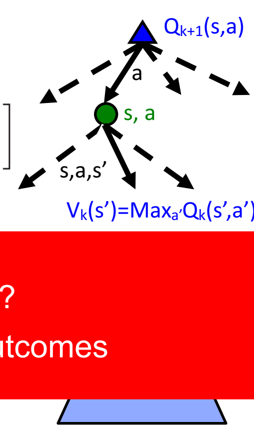
  For every (s,a) pair:

  $$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

  K += 1
- **Until convergence**

Q: How can we compute without R, T ?!?

A: Compute averages using sampled outcomes



Simple Example: Expected Age

Goal: Compute expected age of CSE students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \quad = 0.35 \times 20 + \ldots$$

Note: never know **P(age=22)**

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because samples appear with the right frequencies.

Anytime Model-Free Expected Age

Goal: Compute expected age of CSE students

Let A=0
Loop for i = 1 to ∞
  a_i ← ask "what is your age?"
  A ← (1-α)*A + α*a_i

Without P(A), instead collect samples [a_1, a_2, … a_N]

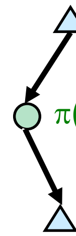
Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Let A=0
Loop for i = 1 to ∞
  a_i ← ask "what is your age?"
  A ← (i-1)/i * A + (1/i) * a_i

Sampling Q-Values

- Big idea: learn from every experience!
  - Follow exploration policy a ← π(s)
  - Update Q(s,a) each time we experience a transition (s, a, s’, r)
  - Likely outcomes s’ will contribute updates more often



- Update towards running average:

Get a sample of Q(s,a): *sample* = $R(s,a,s') + \gamma \,\mathrm{Max}_{a'}\, Q(s', a')$

Update to Q(s,a): $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)$*sample*

Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**
- **Repeat Forever**

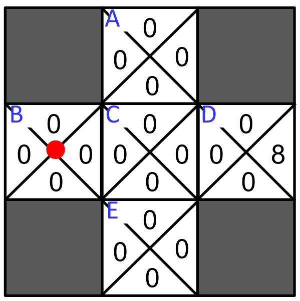
  Where are you? s.
  Choose some action a
  Execute it in real world: *(s, a, r, s’)*
  Do update:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Example

Assume: γ = 1, α = 1/2

Observed Transition: B, east, C, -2



In state B. What should you do?

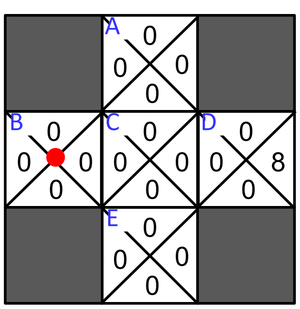
Suppose (for now) we follow a random exploration policy

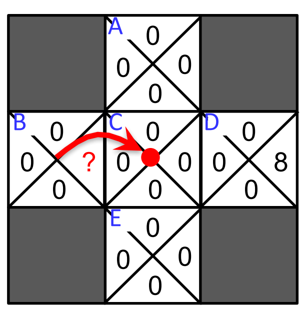
 → "Go east"

Example

Assume: γ = 1, α = 1/2

Observed Transition: B, east, C, -2





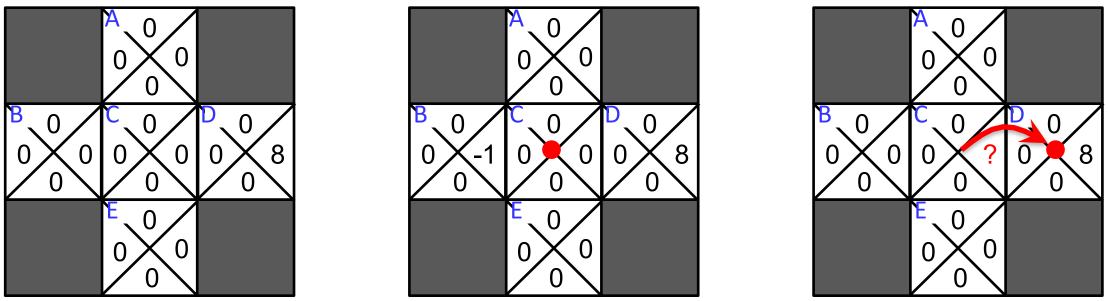
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

