



南方科技大学

STA303: Artificial Intelligence

Reinforcement Learning

Fang Kong

<https://fangkongx.github.io/>

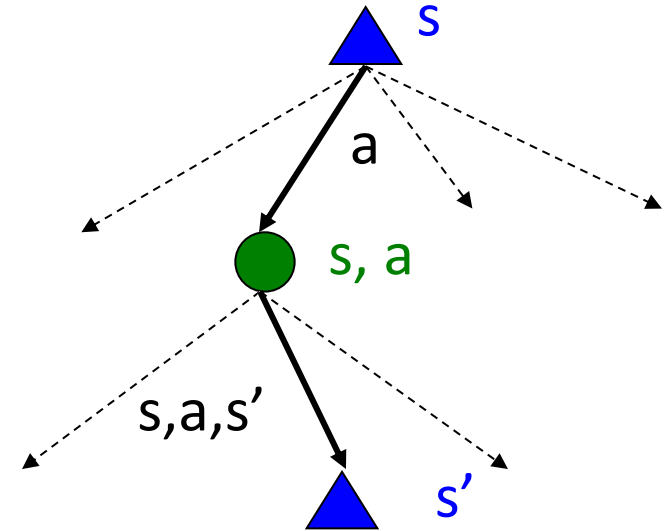
Recap: MDPs

- Markov decision processes:

- States S
- Actions A
- Transitions $P(s' | s, a)$ (or $T(s, a, s')$)
- Rewards $R(s, a, s')$ (and discount γ)
- Start state s_0

- Quantities:

- Policy = map of states to actions
- Utility = sum of discounted rewards
- Values = expected future utility from a state (max node)
- Q-Values = expected future utility from a q-state (chance node)



Recap: The Bellman Equations

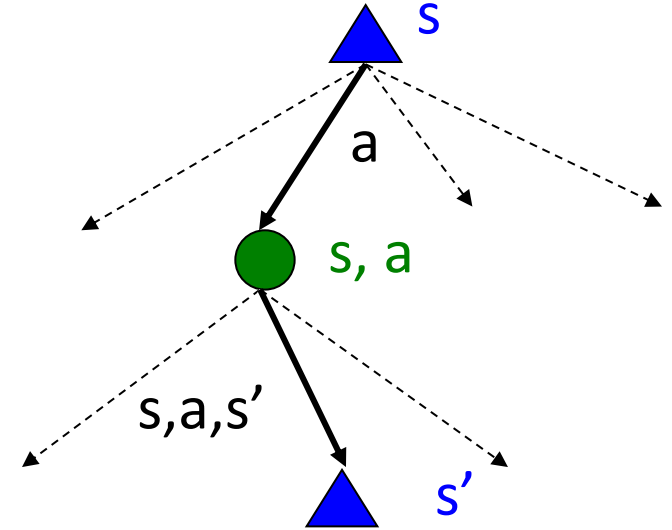
- Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

$$V^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

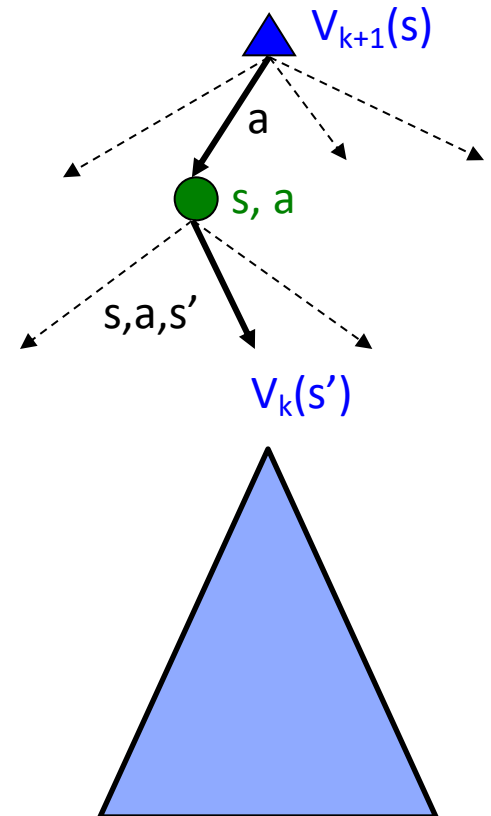


Recap: Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
 - Basic idea: approximations get refined towards optimal values
 - Policy may converge long before values do

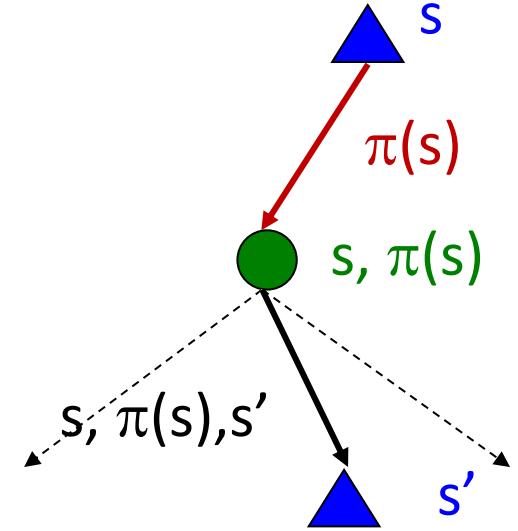


Recap: Policy Evaluation

- How do we calculate the V 's for a fixed policy π ?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



- Efficiency: $O(S^2)$ per iteration
- Idea 2: Without the maxes, the Bellman equations are just a linear system
 - Solve with Matlab (or your favorite linear system solver)

Recap: Policy Extraction

- Let's imagine we have the optimal values $V^*(s)$
- How should we act?
 - It's not obvious!
- We need to do a mini-expectimax (one step)



$$\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

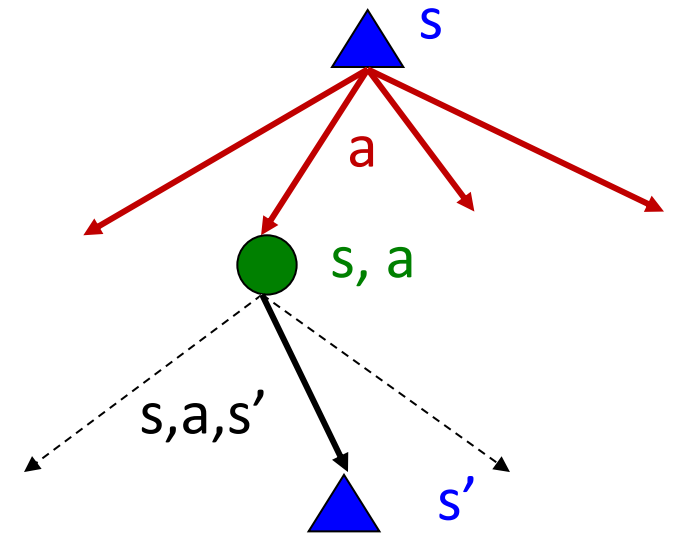
- This is called **policy extraction**, since it gets the policy implied by the values

