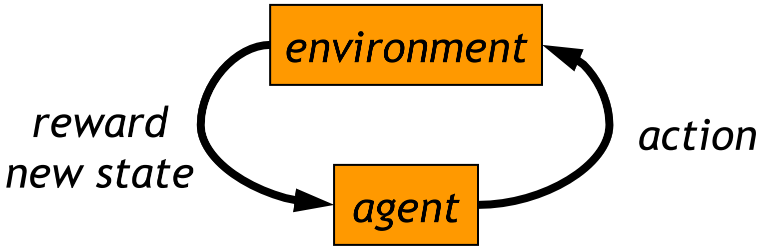
Artificial Intelligence

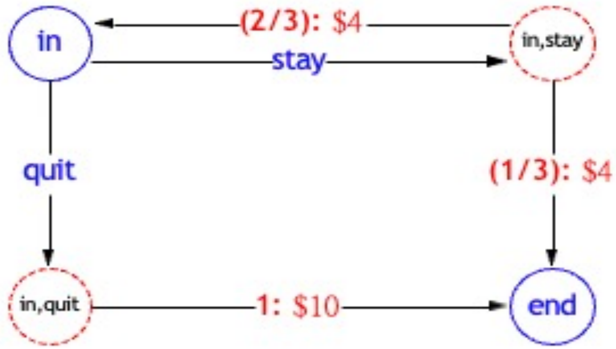
Lecture 7: Reinforcement Learning

Review: Artificial Intelligence

- Supervised learning
  - Classification
  - Regression
- Unsupervised learning
  - Clustering
  - Dimensionality reduction
- Reinforcement learning
  - more general than supervised/unsupervised learning
  - learn from interaction w/ environment to achieve a goal



Review: Markov Decision Process (MDPs)



Definition: Markov decision process

States: the set of states

$s_{start} \in$ States: starting state

Actions(s): possible actions from state s

$T(s, a, s')$: probability of s' if take action a in state s

Reward (s, a, s'): reward for the transition (s, a, s')

IsEnd(s): whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

Review: Markov Decision Process (MDPs)

- Following a policy π produces a path (episode)

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$$

- **Value** function $V_\pi(s)$: expected utility if follow π from state s

$$V_\pi(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

- **Q-value** function $Q_\pi(s, a)$: expected utility if first take action a from state s and then follow π

$$Q_\pi(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_\pi(s')]$$

Unknown transitions and rewards

Definition: Markov decision process

States: the set of states

$s_{start} \in$ States: starting state

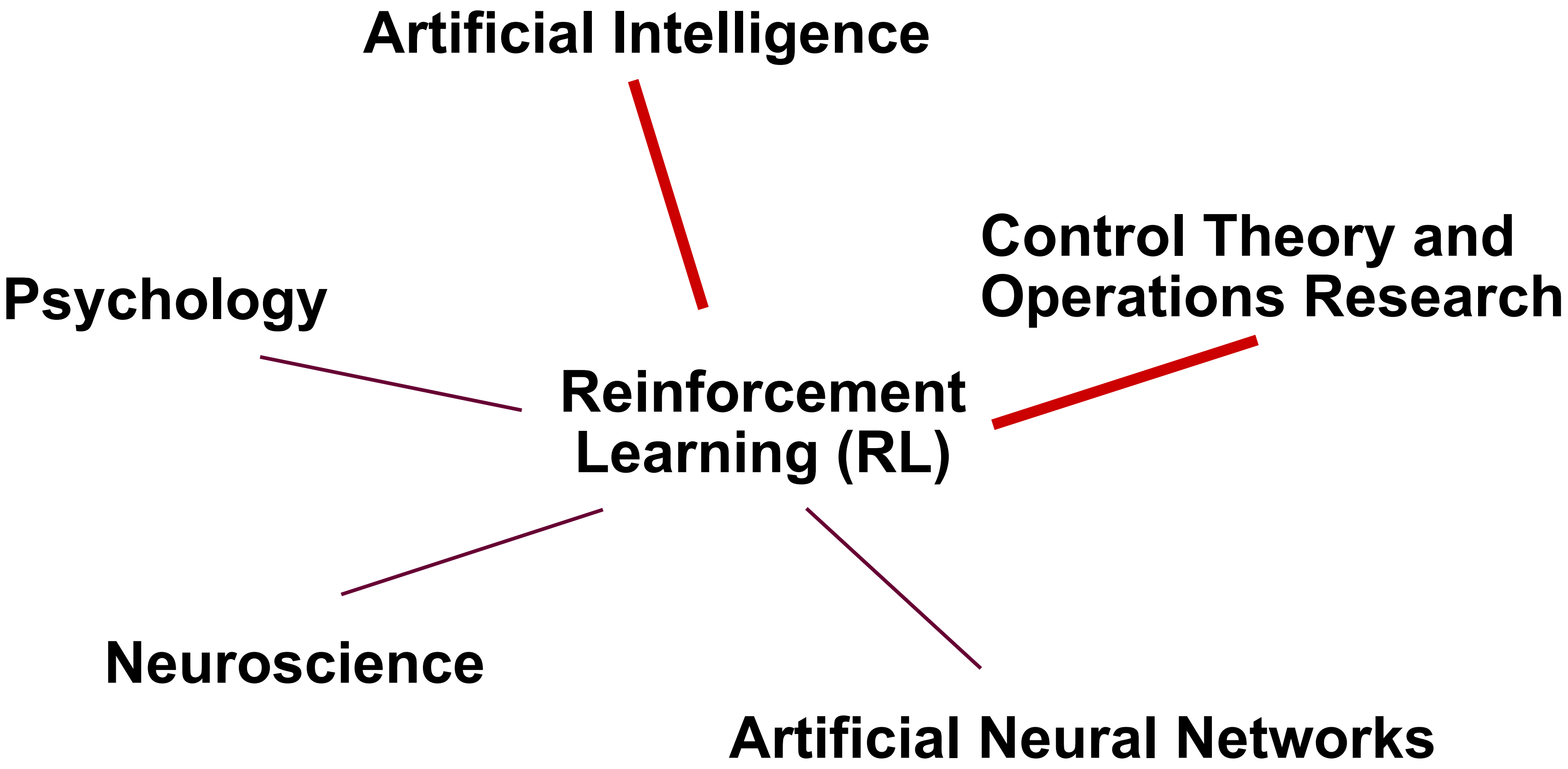
Actions(s): possible actions from state s

IsEnd(s): whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

Learn to make good sequences of decisions

Learning from Experience Plays a Role in …



What is Reinforcement Learning

Fundamental challenge in artificial intelligence and machine learning is learning to make good decisions under uncertainty

What is Reinforcement Learning

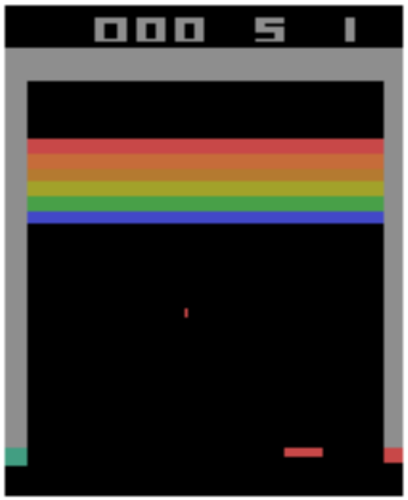
- People and animals learn by interacting with our environment
- This differs from certain other types of learning
  - It is active rather than passive
  - Interactions are often sequential — future interactions can depend on earlier ones
- We are goal-directed
- We can learn without examples of optimal behaviour
- Instead, we optimise some reward signal

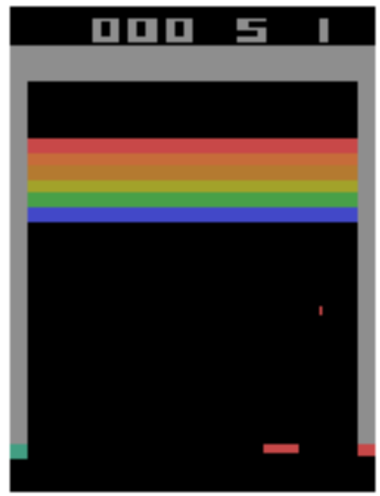
The reward hypothesis

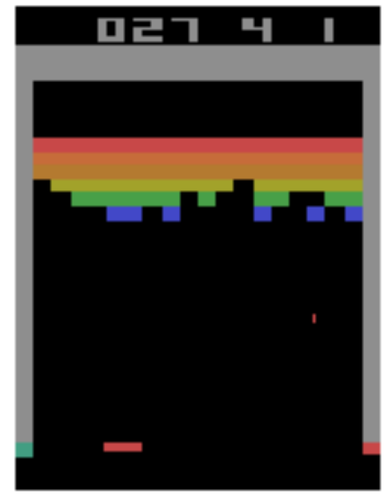
Reinforcement learning is based on the reward hypothesis:

Any goal can be formalized as the outcome of maximizing a cumulative reward

2010s: New Era of RL. Atari







Before any training In early stages of training In later stages of training

https://www.youtube.com/watch?v=V1eYniJ0Rnk

DeepMind Nature, 2015

Mystery game

Example: mystery buttons

For each round r = 1, 2, . . .