-1 ½ 0 ½ -2 0

Example

Assume: γ = 1, α = 1/2

Observed Transition: B, east, C, -2 C, east, D, -2



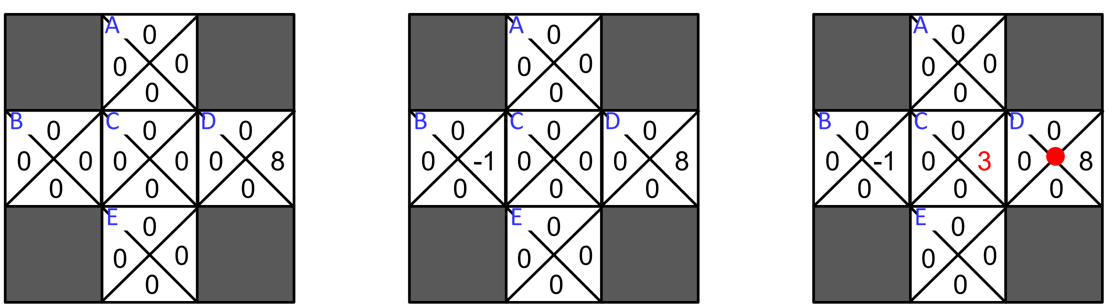
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

3 ½ 0 ½ -2 8

Example

Assume: γ = 1, α = 1/2

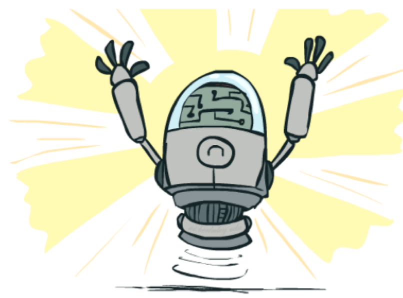
Observed Transition: B, east, C, -2 C, east, D, -2



$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Q-Learning Properties

- Q-learning converges to optimal Q function (and hence ***learns*** optimal policy)
  - even if you're acting suboptimally!
  - This is called off-policy learning
- Caveats:
  - *You have to explore enough*
  - *You have to eventually shrink the learning rate, α*
  - *… but not decrease it too quickly*
- And… if you want to ***act*** optimally
  - You have to switch from explore to exploit



[Demo: Q-learning – auto – cliff grid (L11D1)]

Video of Demo Q-Learning Auto Cliff Grid

Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**
- **Repeat Forever**

  Where are you? s.

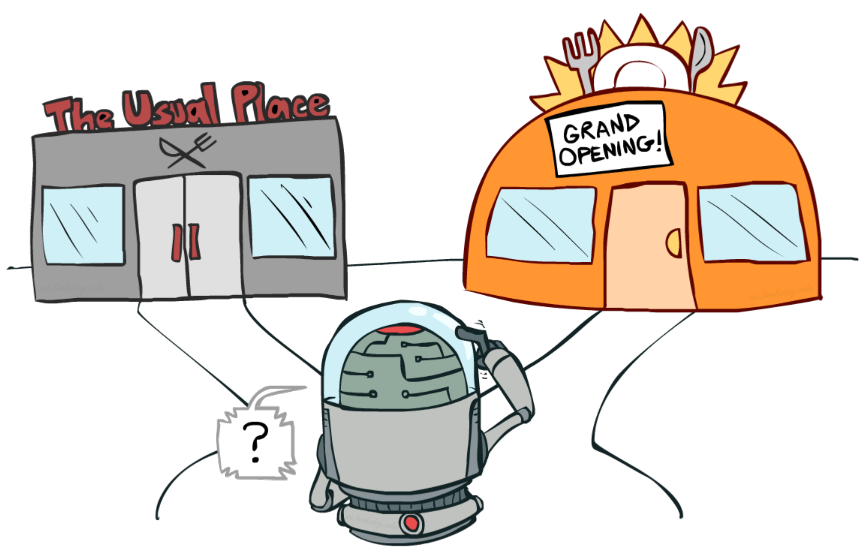
  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