Recap: Problems with Value Iteration

- Value iteration repeats the Bellman updates:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The “arg max” at each state rarely changes
- Problem 3: The policy often converges long before the values



Recap: Policy Iteration

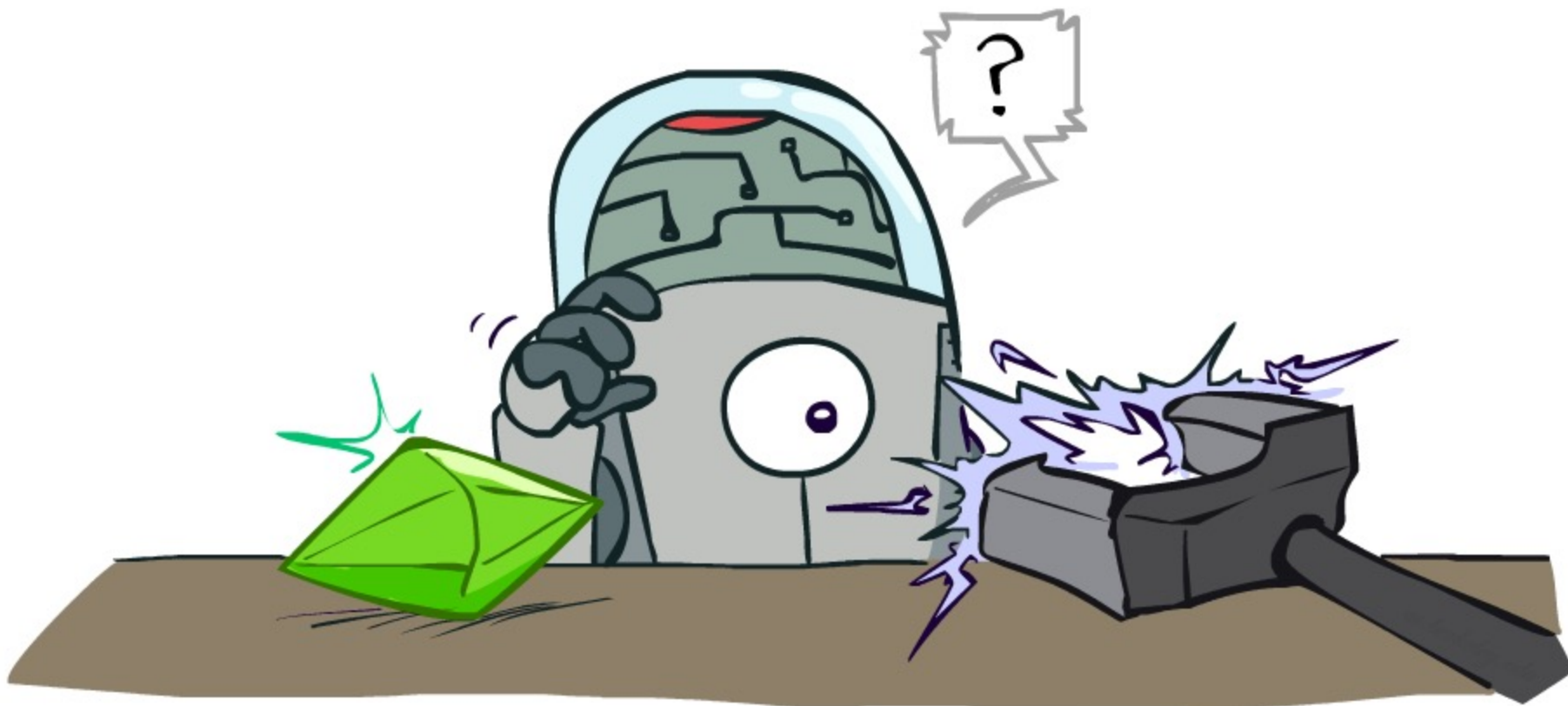
- Evaluation: For fixed current policy π , find values with policy evaluation:
 - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') [R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

- Improvement: For fixed values, get a better policy using policy extraction
 - One-step look-ahead:

$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')]$$

Reinforcement Learning



Reinforcement Learning

- Still assume a Markov decision process (MDP):
 - A set of states $s \in S$
 - A set of actions (per state) $A(s)$
 - A transition model $T(s,a,s')$
 - A reward function $R(s,a,s')$
- Still looking for a policy $\pi(s)$
- New twist: don't know T or R
 - I.e. we don't know which states are good or what the actions do
 - Must explore new states and actions to discover how the world works

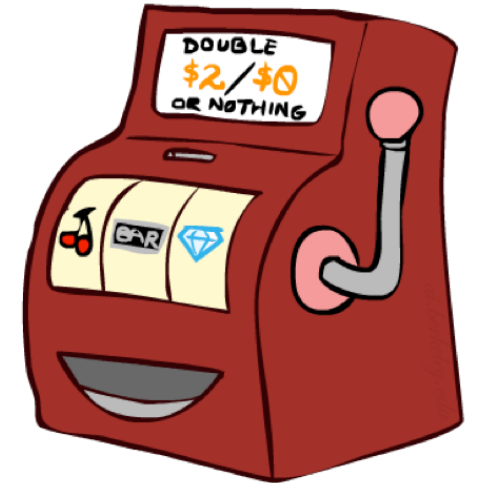
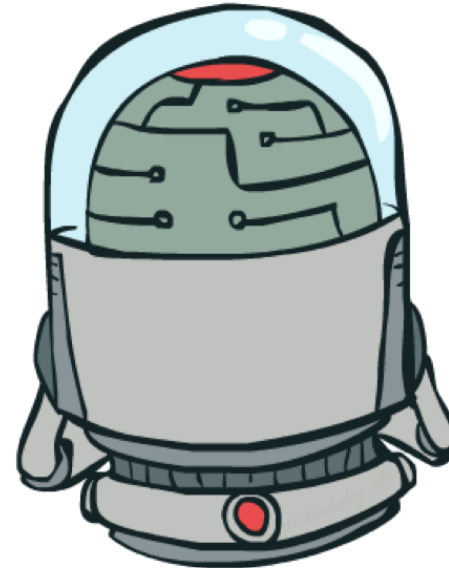


