Artificial Intelligence

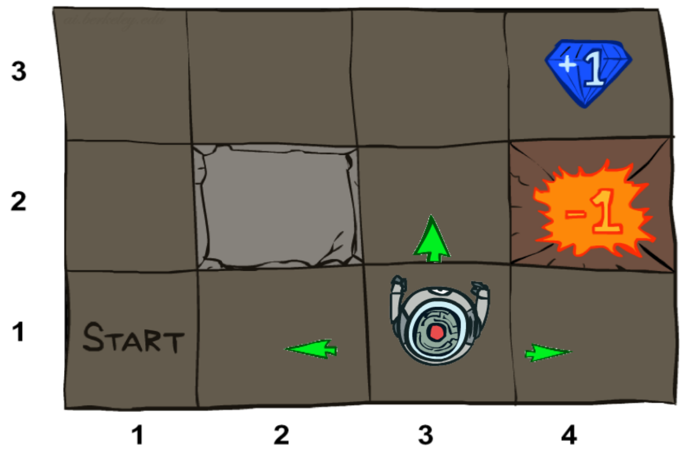
22. Reinforcement Learning

Shashi Prabh

School of Engineering and Applied Science
Ahmedabad University

Recap: Markov Decision Process (MDP)

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition model T(s, a, s')
    - Probability that a from s leads to s', i.e., P (s'| s, a)
  - A reward function R(s, a, s') for each transition
  - A start state
  - Possibly a terminal state (or absorbing state)
  - Utility function which is additive (discounted) rewards



- MDPs are fully observable but probabilistic search problems

[Demo – gridworld]

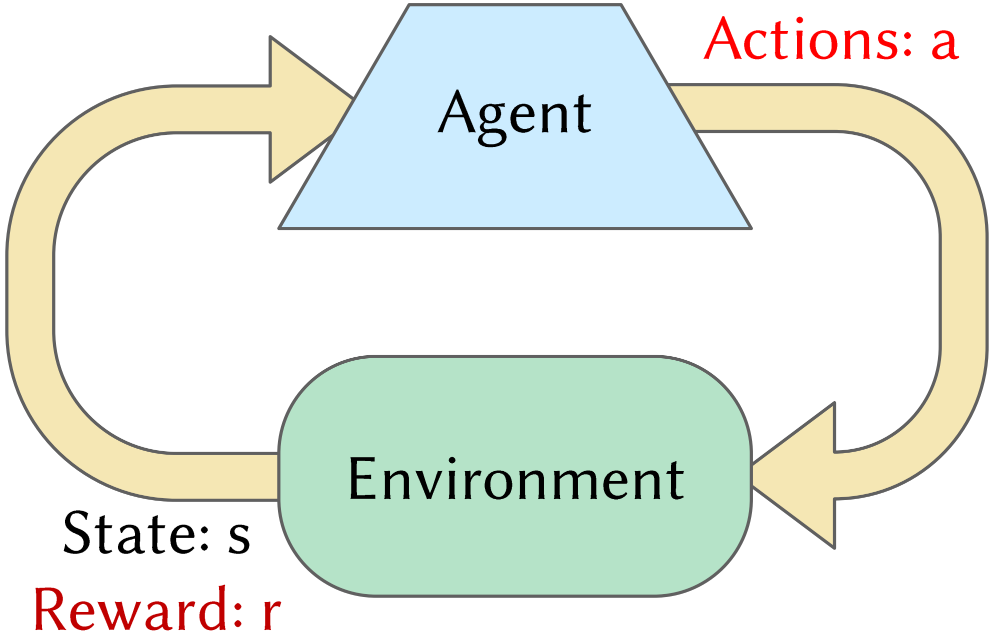
Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A 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
  - That is, we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Reinforcement Learning Loop

- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

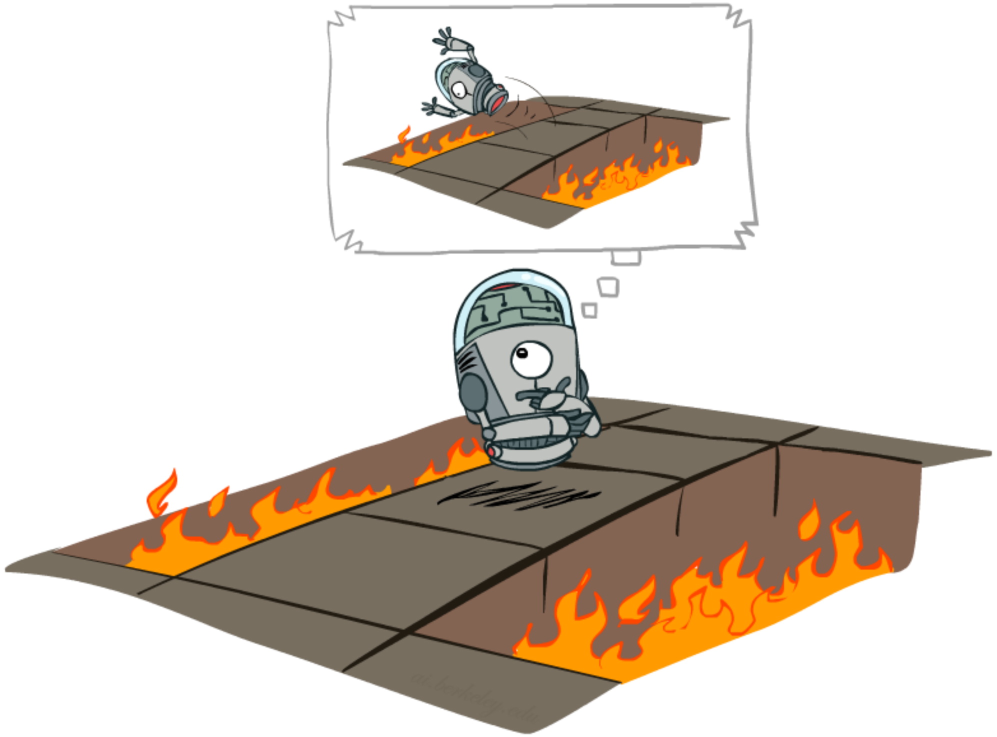


Reinforcement learning

Basic ideas:

- **Exploration**: you have to **try unknown actions** to get information
- **Exploitation**: eventually, you have to use what you know
- **Sampling**: you may need to repeat many times to get good estimates
- **Generalization**: what you learn in one state may apply to others too

Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

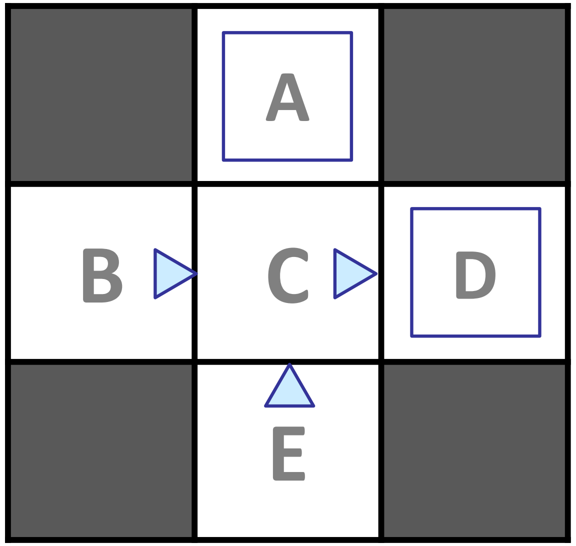
Model-Based Learning

- 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
  - Normalize to give an estimate of $\hat{T}(s, a, s')$
  - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')

- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Example: Model-Based Learning