- You choose A or B
- You move to a new state and get some rewards

Start   *A*   *B*

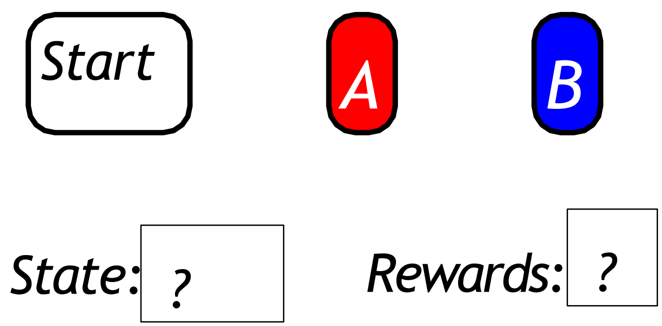
State: 5,0   *Rewards:* 0

Mystery game

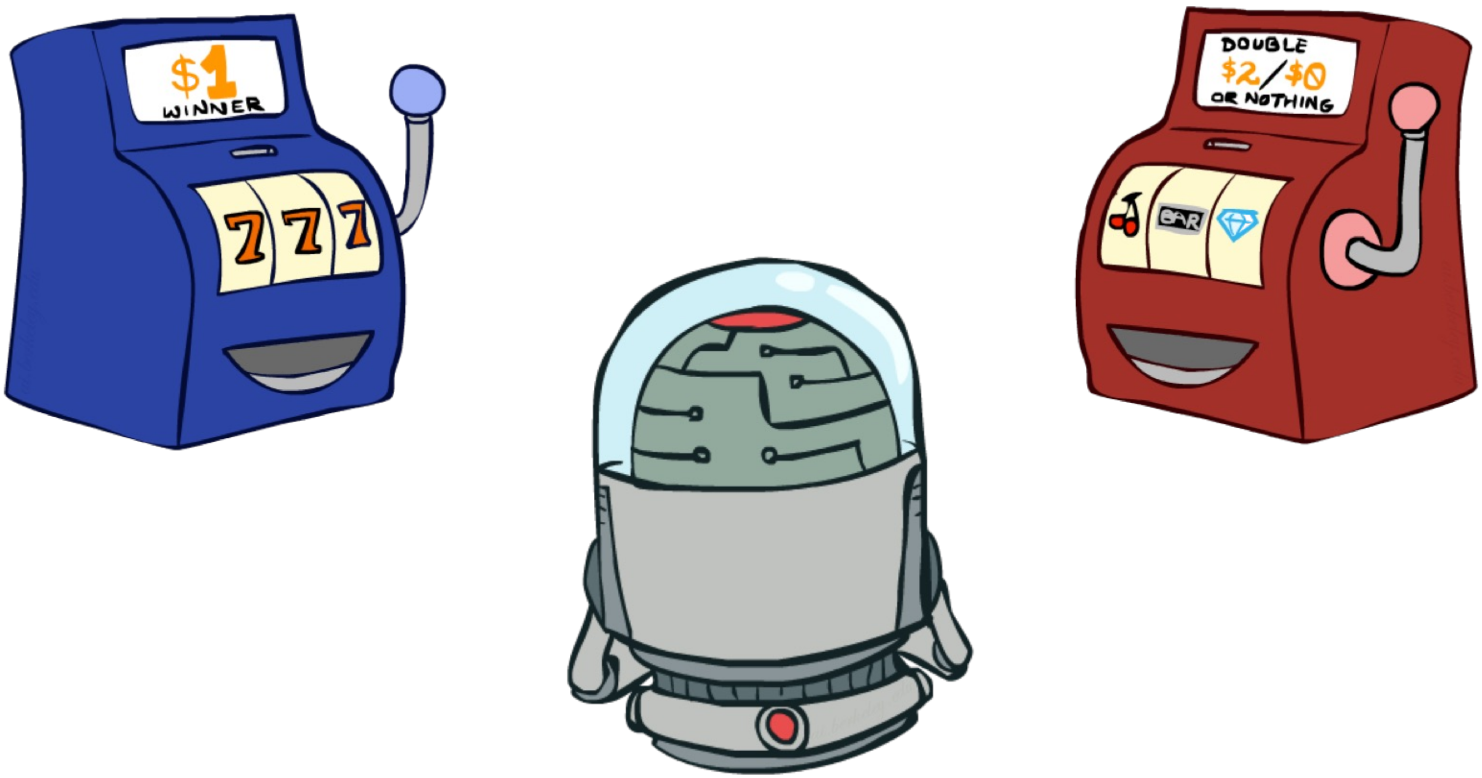
Example: mystery buttons

For each round r = 1, 2, . . .

- You choose A or B
- You move to a new state and get some rewards



Double Bandits



Slides borrowed or adapted from Dan Klein and Pieter Abbeel (ai.berkeley.edu)

Offline Planning

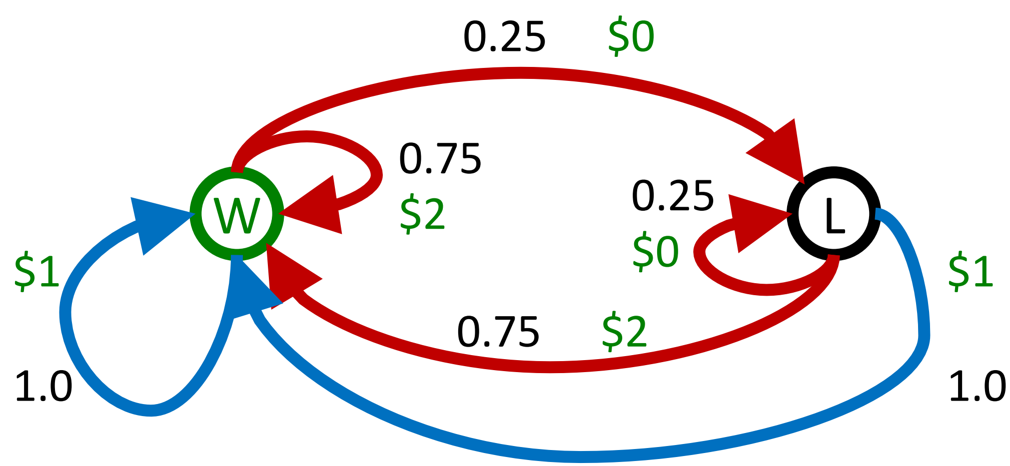
- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

No discount

100 time steps

Both states have the same value

	Value
Play Red	150
Play Blue	100



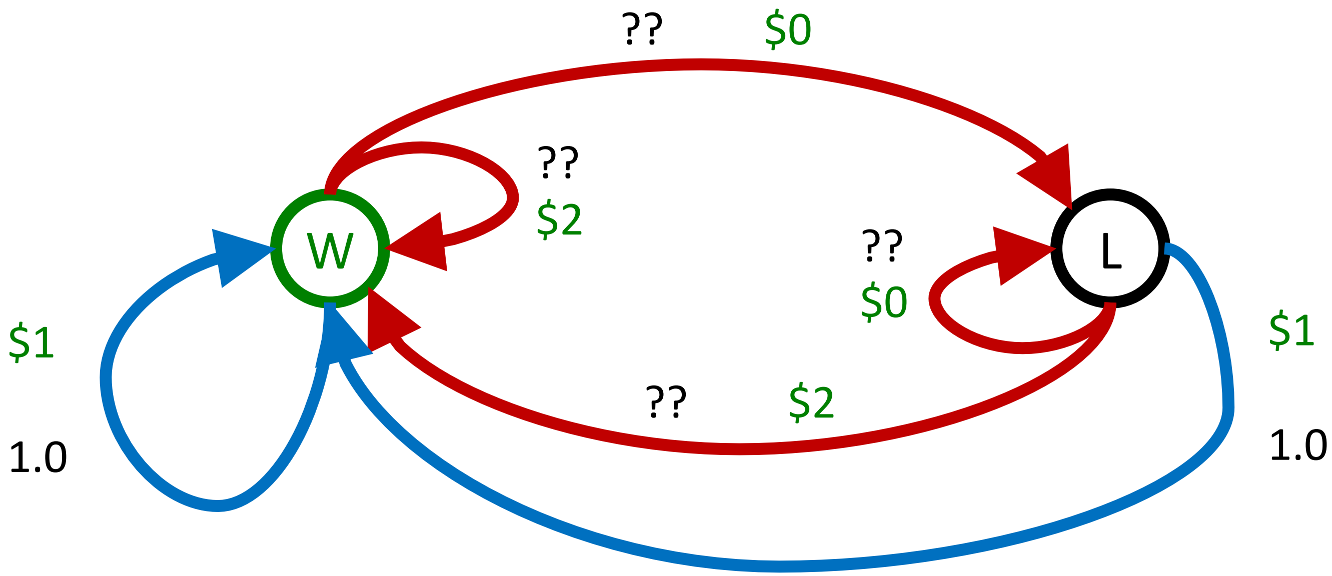
Let's Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

- Rules changed! Red's win chance is different.



Let's Play!

$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out

- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP

From MDPs to reinforcement learning

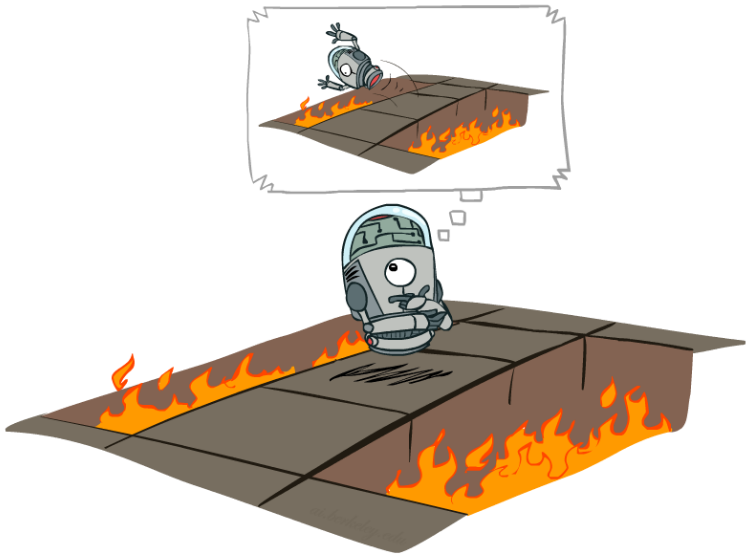
Markov decision process (offline)

- Have mental model of how the world works.
- Find policy to collect maximum rewards.

Reinforcement learning (online)

- Don’t know how the world works.
- Perform actions in the world to find out and collect rewards.