Reinforcement Learning

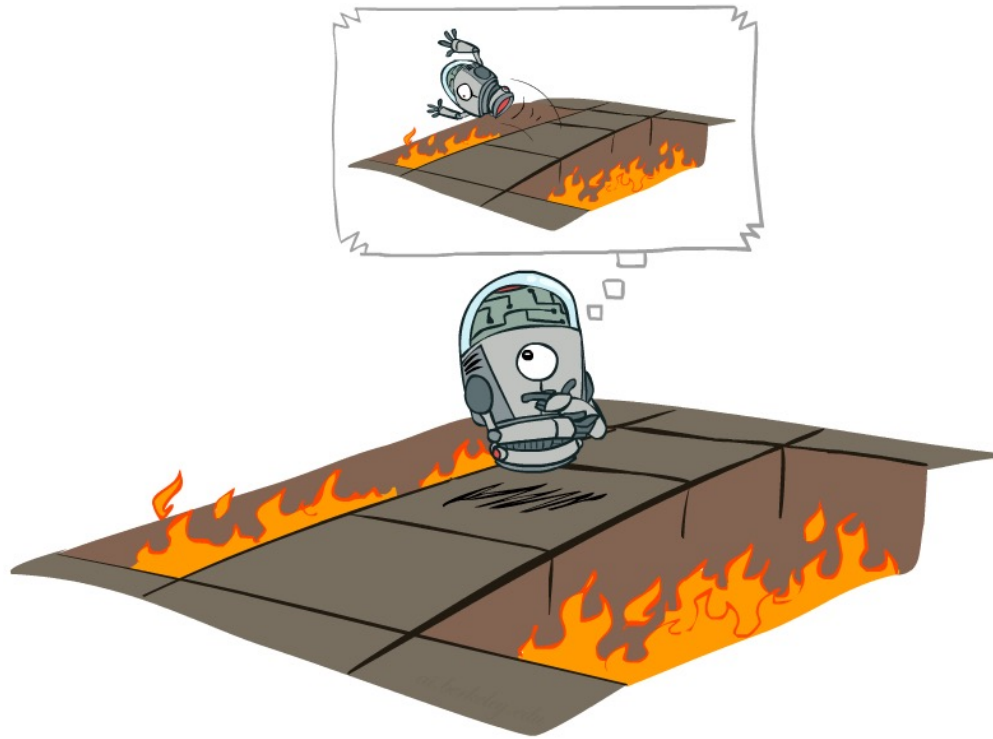
- What if the MDP is initially unknown? Lots of things change!
 - **Exploration**: you have to *try unknown actions* to get information
 - **Exploitation**: eventually, you have to use what you know
 - **Regret**: early on, you inevitably “make mistakes” and lose reward
 - **Sampling**: you may need to repeat many times to get good estimates
 - **Generalization**: what you learn in one state may apply to others too

Bandits

- Exactly one state
- Set of actions: A
- Stochastic reward function: $P(r|a)$



Offline (MDPs) vs. Online (RL)



Offline Solution

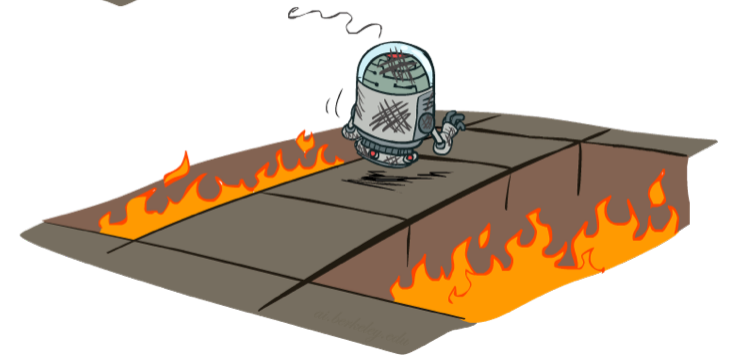
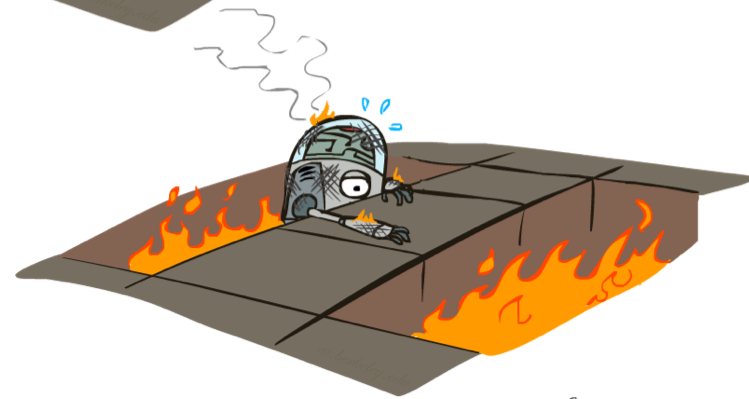
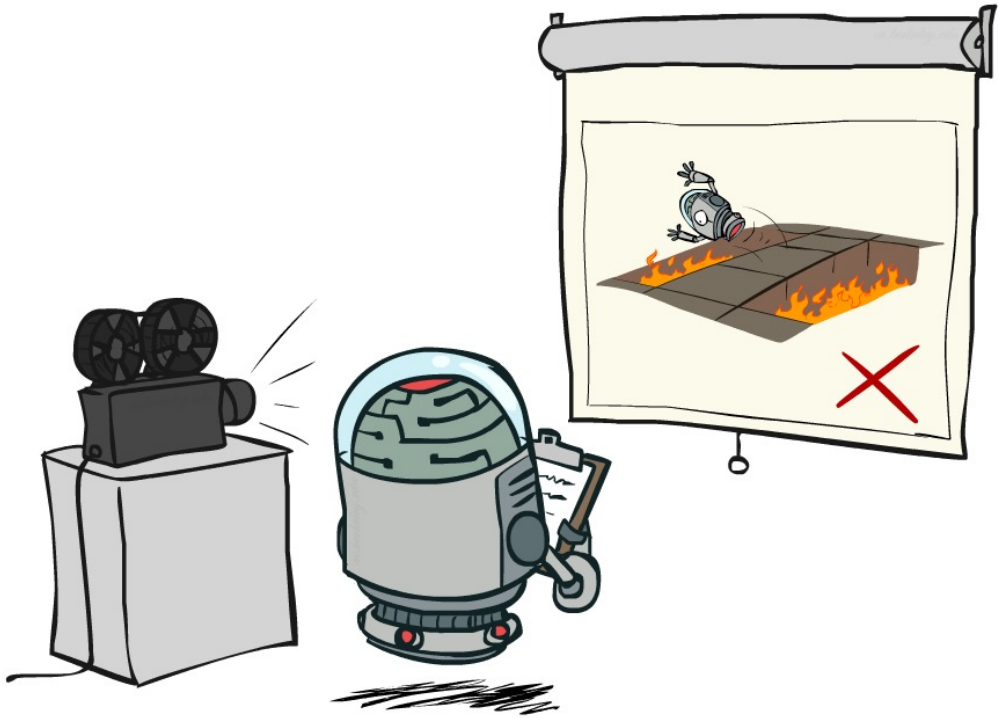