Input Policy π



Assume: γ = 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

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

Pros and cons

- Pro:
    - Makes efficient use of experiences
- Cons:
    - May not scale to large state spaces
        - Learns model one state-action pair at a time (but this is fixable)
        - Cannot solve MDP for very large $|S|$

Analogy: Expected Age

Goal: Compute expected age of students

Known P(A)

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

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

Unknown P(A): "Model Based"

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

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

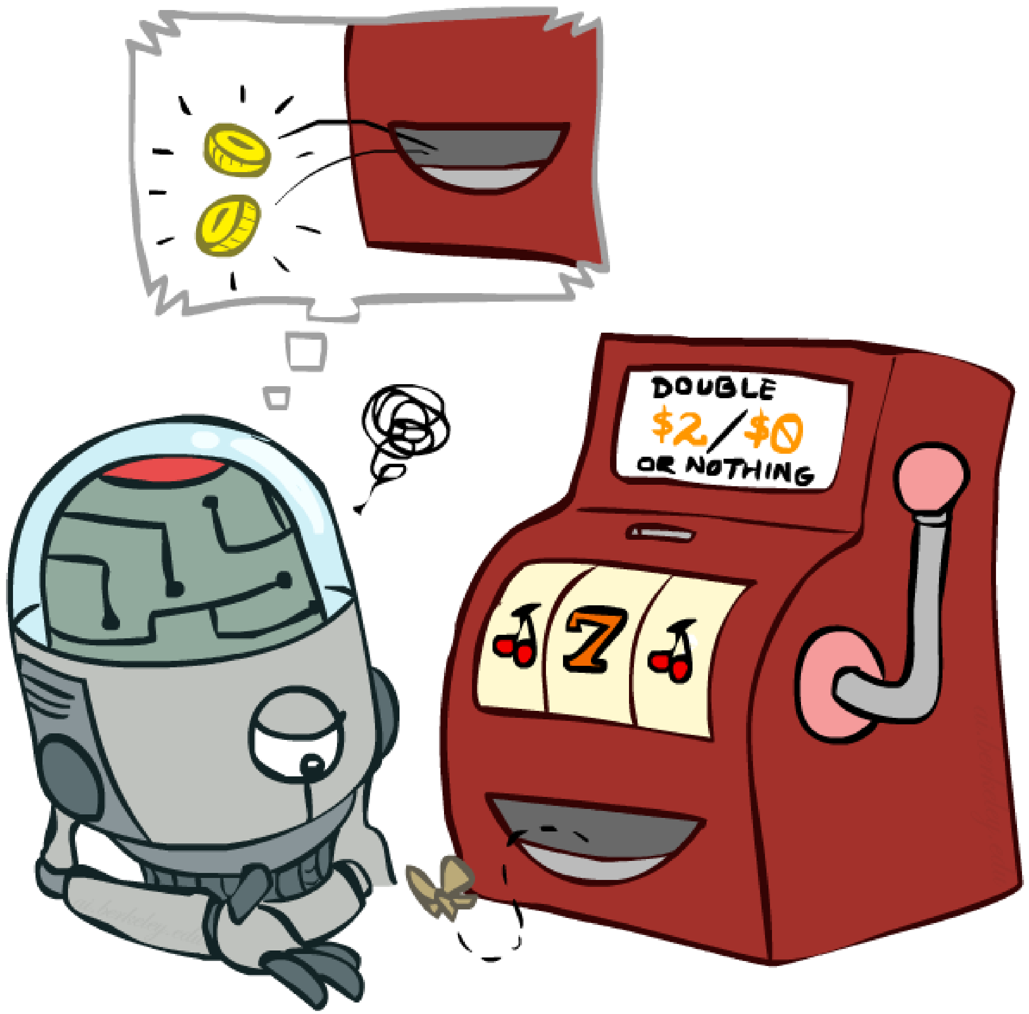
Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

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

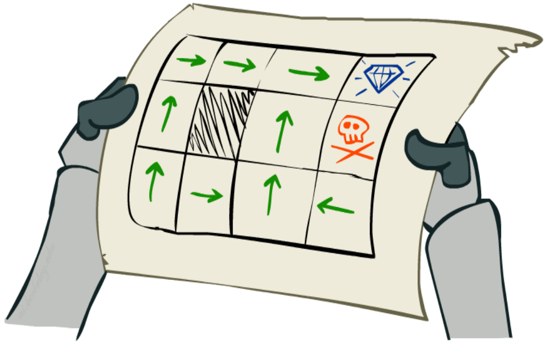
Why does this work? Because samples appear with the right frequencies.