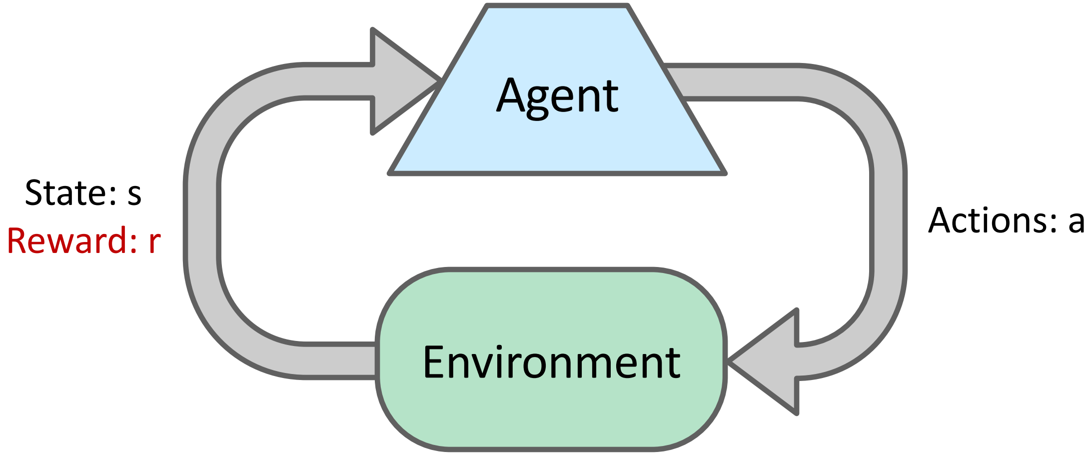
Offline (MDPs) vs. Online (RL)



Offline Solution

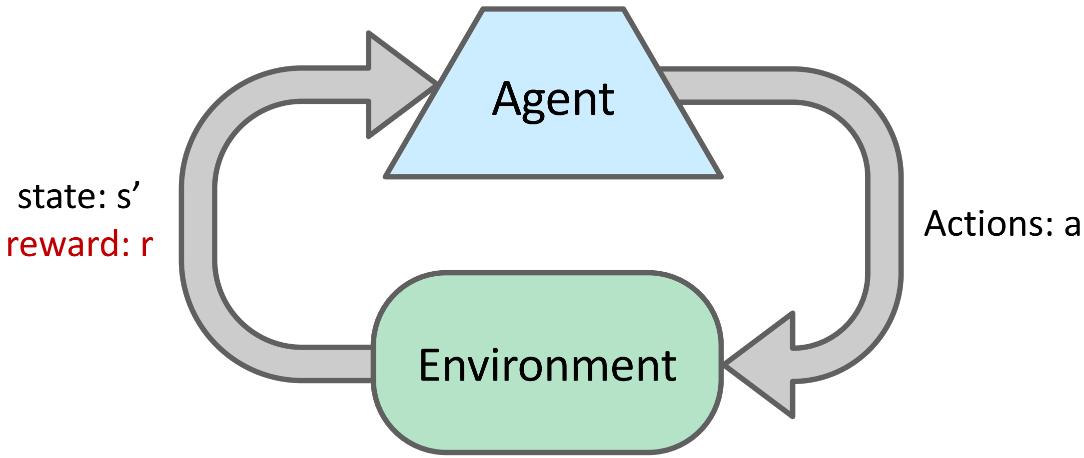
Online Learning

Reinforcement Learning



- 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



Algorithm: reinforcement learning

For t = 1, 2, 3, . . .

Choose action $a_t = \pi_{act}(s_{t-1})$ (**how?**)

Receive reward r_t and observe new state s_t

Update parameters (**how?**)

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
  - I.e. we don't know which states are good or what the actions do
  - Must actually try out actions and states to learn

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 $\widehat{T}(s, a, s')$
  - Discover each $\widehat{R}(s, a, s')$ when we experience (s, a, s')
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Model-Based Learning

Data: $s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots ; a_n, r_n, s_n$

Key idea: model-based learning

Estimate the MDP: T (s, a, s') and Reward(s, a, s')

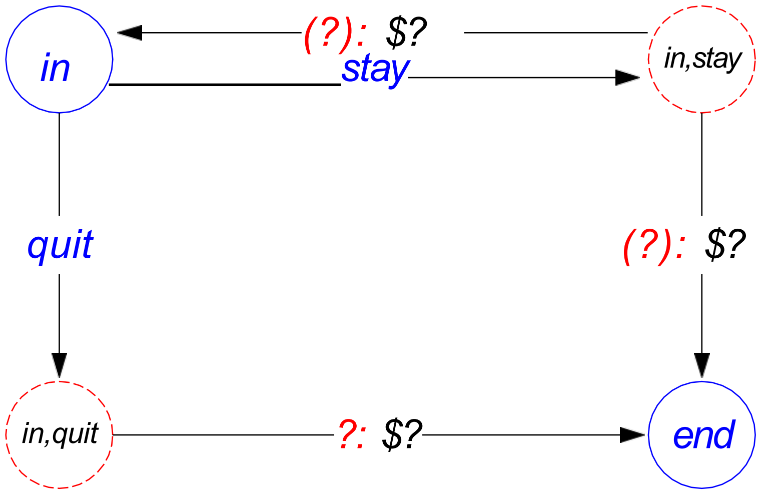
Transitions:

$$\hat{T}(s, a, s') = \frac{\#\text{ times } (s, a, s') \text{ occurs}}{\#\text{ times } (s, a) \text{ occurs}}$$

Rewards:

$$\widehat{\text{Reward}}(s, a, s') = r \text{ in } (s, a, r, s')$$

Model-Based Learning

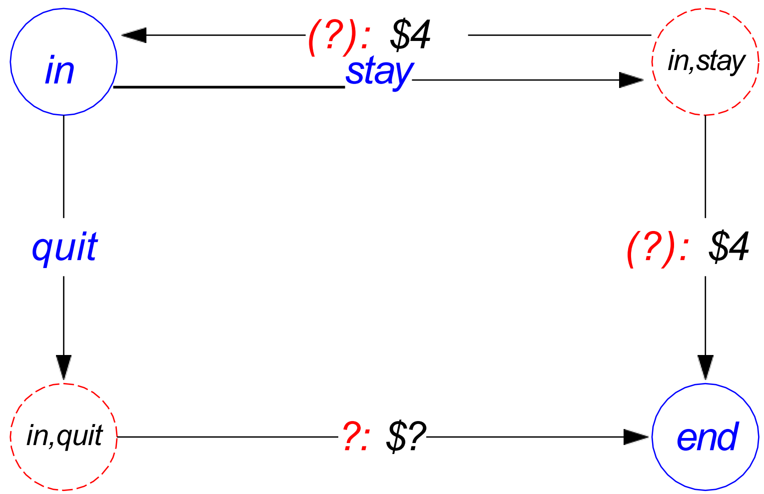


Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

Transition?
Reward?

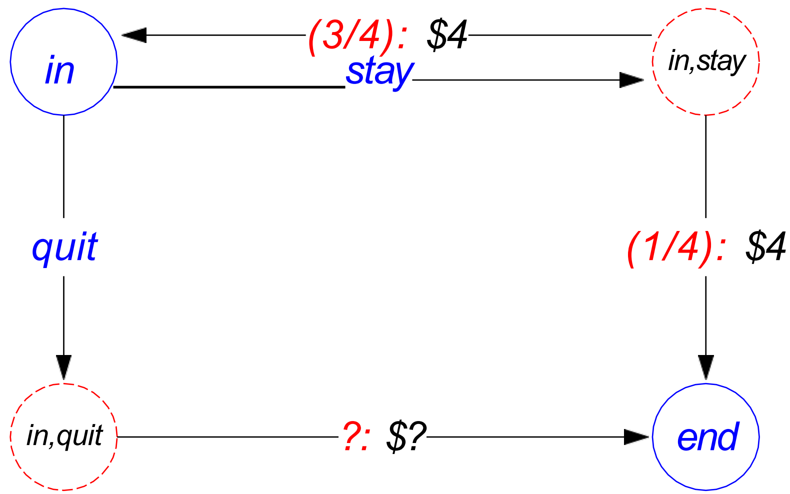
Model-Based Learning



Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

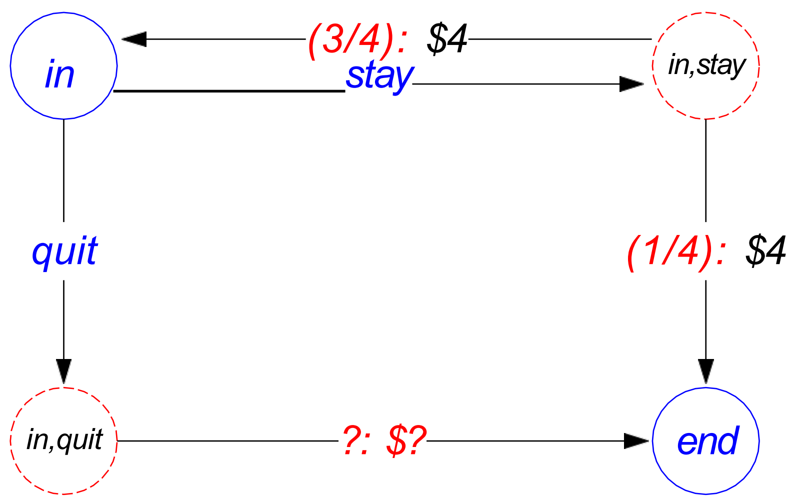
Model-Based Learning



Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

Model-Based Learning

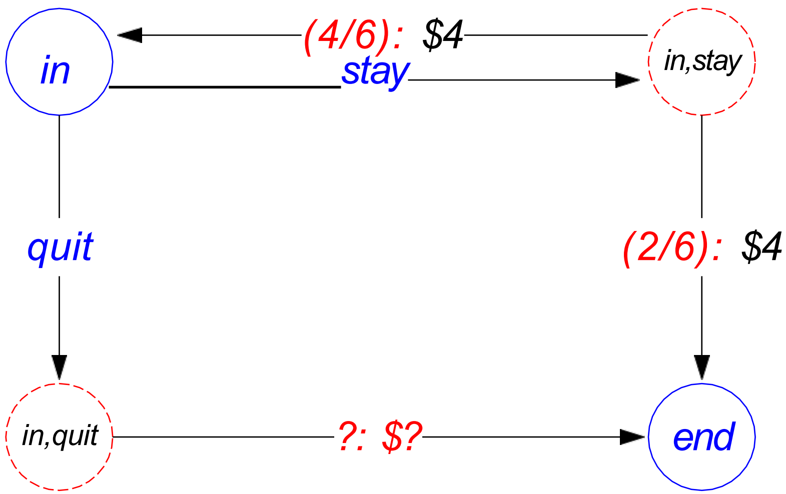


Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, end]

New Transition?