Online Learning

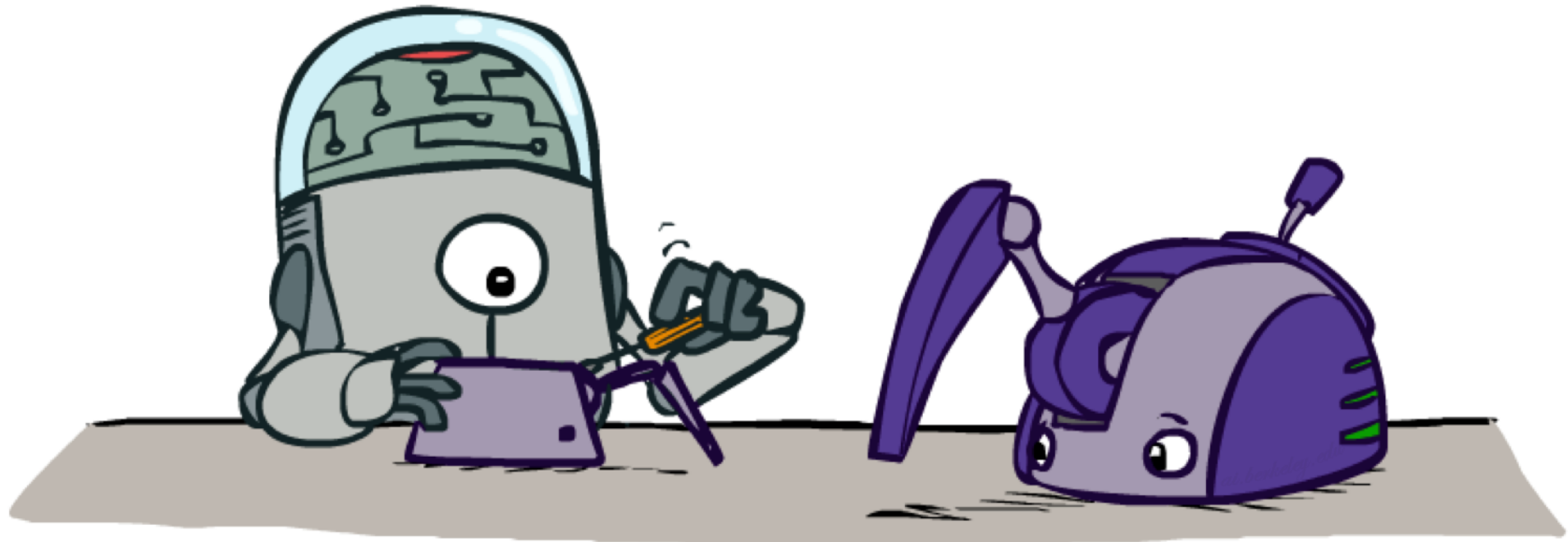
Approaches to reinforcement learning

1. Model-based: Learn the model, solve it, execute the solution
2. Learn values from experiences, use to make decisions
 - a. Direct evaluation
 - b. Temporal difference learning
 - c. Q-learning
3. Optimize the policy directly

Passive vs Active Reinforcement Learning



Model-Based RL



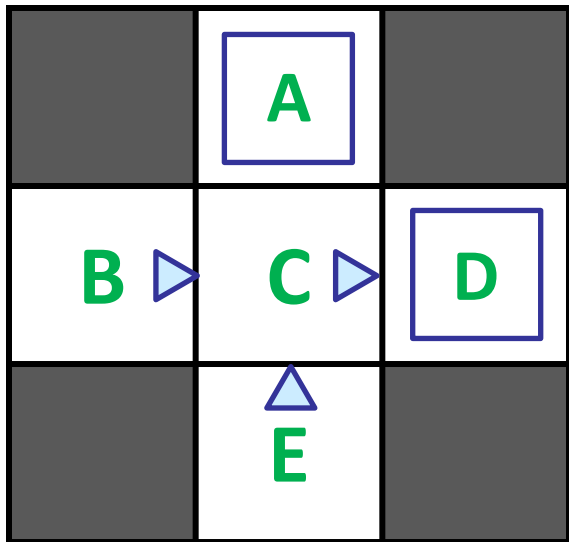
Model-Based Learning

- **Model-Based Idea:**
 - Learn an approximate model based on experiences
 - Solve for values as if the learned model were correct
- **Step 1: Learn empirical MDP model**
 - Count outcomes s' for each s, a
 - Directly estimate each entry in $T(s, a, s')$ from counts
 - Discover each $R(s, a, s')$ when we experience the transition
- **Step 2: Solve the learned MDP**
 - Use, e.g., value or policy iteration, as before



Example: Model-Based Learning

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$T(s, a, s')$

$T(B, \text{east}, C) = 1.00$
 $P(C, \text{east}, D) = 0.75$
 $P(C, \text{east}, A) = 0.25$
...

$R(s, a, s')$

$R(B, \text{east}, C) = -1$
 $R(C, \text{east}, D) = -1$
 $R(D, \text{exit}, x) = +10$
...