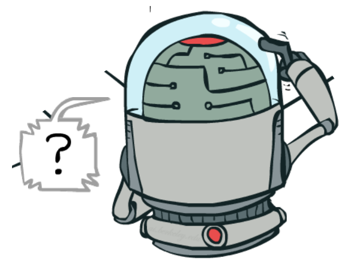
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Exploration vs. Exploitation



Questions

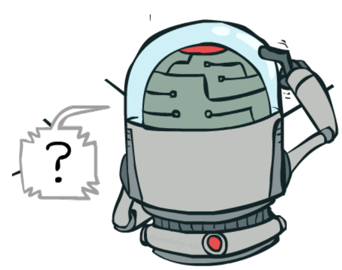
- How to explore?



- When to exploit?
- How to even think about this tradeoff?

Questions

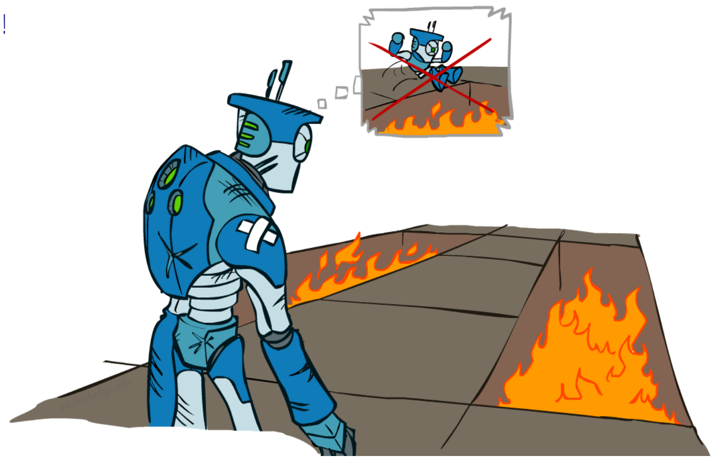
- How to explore?
  - Random Exploration
  - Uniform exploration
  - Epsilon Greedy
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability $1-\varepsilon$, act on ***current policy***



- When to exploit?
- How to even think about this tradeoff?

Regret

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total ***mistake cost***: the difference between your (expected) rewards, including youthful sub-optimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires ***optimally learning to be optimal***

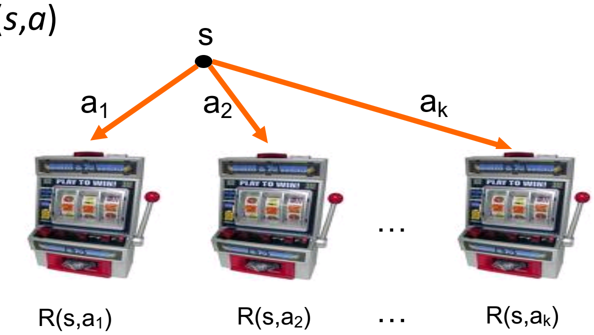


Two KINDS of Regret

- **Cumulative Regret:**
  - achieve near optimal cumulative lifetime reward (in expectation)
- **Simple Regret:**
  - quickly identify policy with high reward (in expectation)

RL on Single State MDP

- Suppose MDP has a single state and k actions
  - Can sample rewards of actions using call to simulator
  - Sampling action *a* is like pulling slot machine arm with random payoff function *R*(*s*,*a*)



50

Multi-Armed Bandit Problem

Multi-Armed Bandits

- Bandit algorithms are not just useful as components for RL & Monte-Carlo planning
- Pure bandit problems arise in many applications
- Applicable whenever:
  - set of independent options with unknown utilities
  - cost for sampling options or a limit on total samples
  - Want to find the best option or maximize utility of samples

Multi-Armed Bandits: Example 1

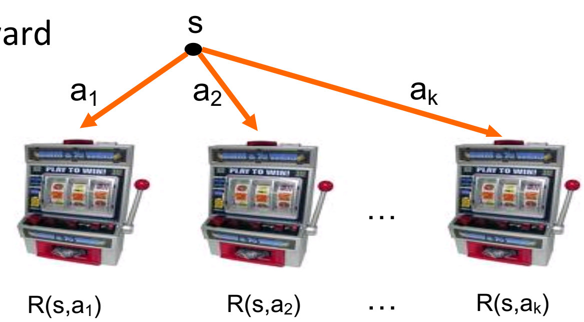
- **Clinical Trials**
  - Arms = possible treatments
  - Arm Pulls = application of treatment to inidividual
  - Rewards = outcome of treatment
  - Objective = maximize cumulative reward = maximize benefit to trial population (or find best treatment quickly)