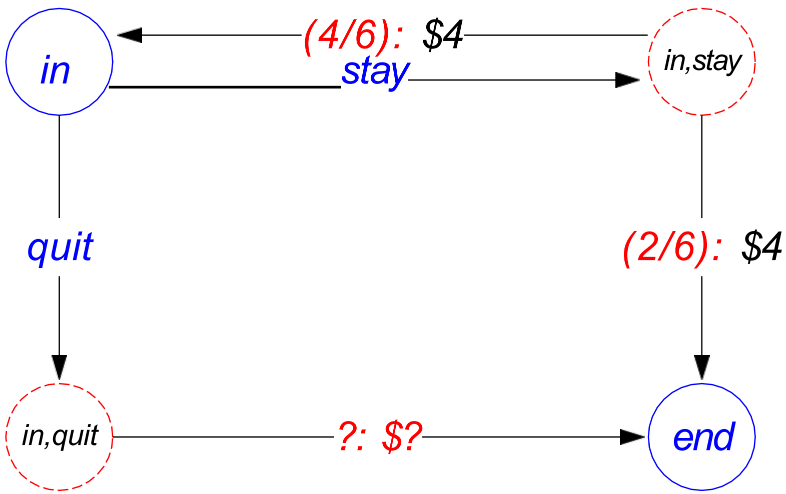
Model-Based Learning



Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, end]

Model-Based Learning

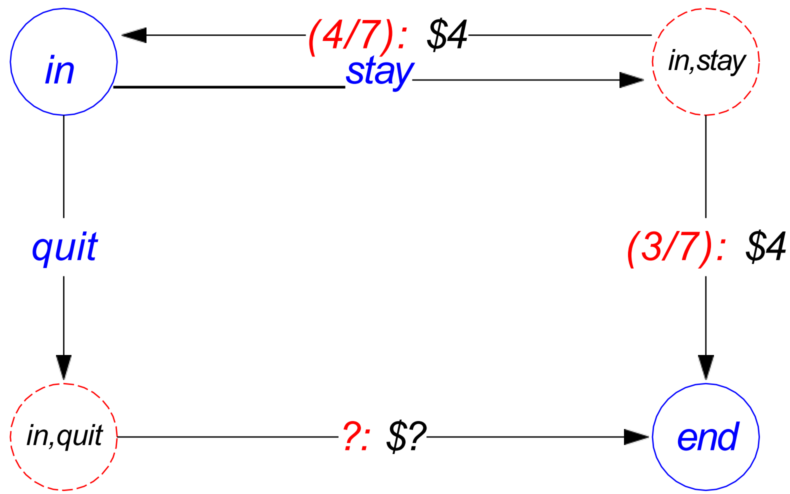


Data (following policy π(s) = stay):

[in; stay, 4, end]

New Transition?

Model-Based Learning

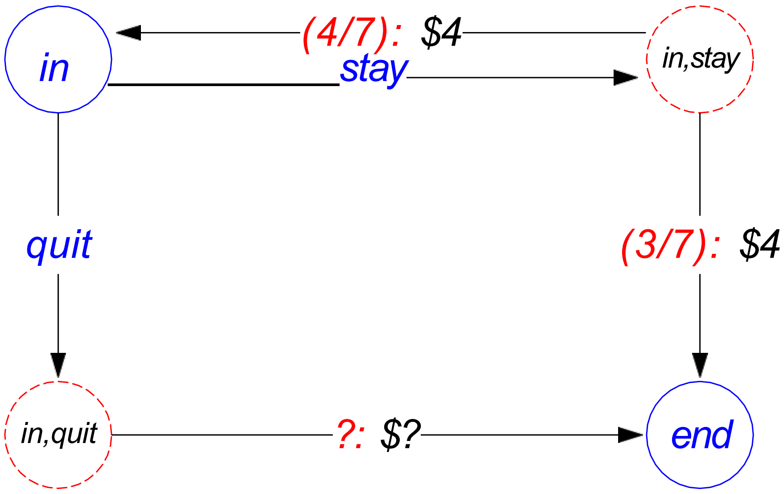


Data (following policy π(s) = stay):

[in; stay, 4, end]

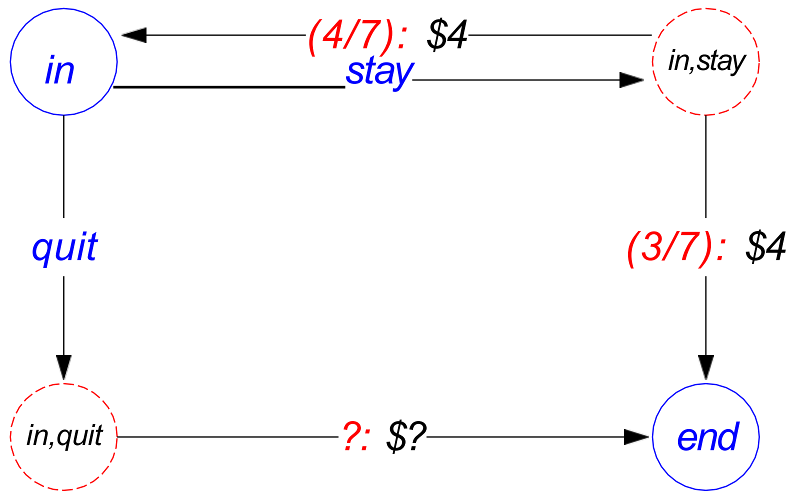
- Estimates converge to true values (under certain conditions)
- With estimated MDP (T, Reward), compute policy using value iteration

Model-Based Learning

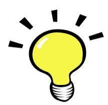


Problem: ?

Model-Based Learning



Problem: won't even see (s, a) if a $\neq \pi(s)$ (a = quit)

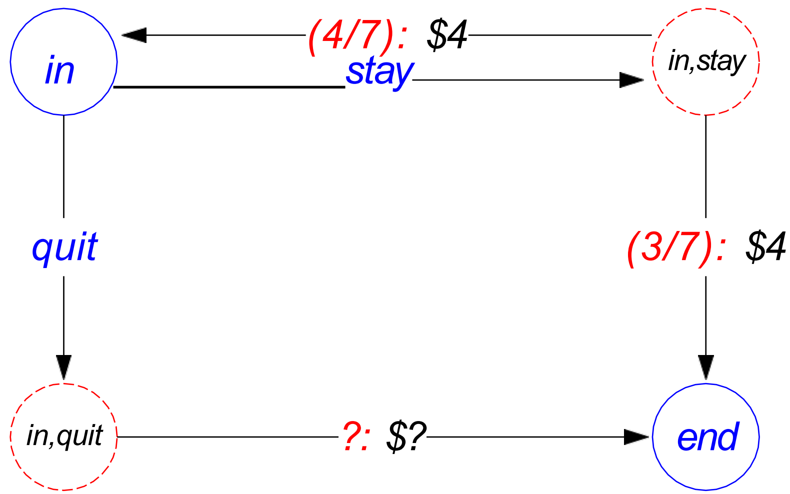


Key idea: exploration

To do reinforcement learning, need to explore the state space.

- Different from classical ML, where data comes passively and learns good function.
- Key challenge in RL, need to figure out how to get the data.

Model-Based Learning



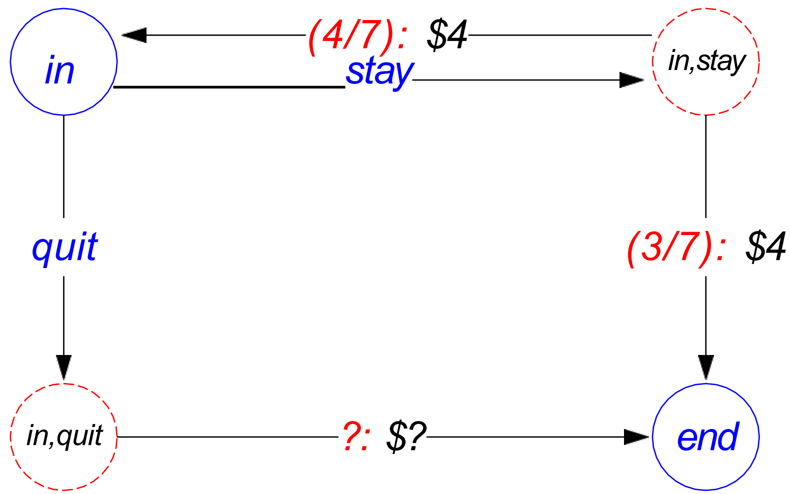
Problem: won't even see (s, a) if a ≠π(s) (a = quit)

Key idea: exploration

To do reinforcement learning, need to explore the state space.

Solution: need π to **explore** explicitly

Model-Based Learning



Data (following policy π(s) = quit):

[in; quit, R, end]

Transitioning?
Reward?

Model-Based Learning

Notes:

- Our policies have been deterministic. However, if we use such a policy to generate data, there are certain (s, a) pairs that we will never see and, therefore, never be able to estimate their Q-value and never know what the effect of those actions are.