Pros and cons

- Pro:

- Makes efficient use of experiences (low *sample complexity*)

- Con:

- May not scale to large state spaces
 - Solving MDP is intractable for very large $|S|$
- RL feedback loop tends to magnify small model errors
- Much harder when the environment is partially observable

Basic idea of model-free methods

- To approximate expectations with respect to a distribution, you can either
 - Estimate the distribution from samples, compute an expectation
 - Or, bypass the distribution and estimate the expectation from samples directly

Example: Expected Age

Goal: Compute expected age of STA303 students

Known $P(A)$

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

Without $P(A)$, instead collect samples $[a_1, a_2, \dots, a_N]$

“Model Based”: estimate $P(A)$:

$$\hat{P}(A=a) = N_a/N$$

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

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

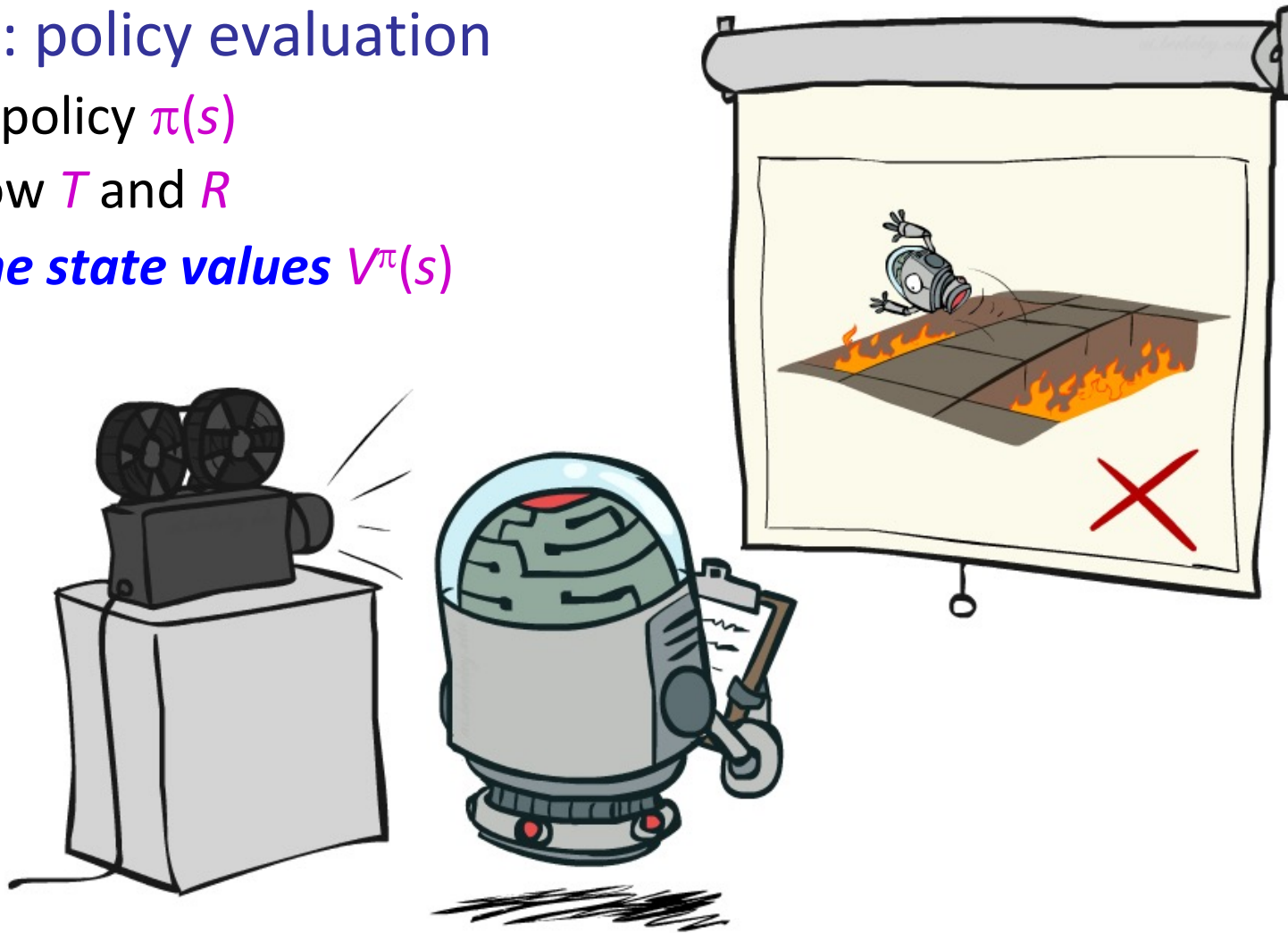
“Model Free”: estimate expectation

$$E[A] \approx 1/N \sum_i a_i$$

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

Passive Reinforcement Learning

- Simplified task: policy evaluation
 - Input: a fixed policy $\pi(s)$
 - You don't know T and R
 - **Goal: learn the state values $V^\pi(s)$**



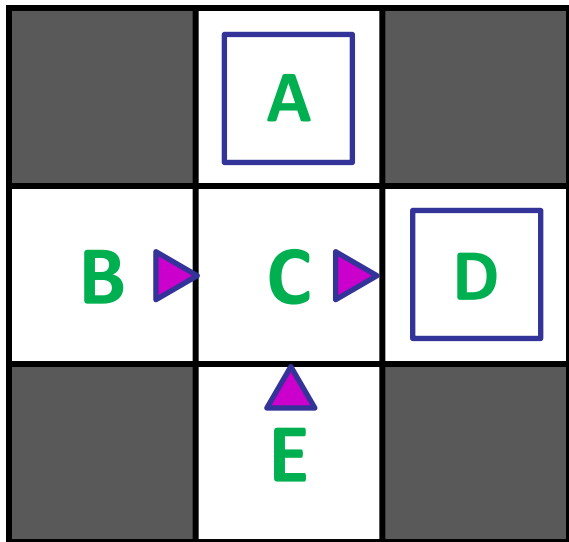
Direct evaluation

- Goal: Estimate $V^\pi(s)$, i.e., expected total discounted reward from s onwards
- Idea:
 - Use *returns*, the actual sums of discounted rewards from s
 - Average over multiple trials and visits to s
- This is called **direct evaluation** (or direct utility estimation)



Example: Direct Estimation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values

	-10	
	A	
+8	+4	+10
B	C	D
	-2	
	E	

Problems with Direct Estimation

- What's good about direct estimation?
 - It's easy to understand
 - It doesn't require any knowledge of T and R
 - It converges to the right answer in the limit
- What's bad about it?
 - Each state must be learned separately (fixable)
 - It **ignores information about state connections**
 - So, it takes a long time to learn