Model-Free Learning



Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You don't know the transitions $T(s,a,s')$
  - You don't know the rewards $R(s,a,s')$
  - Goal: learn the state values



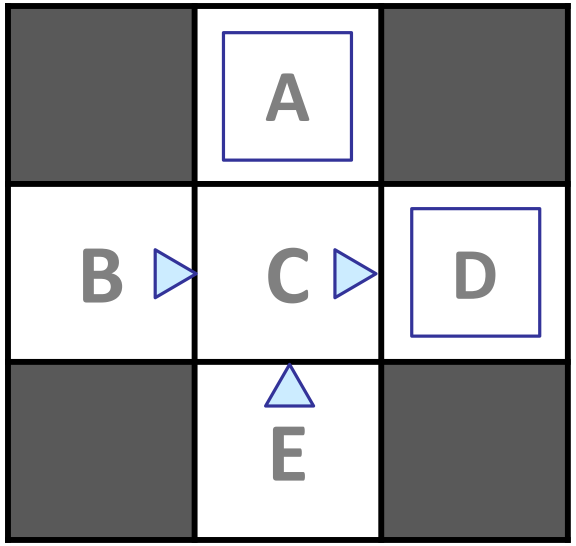
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

Direct Evaluation

- Goal: Compute values for each state under π
- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples
- This is called direct evaluation

Example: Direct Evaluation

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

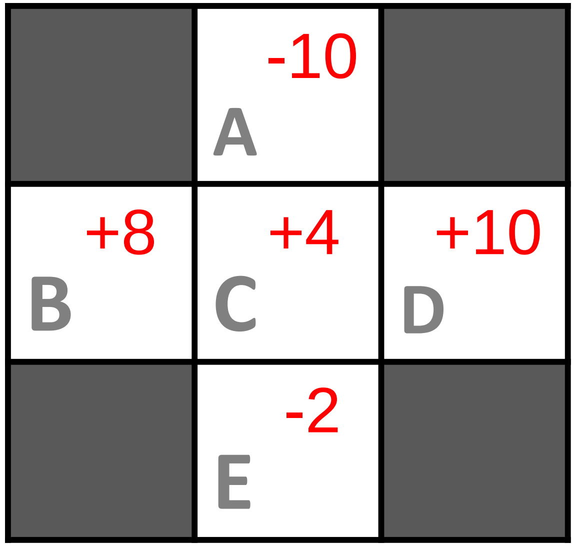
Episode 3

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

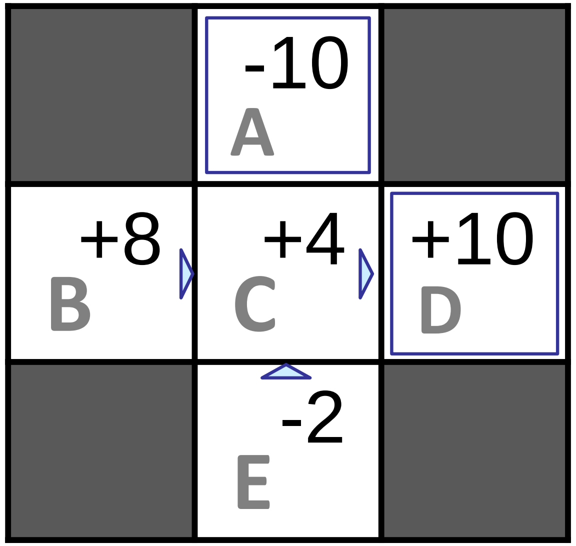


If B and E both go to C under this policy, how can their values be different?

Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions
- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

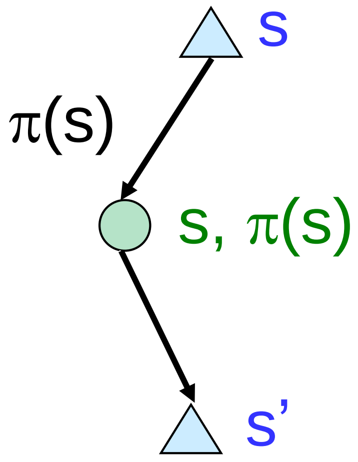
Output Values



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

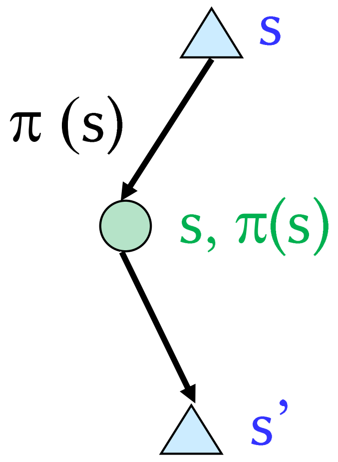


Temporal Difference Learning

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: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$



Exponential Moving Average

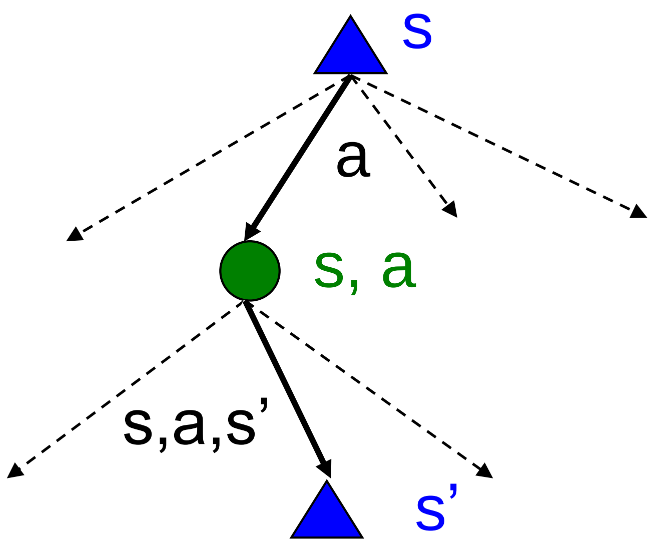
- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1-\alpha)\cdot\bar{x}_{n-1} + \alpha\cdot x_n$
  - Makes recent samples more important
  - Forgets about the past (distant past values were wrong anyway)
- Decreasing learning rate (alpha) can give converging averages

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, turning values into a new policy is not possible
  - Values only tell you the expected future reward of a state, not the value of taking a specific *action* within that state
- Idea: learn Q-values, not values
  - Makes action selection model-free too!

$$\pi(s) = \arg\max_a Q(s,a)$$

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



Recap: 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:

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

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$
  - Given Q_k, calculate the depth $k+1$ q-values for all q-states:

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

Q-Learning

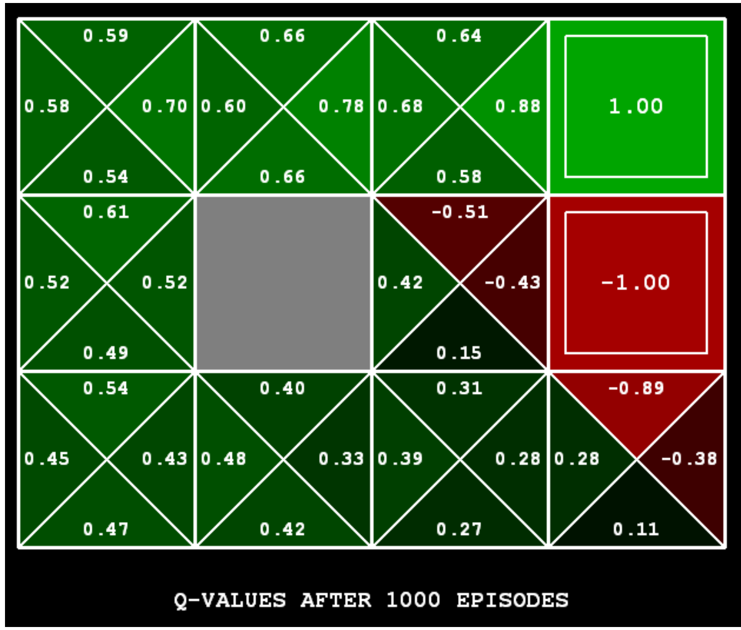
- Q-Learning: sample-based Q-value iteration

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

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

No longer policy evaluation!

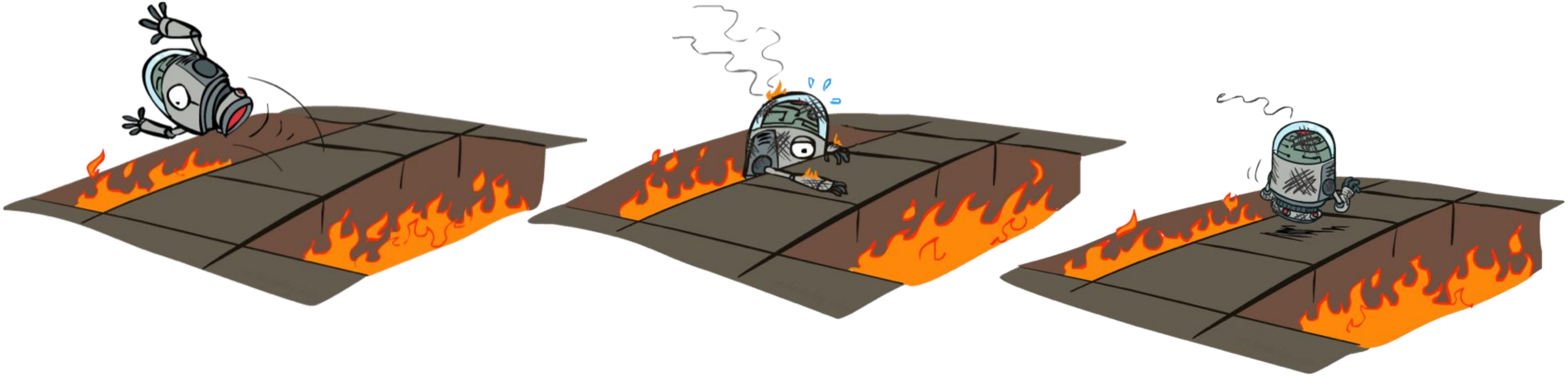


  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

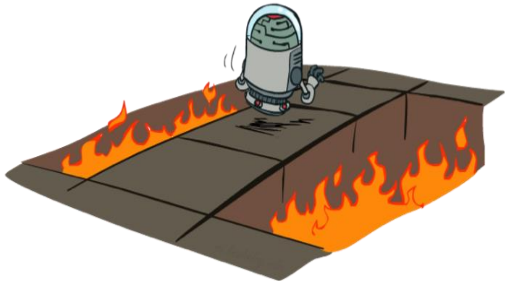
Active Reinforcement Learning

- Passive reinforcement learning:
  - A passive learning agent has a fixed policy that determines its behavior
- Active reinforcement learning:
  - An active learning agent gets to decide what actions to take



Q-Learning: Explore and Exploit

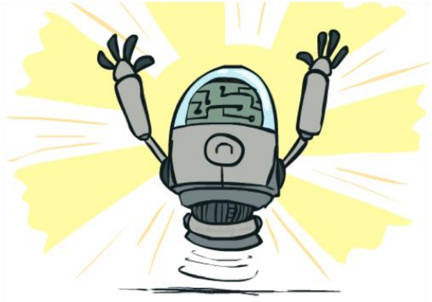
- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
  - 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 (!)