- This problem points at the most important characteristic of reinforcement learning, which is the need for **exploration**.

- This distinguishes reinforcement learning from supervised learning, because now we actually have to act to get data, rather than just having data poured over us.

- if π is a non-deterministic policy that allows us to explore each state and action infinitely often (possibly over multiple episodes), then the estimates of the transitions and rewards will converge.

- Once we get an estimate for the T and R, we can simply plug them into our MDP and solve it using standard value or policy iteration to produce a policy.

From model-based to model-free

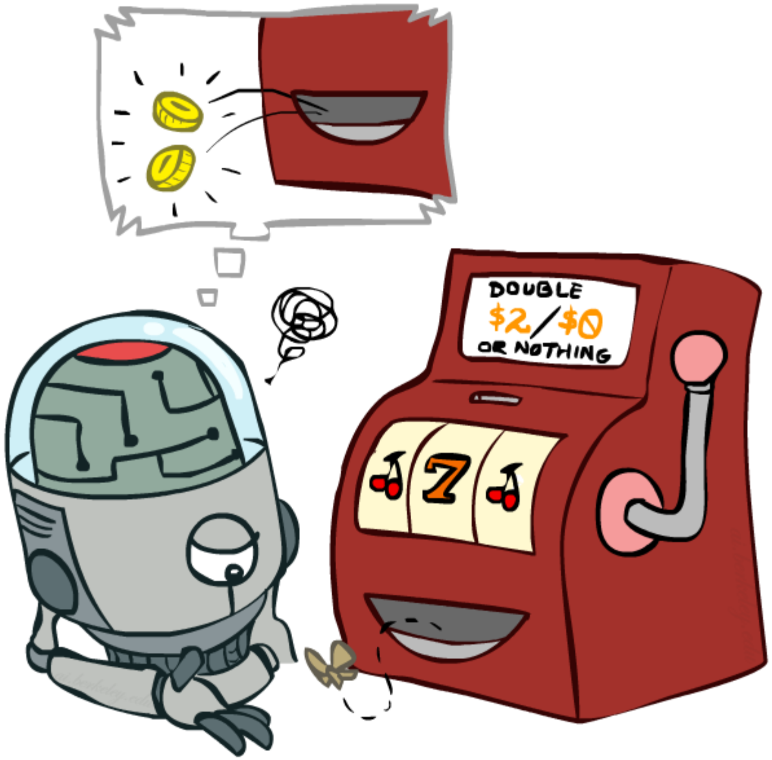
$$\hat{Q}_{\text{opt}}(s,a) = \sum_{s'} \hat{T}(s,a,s')[\widehat{\text{Reward}}(s,a,s') + \gamma \hat{V}_{\text{opt}}(s')]$$

All that matters for prediction is (estimate of) $Q_{opt}(s, a)$.

Key idea: model-free learning

Try to estimate $Q_{opt}(s, a)$ directly.

Model-Free Learning



Model-free Learning

Data (following policy π):

$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$

Recall:

$\boldsymbol{Q_\pi(s, a)}$ is expected utility starting at s, first taking action $\boldsymbol{a}$, and then following policy $\boldsymbol{\pi}$

Utility:

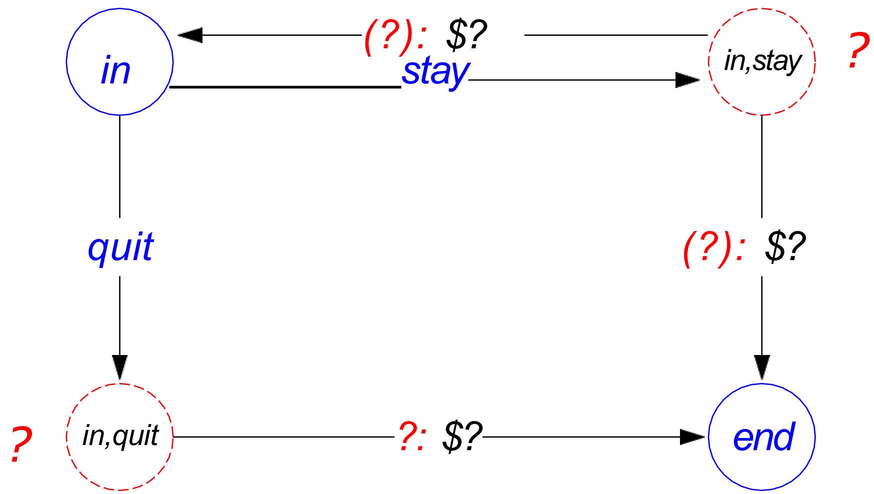
$u_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots$

Estimate:

$Q_\pi(s, a)$ = average of u_t where $s_{t-1} = s,\ a_t = a$

(and $s,\ a$ doesn't occur in $s_0, \cdots, s_{t-2}$)

Model-free Learning

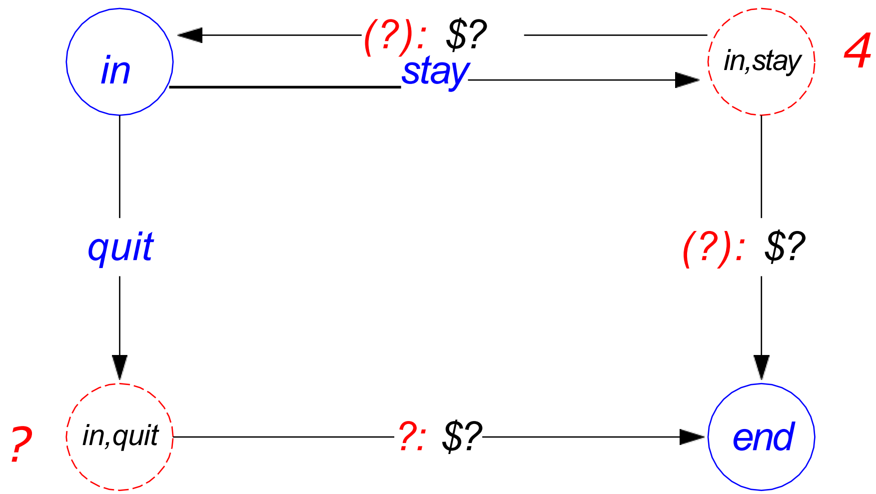


Data (following policy π(s) = stay):

[in; stay, 4, end]

Utility?

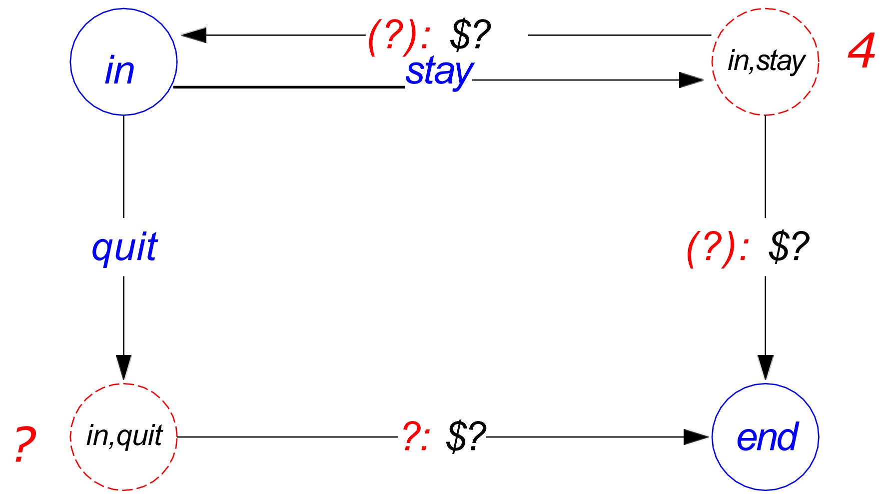
Model-free Learning



Data (following policy π(s) = stay):

[in; stay, 4, end]

Model-free Learning

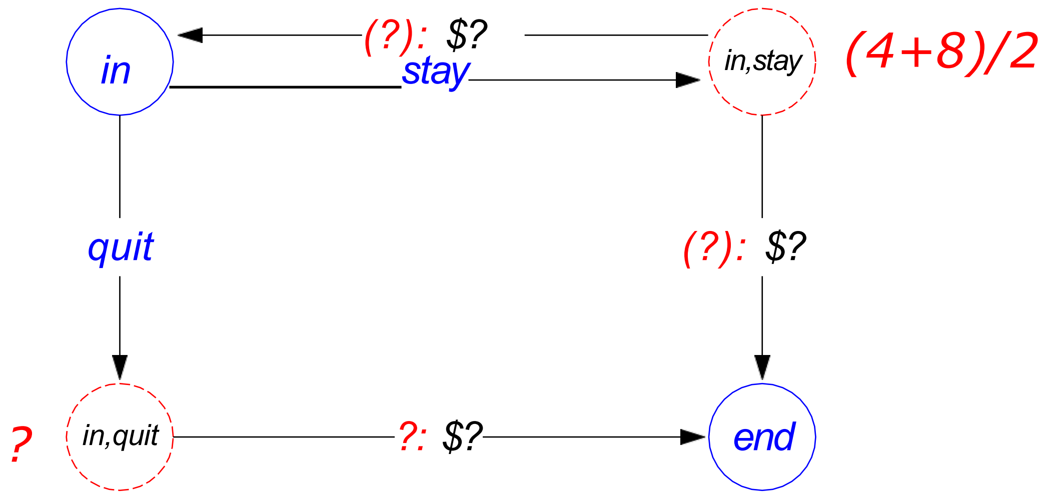


Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, end]

Utility?