*E.g., B=at home, study hard
E=at library, study hard
C=know material, go to exam*

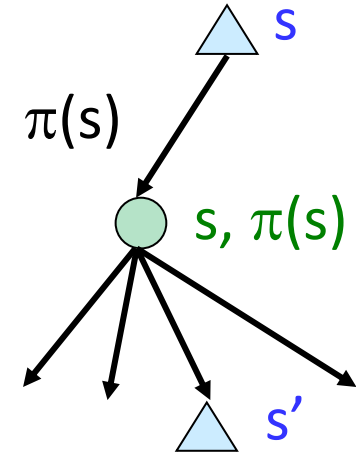
Output Values

	-10 A	
+8 B	+4 C	+10 D
	-2 E	

*If B and E both go to C
under this policy, how can
their values be different?*

Temporal Difference Learning

- Big idea: learn from every experience!
 - Update $V(s)$ each time we experience a transition (s, a, s', r)
 - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values
 - Policy still fixed, still doing evaluation!
 - Move values toward value of whatever successor occurs: running average



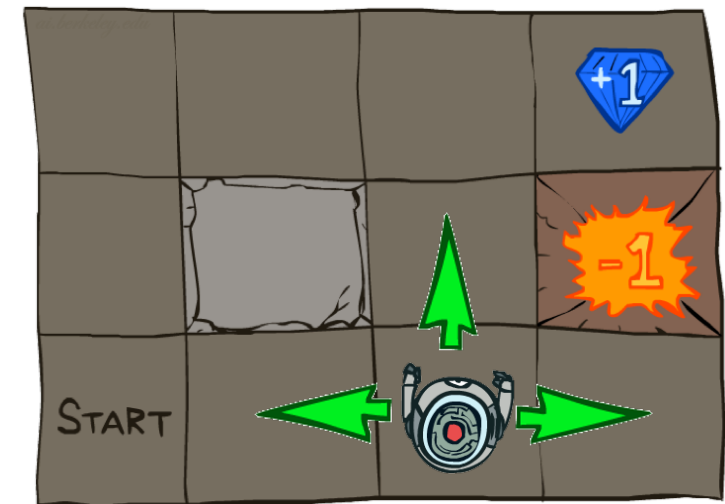
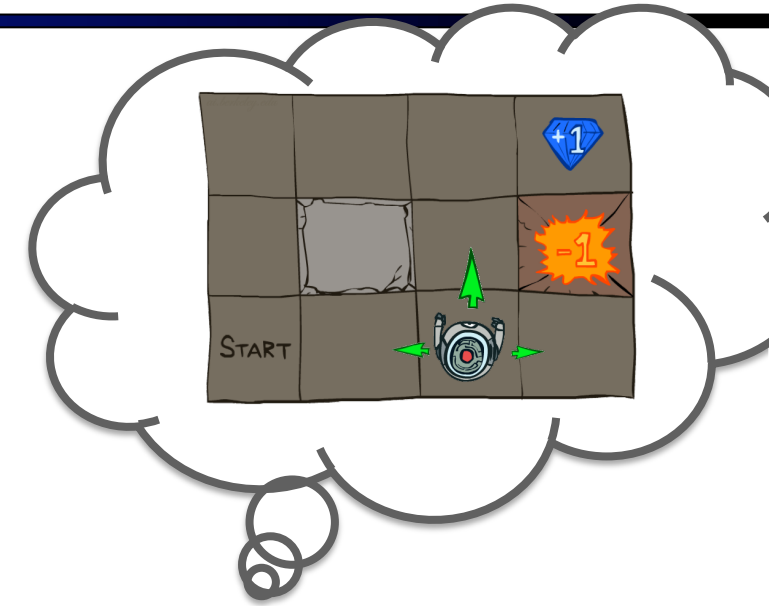
Sample of $V(s)$: $sample = R(s, \pi(s), s') + \gamma V^\pi(s')$

Update to $V(s)$: $V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample$

Same update: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

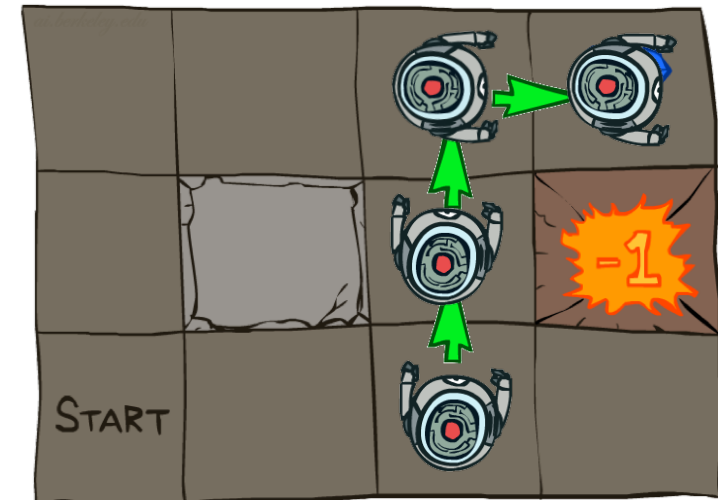
TD as approximate Bellman update

- Given a fixed policy, the value of a state is an expectation over next-state values:
 - $V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')]]$
- Idea 1: Use actual samples to estimate the expectation:
 - $\text{sample}_1 = R(s, \pi(s), s_1') + \gamma V^\pi(s_1')$
 - $\text{sample}_2 = R(s, \pi(s), s_2') + \gamma V^\pi(s_2')$
 - ...
 - $\text{sample}_N = R(s, \pi(s), s_N') + \gamma V^\pi(s_N')$
 - $V^\pi(s) \leftarrow 1/N \sum_i \text{sample}_i$



TD as approximate Bellman update

- Idea 2: Update value of s after each transition s, a, s', r :
- Update $V^\pi([3,1])$ based on $R([3,1], \text{up}, [3,2])$ and $\gamma V^\pi([3,2])$
- Update $V^\pi([3,2])$ based on $R([3,2], \text{up}, [3,3])$ and $\gamma V^\pi([3,3])$
- Update $V^\pi([3,3])$ based on $R([3,3], \text{right}, [4,3])$ and $\gamma V^\pi([4,3])$