Multi-Armed Bandits: Example 2

- **Online Advertising**
  - Arms = different ads/ad-types for a web page
  - Arm Pulls = displaying an ad upon a page access
  - Rewards = click through
  - Objective = maximize cumulative reward = maximum clicks (or find best add quickly)

Multi-Armed Bandit: Possible Objectives

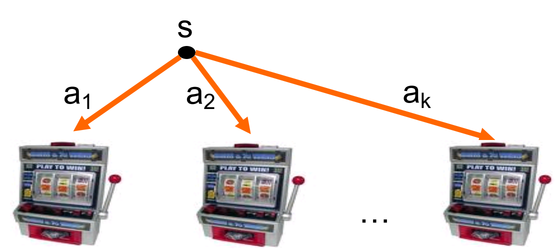
- **PAC Objective:**
  - find a near optimal arm w/ high probability
- **Cumulative Regret:**
  - achieve near optimal cumulative reward over lifetime of pulling (in expectation)
- **Simple Regret:**
  - quickly identify arm with high reward
  - (in expectation)



54

Cumulative Regret Objective

- **Problem:** find arm-pulling strategy such that the expected total reward at time *n* is ***close*** to the best possible (one pull per time step)
  - Optimal (in expectation) is to pull optimal arm *n* times
  - UniformBandit is poor choice --- waste time on bad arms
  - Must balance **exploring** machines to find good payoffs and **exploiting** current knowledge



55

How to Explore?

Several schemes for forcing exploration

- Simplest: random actions (ε-greedy)
  - Every time step, flip a coin
  - With (small) probability ε, act randomly
  - With (large) probability 1-ε, act on ***current policy***

- Problems with random actions?
  - You do eventually explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

- Theory of Multi-Armed Bandits

Exploration Functions

- When to explore?
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

- Note: this propagates the "bonus" back to states that lead to unknown states as well!

[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

Cumulative Regret Objective

- Theoretical results are often about "expected cumulative regret" of an arm pulling strategy.

- **Protocol:** At time step n the algorithm picks an arm a_n based on what it has seen so far and receives reward r_n (a_n *and* r_n *are random variables)*.

Strategy if one knew which arm was best

- **Expected Cumulative Regret ($E[Reg_n]$):** difference between optimal expected cummulative reward and expected cumulative reward of our strategy at time n

$$E[Reg_n] = n \cdot R^* - \sum_{i=1}^{n} E[r_n]$$

UCB Algorithm for Minimizing Cumulative Regret

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, *47*(2), 235-256.

- Q(a) : average reward for trying action a (in our single state *s*) so far
- n(a) : number of pulls of arm a so far
- Action choice by UCB after n pulls:

$$a_n = \arg\max_a Q(a) + \sqrt{\frac{2\ln n}{n(a)}}$$

- Assumes rewards in [0,1] – normalized from R_{max}.

59

UCB: Bounded Sub-Optimality

$$a_n = \arg\max_a Q(a) + \sqrt{\frac{2\ln n}{n(a)}}$$

Value Term:
favors actions that looked good historically

Exploration Term:
actions get an exploration bonus that grows with ln(n)

Expected number of pulls of sub-optimal arm **a** is bounded by: $\frac{8}{\Delta_a^2}\ln n$

where Δ_a is the sub-optimality of arm **a**

Doesn't waste much time on sub-optimal arms, unlike uniform!

60

UCB Performance Guarantee

[Auer, Cesa-Bianchi, & Fischer, 2002]

Theorem: The expected cumulative regret of UCB $\boldsymbol{E}[\boldsymbol{Reg_n}]$ after *n* arm pulls is bounded by O(log *n*)

- Is this good?

Yes. The average per-step regret is $O\left(\frac{\log(n)}{n}\right)$

Theorem: No algorithm can achieve a better expected regret (up to constant factors)

61