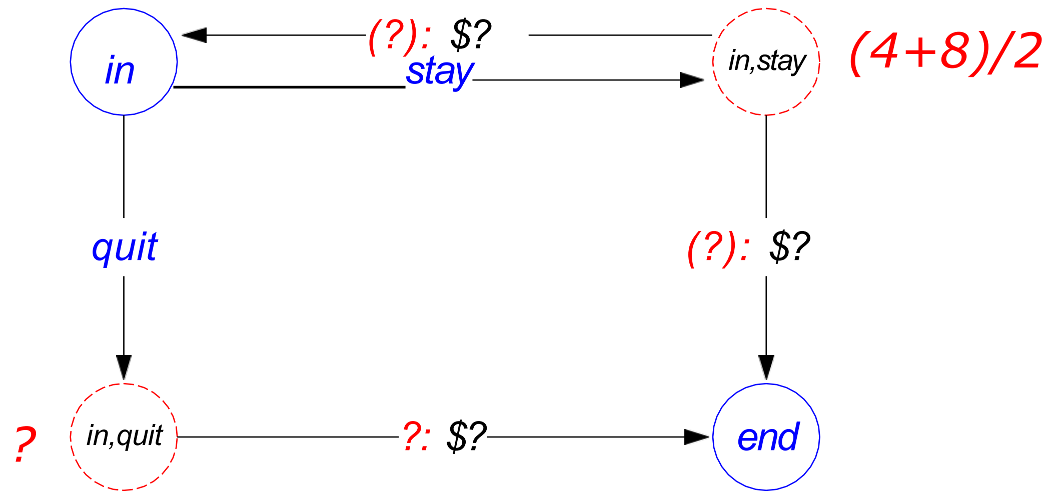
Model-free Learning



Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, end]

Model-free Learning

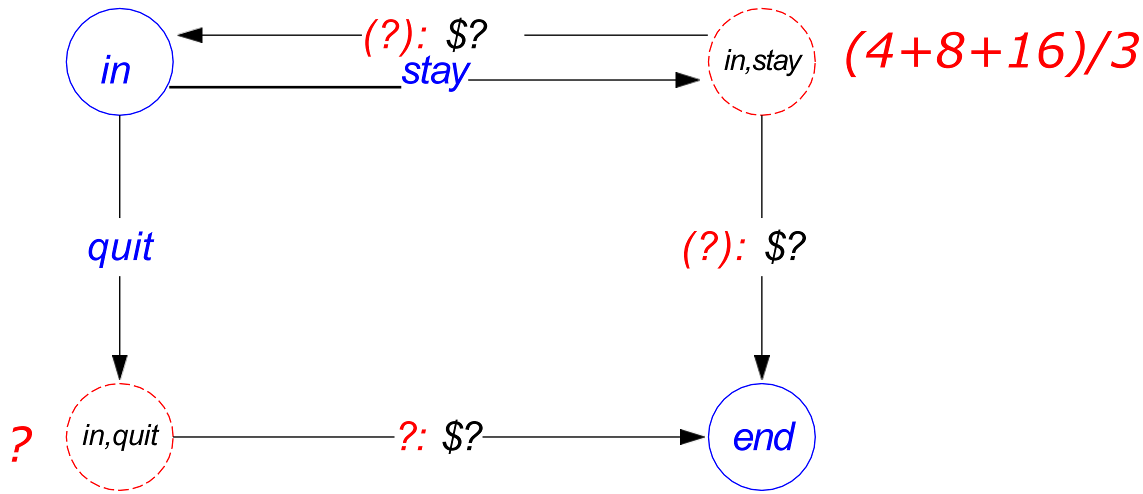


Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

Utility?

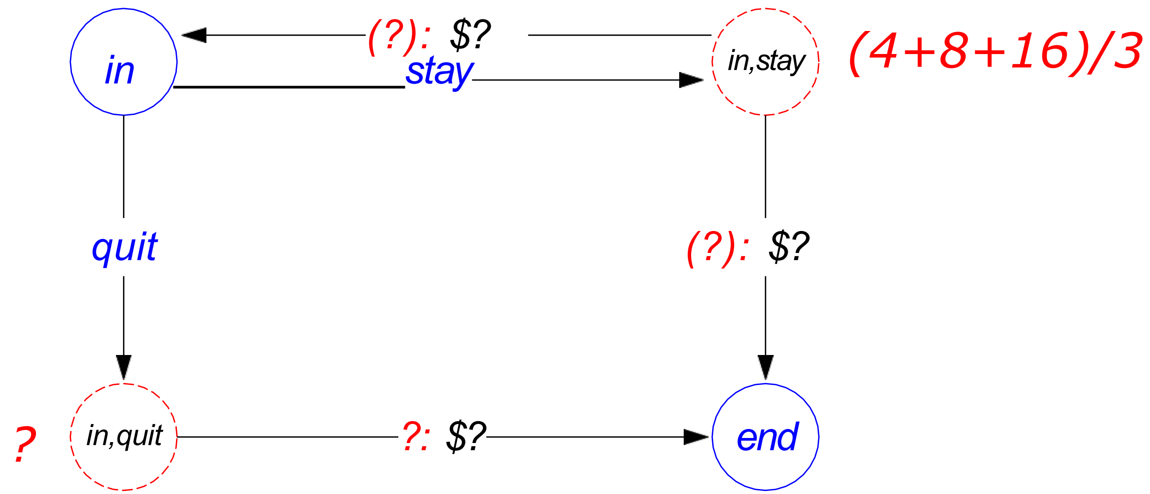
Model-free Learning



Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

Model-free Learning



Data (following policy π(s) = stay):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

Note: we are estimating Q_π now, not Q_{opt}

Definition: on-policy versus off-policy

On-policy: estimate the value of data-generating policy

Off-policy: estimate the value of another policy

Model based vs model-free

?

Model-free Learning (equivalences)

Data (following policy π):

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$$

Original formulation

$$\hat{Q}_\pi(s,a) = \text{average of } u_t \text{ where } s_{t-1} = s, a_t = a$$

Equivalent formulation (convex combination)

On each (s, a, u):

$$\eta = \frac{1}{1+(\#\text{ updates to }(s,a))}$$

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

Model-free Learning (equivalences)

Equivalent formulation (convex combination)

On each (s, a, u):

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

Equivalent formulation (stochastic gradient)

On each (s, a, u):

$$\hat{Q}_\pi(s,a) \leftarrow \hat{Q}_\pi(s,a) - \eta[\underbrace{\hat{Q}_\pi(s,a)}_{\text{prediction}} - \underbrace{u}_{\text{target}}]$$

Implied objective: least squares regression

$$(\hat{Q}_\pi(s,a) - u)^2$$

Model-free Learning (equivalences)

Data (following policy $\pi(s)$ = stay):

[in; stay, 4, end]	$u = 4$
[in; stay, 4, in; stay, 4, end]	$u = 8$
[in; stay, 4, in; stay, 4, in; stay, 4, end]	$u = 12$
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	$u = 16$

Algorithm: model-free Monte Carlo

On each (s, a, u):

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta \underbrace{u}_{\text{data}}$$

Using the reward + Q-value

Current estimate: $\hat{Q}_\pi(s, \text{stay}) = 11$

Data (following policy $\pi(s) =$ stay):

[in; stay, 4, end]	4+0
[in; stay, 4, in; stay, 4, end]	4+11
[in; stay, 4, in; stay, 4, in; stay, 4, end]	4+11
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	4+11

Algorithm: SARSA

On each (s, a, r, s', a'):

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta[\underbrace{r}_{\text{data}} + \gamma \underbrace{\hat{Q}_\pi(s',a')}_{\text{estimate}}]$$

Model-free versus SARSA

Key idea: bootstrapping

SARSA uses estimate $\hat{Q}_\pi(s, a)$ instead of just raw data u.

u

based on one path

large variance

wait until end to update

$r + \hat{Q}_\pi(s, a)$ based on estimate

small variance

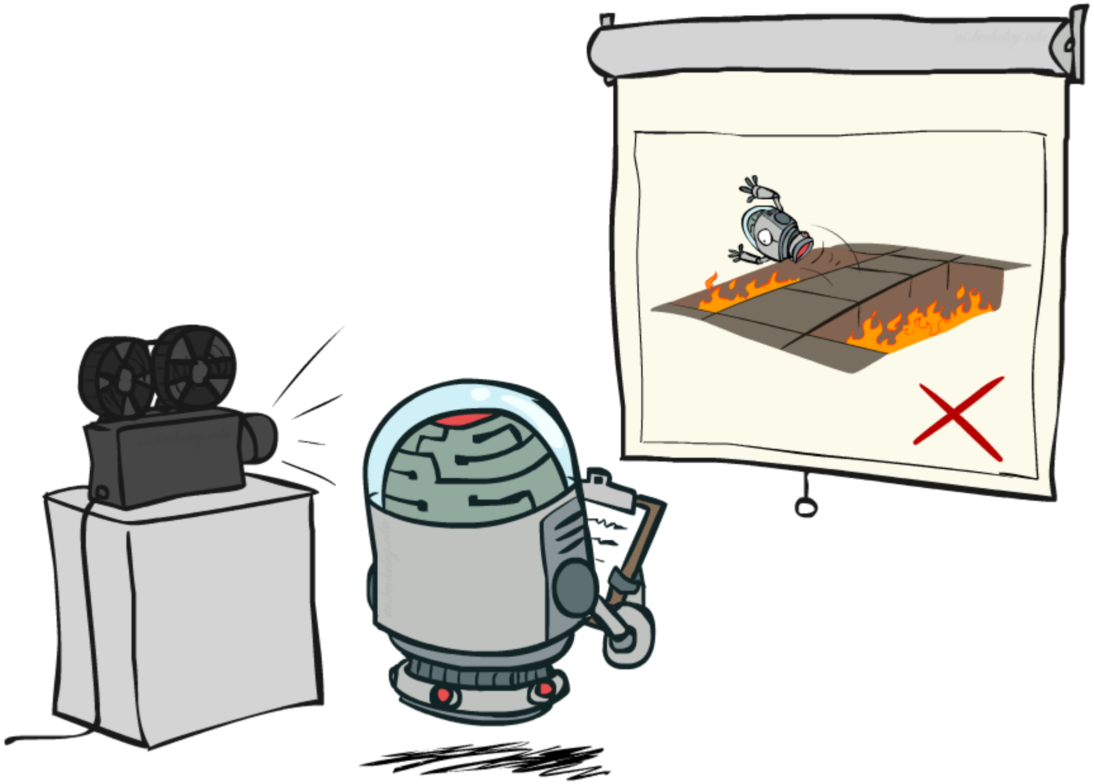
can update immediately

Question

Which of the following algorithms allows you to estimate $Q_{\text{opt}}(s, a)$ (select all that apply)?