TD as approximate Bellman update

- Idea 3: Update values by maintaining a *running average*

Running averages

- How do you compute the average of 1, 4, 7?
- Method 1: add them up and divide by N
 - $1+4+7 = 12$
 - $\text{average} = 12/N = 12/3 = 4$
- Method 2: keep a running average μ_n and a running count n
 - $n=0 \quad \mu_0=0$
 - $n=1 \quad \mu_1 = (0 \cdot \mu_0 + x_1)/1 = (0 \cdot 0 + 1)/1 = 1$
 - $n=2 \quad \mu_2 = (1 \cdot \mu_1 + x_2)/2 = (1 \cdot 1 + 4)/2 = 2.5$
 - $n=3 \quad \mu_3 = (2 \cdot \mu_2 + x_3)/3 = (2 \cdot 2.5 + 7)/3 = 4$
 - General formula: $\mu_n = ((n-1) \cdot \mu_{n-1} + x_n)/n$
 - $= [(n-1)/n] \mu_{n-1} + [1/n] x_n$ (weighted average of old mean, new sample)

Running averages contd.

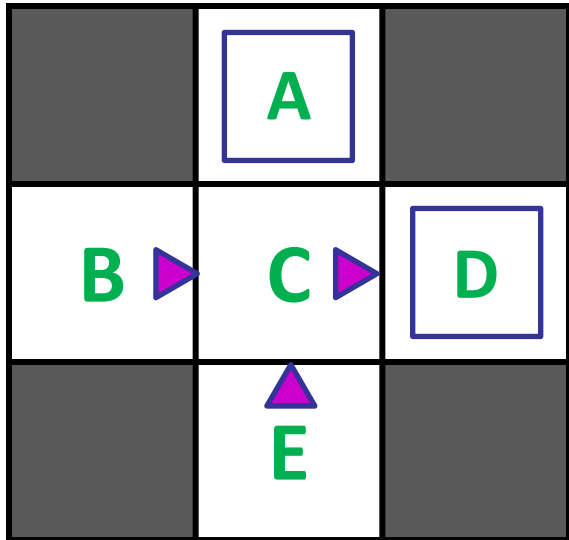
- What if we use a weighted average with a fixed weight?
 - $\mu_n = (1-\alpha) \mu_{n-1} + \alpha x_n$
 - $n=1 \quad \mu_1 = x_1$
 - $n=2 \quad \mu_2 = (1-\alpha) \cdot \mu_1 + \alpha x_2 = (1-\alpha) \cdot x_1 + \alpha x_2$
 - $n=3 \quad \mu_3 = (1-\alpha) \cdot \mu_2 + \alpha x_3 = (1-\alpha)^2 \cdot x_1 + \alpha(1-\alpha)x_2 + \alpha x_3$
 - $n=4 \quad \mu_4 = (1-\alpha) \cdot \mu_3 + \alpha x_4 = (1-\alpha)^3 \cdot x_1 + \alpha(1-\alpha)^2 x_2 + \alpha(1-\alpha)x_3 + \alpha x_4$
- I.e., ***exponential forgetting*** of old values
- μ_n is unbiased

TD as approximate Bellman update

- Idea 3: Update values by maintaining a *running average*
 - $\text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s')$
 - $V^\pi(s) \leftarrow (1-\alpha) \cdot V^\pi(s) + \alpha \cdot \text{sample}$
 - $V^\pi(s) \leftarrow V^\pi(s) + \alpha \cdot [\text{sample} - V^\pi(s)]$
 - This is the *temporal difference learning rule*
 - $[\text{sample} - V^\pi(s)]$ is the “TD error”
 - α is the *learning rate*
- Observe a sample, move $V^\pi(s)$ a little bit to make it more consistent with its neighbor $V^\pi(s')$

Example: TD Value Estimation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

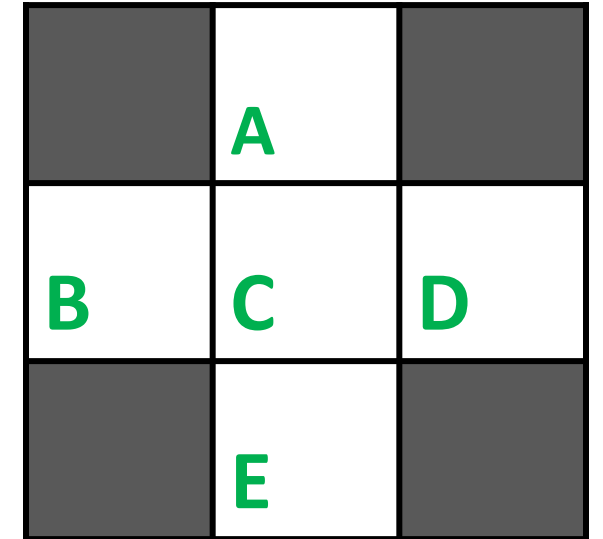
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Example: TD Value Estimation

- Experience transition i : (s_i, a_i, s'_i, r_i) .
- Compute sampled value “target”: $r_i + \gamma V^\pi(s'_i)$.
- Compute “TD error”: $\delta_i = (r_i + \gamma V^\pi(s'_i)) - V^\pi(s_i)$.
- Update: $V^\pi(s_i) += \alpha_i \cdot \delta_i$.

B, east, C, -1
C, east, D, -1
D, exit, x, +10

B, east, C, -1
C, east, D, -1
D, exit, x, +10

E, north, C, -1
C, east, D, -1
D, exit, x, +10

E, north, C, -1
C, east, A, -1
A, exit, x, -10

s	V(s)
A	
B	
C	
D	
E	

i	s	a	s'	r	$r + \gamma V^\pi(s')$	$V^\pi(s)$	δ
1							
2							
3							
4							
5							
6							
7							

Example: TD Value Estimation

- Experience transition $i: (s_i, a_i, s'_i, r_i)$.
- Compute sampled value “target”: $r_i + \gamma V^\pi(s'_i)$.
- Compute “TD error”: $\delta_i = (r_i + \gamma V^\pi(s'_i)) - V^\pi(s_i)$.
- Update: $V^\pi(s_i) += \alpha_i \cdot \delta_i$.