Frozen-Lake Demo

Quiz

- Which of the following best characterizes RL?

  A. Learning from labelled examples provided by a teacher

  B. Learning an explicit model of the environment before acting

  C. Learning through trial and error by interacting with the environment and receiving rewards

  D. Learning by memorizing optimal policies from a dataset

Quiz

- Which of the following best characterizes RL?
  A. Learning from labelled examples provided by a teacher
  B. Learning an explicit model of the environment before acting
  C. Learning through trial and error by interacting with the environment and receiving rewards
  D. Learning by memorizing optimal policies from a dataset

- In model-free RL methods such as Q-learning

A. The agent must know $P(s'|s, a)$ before learning begins.
B. The agent learns directly from experience without an explicit model of transitions or rewards.
C. The agent uses a known model to simulate rollouts.
D. The agent uses logical inference instead of sampling.

- In model-free RL methods such as Q-learning

  A. The agent must know $P(s'|s, a)$ before learning begins.

  B. The agent learns directly from experience without an explicit model of transitions or rewards.

  C. The agent uses a known model to simulate rollouts.

  D. The agent uses logical inference instead of sampling.

- Which equation correctly represents the Q-learning update?
  A. $Q(s,a) \leftarrow R(s,a)$
  B. $Q(s,a) \leftarrow Q(s,a)+\alpha[R+ \gamma \max_{a'} Q(s',a')-Q(s,a)]$
  C. $V(s)=\max_a Q(s,a)$
  D. $Q(s,a)=\alpha R(s,a)+(1-\alpha)Q(s,a)$

- Which equation correctly represents the Q-learning update?
  A. $Q(s,a) \leftarrow R(s,a)$
  B. $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  C. $V(s) = \max_{a} Q(s,a)$
  D. $Q(s,a) = \alpha R(s,a) + (1 - \alpha) Q(s,a)$

- Why is the exploration–exploitation trade-off fundamental in reinforcement learning?
  A. Because agents must randomly switch policies during training
  B. Because exploration increases rewards in deterministic environments
  C. Because exploitation is only useful after convergence
  D. Because the agent must balance learning new information with using what it already knows to maximize reward
- Why is reinforcement learning central to today's AI breakthroughs (e.g., AlphaGo, robotics, ChatGPT fine-tuning)?
  A. It formalizes how agents can learn sequential behaviors to maximize cumulative reward through experience.
  B. It provides the theoretical basis for reasoning with logical rules.
  C. It replaces deep learning entirely.
  D. It ensures perfect optimality in stochastic environments.

- Why is the exploration–exploitation trade-off fundamental in reinforcement learning?
  A. Because agents must randomly switch policies during training
  B. Because exploration increases rewards in deterministic environments
  C. Because exploitation is only useful after convergence
  D. Because the agent must balance learning new information with using what it already knows to maximize reward
- Why is reinforcement learning central to today’s AI breakthroughs (e.g., AlphaGo, robotics, ChatGPT fine-tuning)?
  A. It formalizes how agents can learn sequential behaviors to maximize cumulative reward through experience.
  B. It provides the theoretical basis for reasoning with logical rules.
  C. It replaces deep learning entirely.
  D. It ensures perfect optimality in stochastic environments.

Summary

- In RL, agents interact with the environment via state, action, reward, next state loop
  - Goal: maximize expected cumulative reward
- Model-free learning involves estimating Q(s, a) directly from experience
- Temporal-Difference (TD) learning updates current estimates using future predictions
- Exploration–exploitation trade-off balances learning new strategies vs. using known good ones