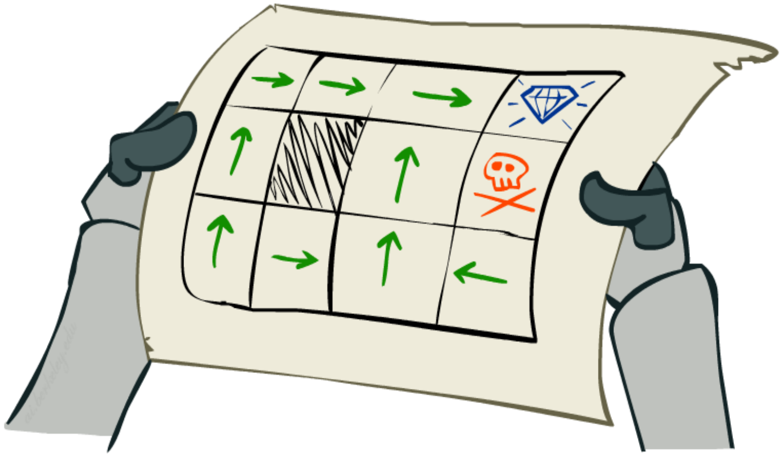
- model-based learning
- model-free learning
- SARSA

Passive Reinforcement Learning



Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$-- told what to do
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: evaluate how good an optimal policy is, learn the expected utility U for each s
- 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

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

Adaptive Dynamic Programming (ADP)

- Smarter method than Direct Utility Estimation.
- Estimating the utility of a state as a sum of reward for being in that state and the expected discounted reward of being in the next state.
- Converges fast but can become quite costly to compute for large state spaces.
- ADP is a model-based approach
- ADP adjusts the utility of s with all its successor states

Temporal Difference Learning (TD)

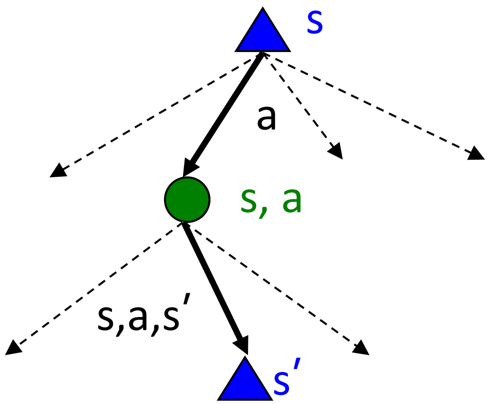
- model-free approach
- not require to learn the transition model
- update occurs between successive states and agent only updates states that are directly affected
- TD learning adjusts the utility of s with that of a single successor state s’

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation
- However, if we want to turn values into a (new) policy, we're sunk:

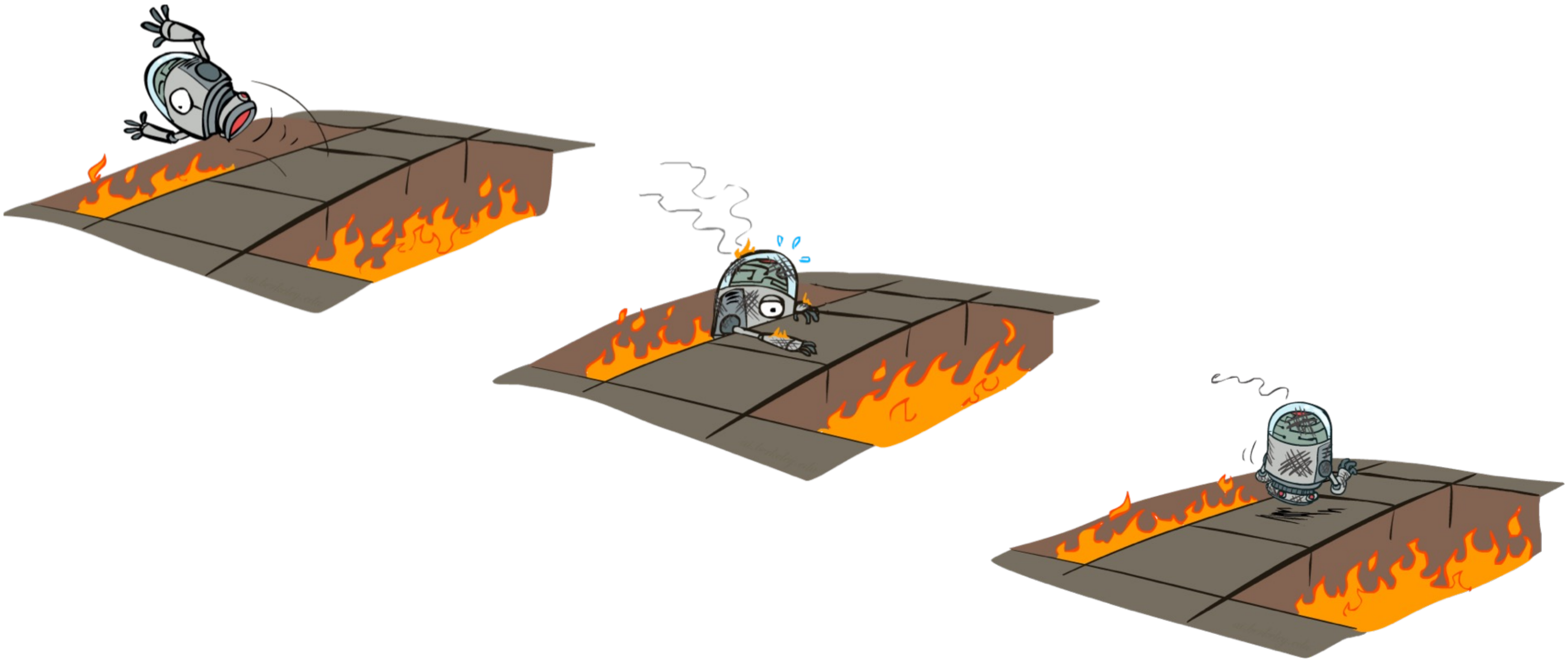
$$\pi(s) = \arg\max_a Q(s,a)$$

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



- Idea: learn Q-values, not values
- Makes action selection model-free too!

Active Reinforcement Learning



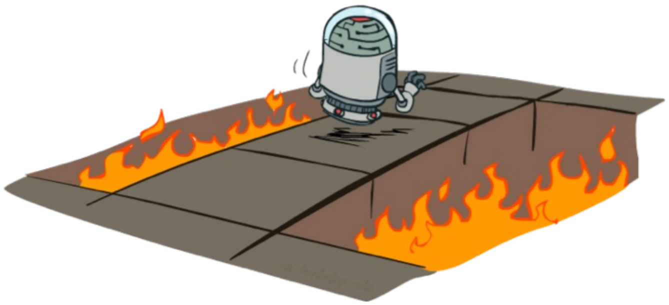
Active Reinforcement Learning

Problem: model-free and SARSA only estimate Q_π , but want Q_{opt} to act optimally

Output	MDP	reinforcement learning
Q_π	policy evaluation	model-free, SARSA
Q_{opt}	value iteration	**Q-learning**

Active Reinforcement Learning

- 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

MDP recurrence:

$$Q_{\text{opt}}(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')]$$

Algorithm: Q-learning [Watkins/Dayan, 1992]

On each (s,a,r,s'):

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\underbrace{\hat{Q}_{\text{opt}}(s,a)}_{\text{prediction}} + \eta\underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}$$

Recall: $\hat{V}_{\text{opt}}(s') = \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s',a')$

SARSA versus Q-learning

Algorithm: SARSA

On each (s, a, r, s', a'):

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta(r + \gamma\hat{Q}_\pi(s',a'))$$

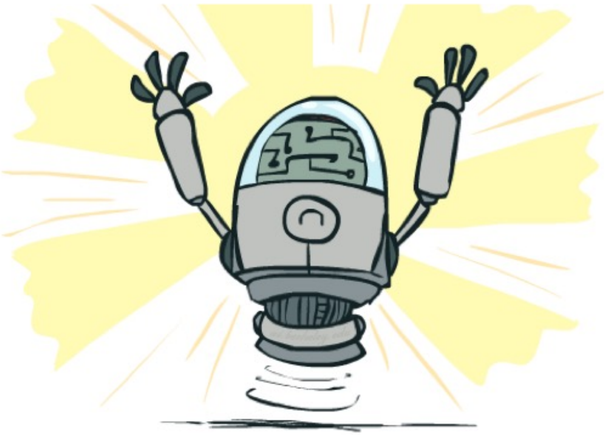
Algorithm: Q-learning [Watkins/Dayan, 1992]

On each (s, a, r, s'):

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\hat{Q}_{\text{opt}}(s,a) + \eta(r + \gamma \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s',a'))]$$

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 (!)



How to Explore?

Algorithm: reinforcement learning

For t = 1, 2, 3, . . .

Choose action $a_t = \pi_{act}(s_{t-1})$ (**how?**)

Receive reward r_t and observe new state s_t

Update parameters (**how?**)

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$$

Which **exploration policy** π_{act} to use?

Exploration/exploitation tradeoff

- No exploration, all exploitation

$$\text{Attempt 1: Set } \pi_{\text{act}}(s) = \arg\max_{a \in \text{Actions}(s)} \hat{Q}_{\text{opt}}(s, a)$$

- No exploitation, all exploration

$$\text{Attempt 2: Set } \pi_{\text{act}}(s) = \text{random from Actions}(s)$$

Exploration/exploitation tradeoff

Key idea: balance

Need to balance **exploration** and **exploitation**.