B, east, C, -1
C, east, D, -1
D, exit, x, +10

B, east, C, -1
C, east, D, -1
D, exit, x, +10

E, north, C, -1
C, east, D, -1
D, exit, x, +10

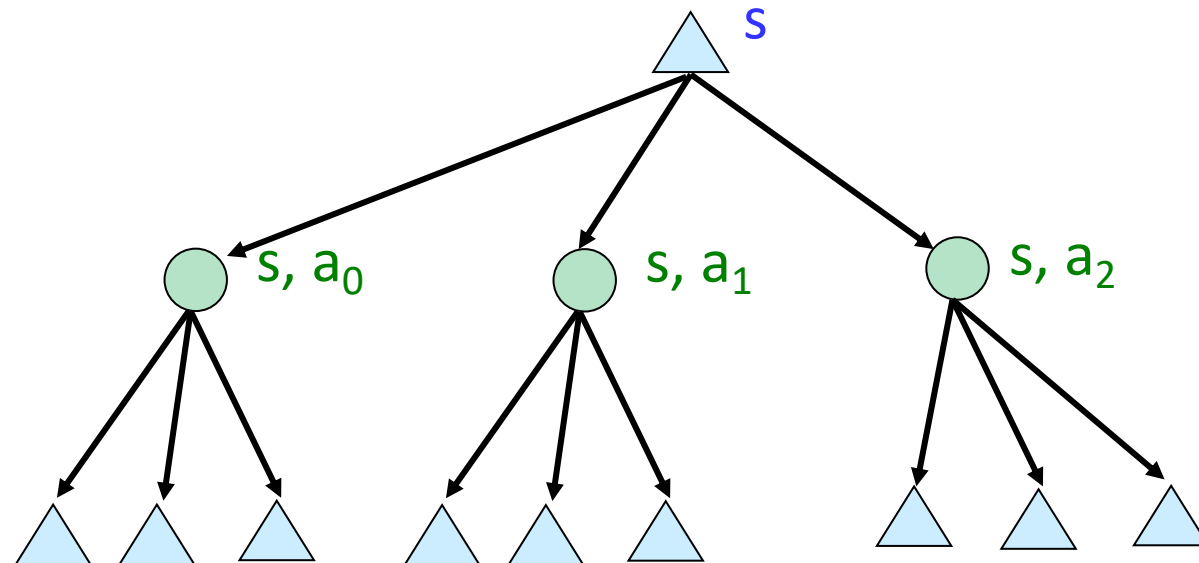
E, north, C, -1
C, east, A, -1
A, exit, x, -10

s	V(s)
A	0
B	-2
C	9
D	10
E	8

i	s	a	s'	r	$r + \gamma V^\pi(s')$	$V^\pi(s)$	δ
1	B	east	C	-1	$-1 + 0$	0	-1
2	C	east	D	-1	$-1 + 0$	0	-1
3	D	exit	---	10	$10 + 0$	0	+10
4	B	east	C	-1	$-1 + -1$	-1	-1
5	C	east	D	-1	$-1 + 10$	-1	+10
6	D	exit	---	10	$10 + 0$	10	0
7	E	north	C	-1	$-1 + 9$	0	+8

Problems with TD Value Learning

- Model-free policy evaluation! 🎉
- Bellman updates with running sample mean! 🎉



- Need the transition model to improve the policy! 🤖

Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values

- Start with $V_0(s) = 0$, which we know is right
- Given V_k , calculate the depth $k+1$ values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- But Q-values are more useful, so compute them instead

- Start with $Q_0(s,a) = 0$, which we know is right
- Given Q_k , calculate the depth $(k+1)$ q-values for all q-states:

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

Q-learning as approximate Q-iteration

- Recall the definition of Q values:
 - $Q^*(s,a)$ = expected return from doing a in s and then behaving optimally thereafter; and $\pi^*(s) = \max_a Q^*(s,a)$
- Bellman equation for Q values:
 - $Q^*(s,a) = \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]]$
- Approximate Bellman update for Q values:
 - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a')]]$
- We obtain a policy from learned $Q(s,a)$, with no model!
 - (No free lunch: $Q(s,a)$ table is $|A|$ times bigger than $V(s)$ table)

- $$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$



[Demo: Q-learning – crawler (L10D3)]

Video of Demo Q-Learning -- Gridworld

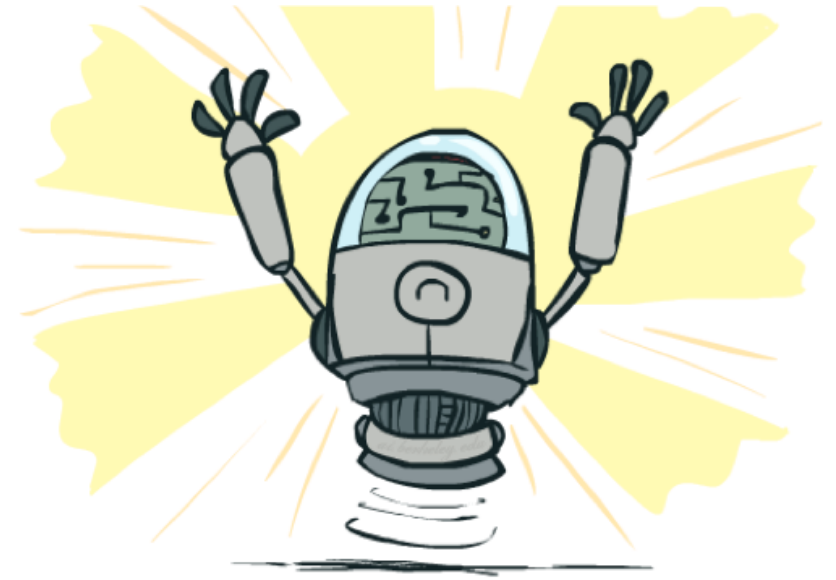


Video of Demo Q-Learning -- Crawler



Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if samples are generated from a suboptimal policy!
- This is called **off-policy learning**
- Caveats:
 - You have to explore enough
 - You have to eventually make the learning rate small enough
 - ... but not decrease it too quickly
 - Basically, in the limit, it doesn't matter how you select actions (!)



Summary

- RL solves MDPs via direct experience of transitions and rewards
- There are several approaches:
 - Learn the MDP model and solve it
 - Learn V directly from sums of rewards, or by TD local adjustments
 - Still need a model to make decisions by lookahead
 - Learn Q by local Q-learning adjustments, use it directly to pick actions
 - (and about 100 other variations)
- Big missing pieces:
 - How to explore without too much regret?
 - How to scale this up to Tetris (10^{60}), Go (10^{172}), StarCraft ($|A|=10^{26}$)?