Examples from life: restaurants, routes, research

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 $1-\varepsilon$, 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 for large state space

ε-greedy

Algorithm: epsilon-greedy policy

$$\pi_{\text{act}}(s) = \begin{cases} \arg\max_{a \in \text{Actions}} \hat{Q}_{\text{opt}}(s,a) & \text{probability } 1-\epsilon, \\ \text{random from Actions}(s) & \text{probability } \epsilon. \end{cases}$$

Q-Learning

Stochastic gradient update:

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow \hat{Q}_{\text{opt}}(s,a) - \eta[\underbrace{\hat{Q}_{\text{opt}}(s,a)}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}]$$

This is **rote learning**: every $Q_{\text{opt}}(s, a)$ has a different value

Problem: doesn't generalize to unseen states/actions

Function approximation

Key idea: linear regression model

Define **features** $\phi(s, a)$ and **weights** $\mathbf{w}$:

$$\hat{Q}_{\text{opt}}(s, a; \mathbf{w}) = \mathbf{w} \cdot \phi(s, a)$$

Example: features for volcano crossing

$$\phi_1(s, a) = \mathbf{1}[a = \text{W}] \qquad \phi_7(s, a) = \mathbf{1}[s = (5, *)]$$

$$\phi_2(s, a) = \mathbf{1}[a = \text{E}] \qquad \phi_8(s, a) = \mathbf{1}[s = (*, 6)]$$

... ...

Function approximation

Algorithm: Q-learning with function approximation

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}]\phi(s, a)$$

Implied objective function:

$$(\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}})^2$$

Covering the unknown

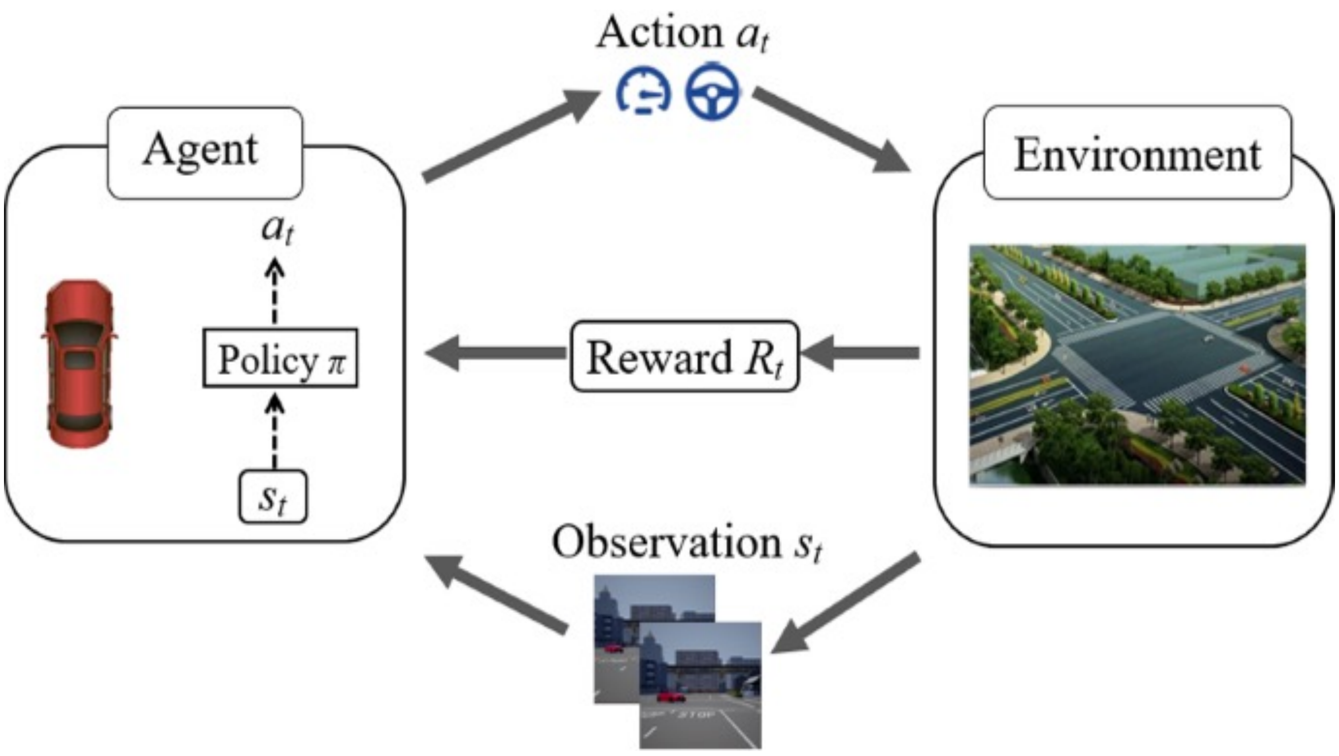
Epsilon-greedy: balance the exploration/exploitation tradeoff

Function approximation: can generalize to unseenstates

Summary so far

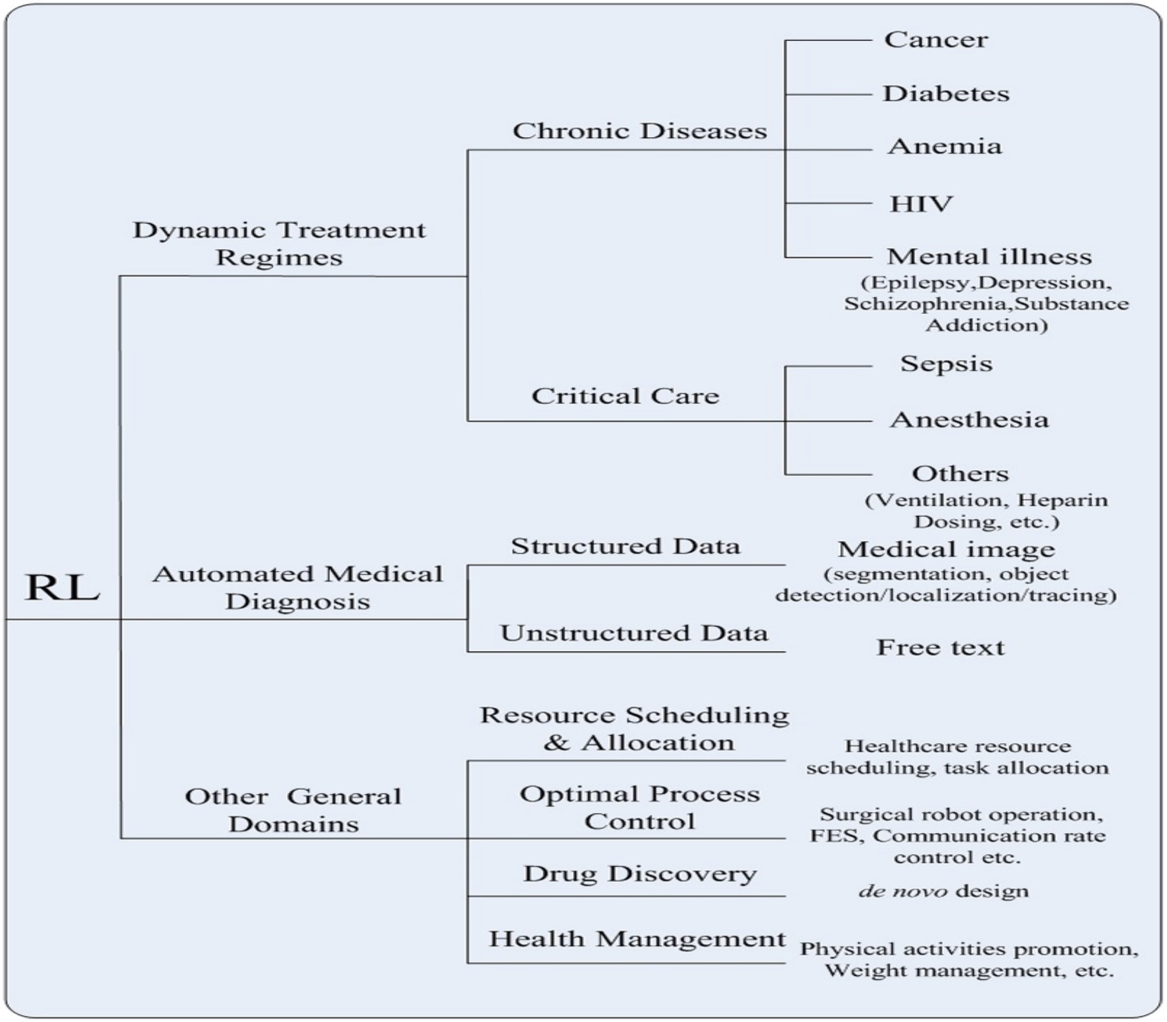
- Online setting: learn and take action in the real world!
- Exploration/exploitation tradeoff
- Monte Carlo: estimate transitions, rewards, Q-values from data
- Bootstrapping: update towards target that depends on estimate rather than just raw data

Applications- Autonomous cars



https://link.springer.com/article/10.1007/s42154-020-00113-1

Applications- Healthcare



Reinforcement Learning in Healthcare: A Survey

Applications

Autonomous helicopters: control helicopter to do maneu-vers in the air

Backgammon: TD-Gammon plays 1-2 million games against itself, human-level performance

Elevator scheduling; send which elevators to which floors to maximize throughput of building

Managing datacenters; actions: bring up and shut down machine to minimize time/cost

Deep reinforcement learning

- Policy gradient: train a policy $\pi(a \mid s)$ (say, a neural network) to directly maximize expected reward

- Google DeepMind's AlphaGo (2016), AlphaZero (2017)

- Andrej Karpathy's blog post

http://karpathy.github.io/2016/05/31/rl

https://www.youtube.com/watch?v=SUbqykXVx0A