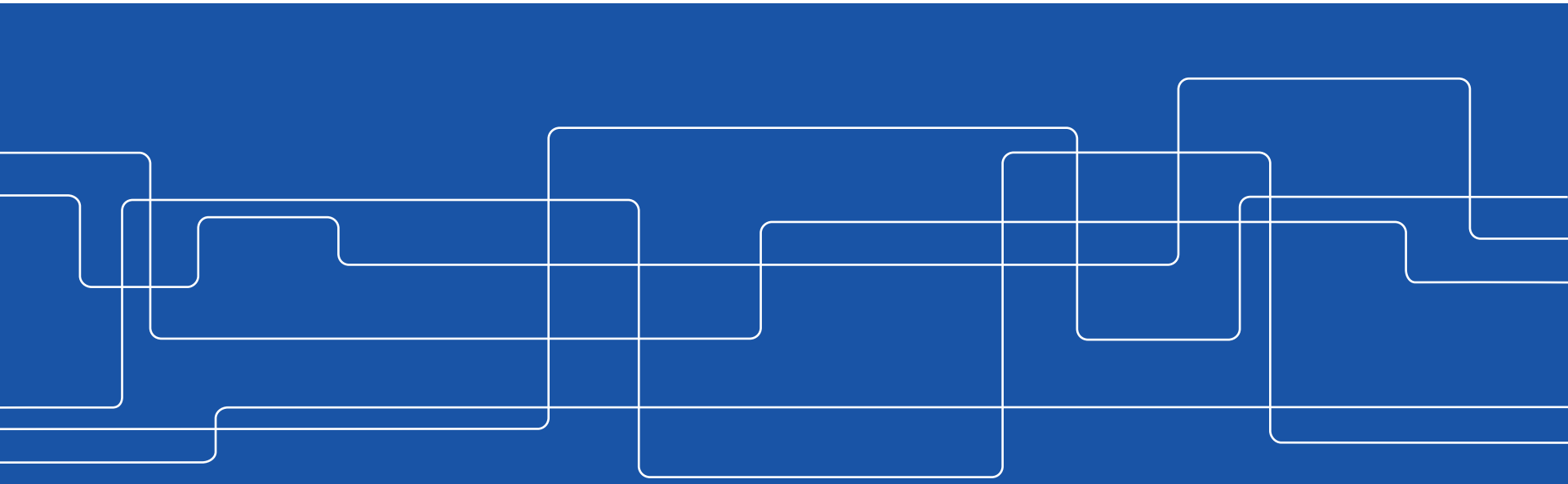

KTH ROYAL INSTITUTE
OF TECHNOLOGY

# DD2380 Artificial Intelligence

## Reinforcement Learning

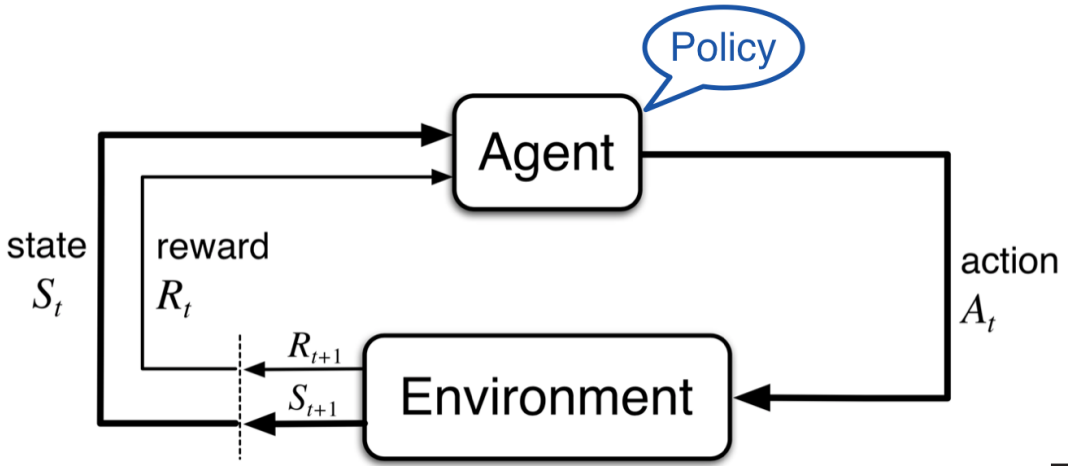
Iolanda Leite



# What is Reinforcement Learning?

*Rewards provide a positive reinforcement to our actions*

# What is Reinforcement Learning?

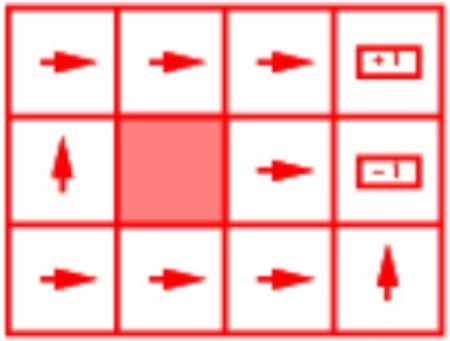



- Goal-directed **learning** from **interaction**
- Feedback in the form of **rewards**
- Agent must (learn to) act so as to **maximize expected rewards**
- Learning is based on observed samples of outcomes

# Characteristics of Reinforcement Learning

- Unlike other forms of supervised learning
  - No supervisor showing the best action to take, only reward signals
  - Actions affect observations – agent decides based on its own past observations

- Sequential decisions – time matters!
- Feedback is often delayed



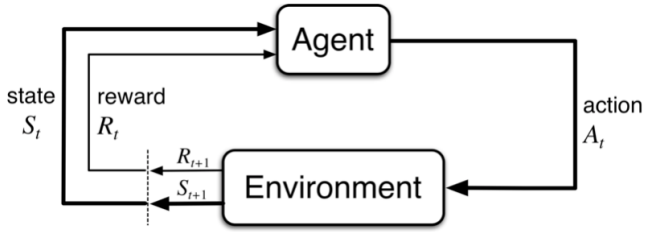
# RL Applications

- Automated vehicle control (e.g drones)
- Game playing
  - Playing Atari games, Tetris, etc.
- Medical treatment
  - Planning a sequence of treatments based on the effect of past treatments
- Chat bots (e.g. Siri, Alexa, …)
  - Learning the right thing to say at the right time

# MDP Formulation

Mathematical model: Markov Decision Process $(S, A, T, R, \gamma)$

- State space $S$
- Action space $A$
- Environment Transition model $T$
- Reward function $R$
- Discount factor $\gamma$




$$T(s, a, s') : p(s_{t+1} = s' \,|\, s_t = s, a_t = a)$$

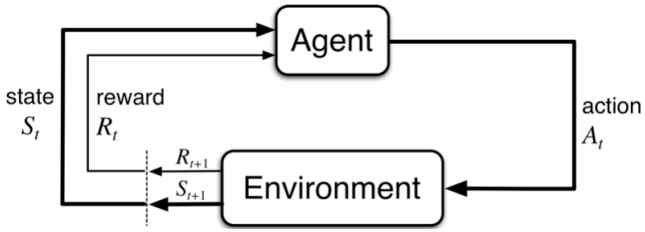
$R(s, a, s')$ : immediate reward for transitioning from state $s$ to state $s'$ due to action $a$

Frequently reward depends only on the state, so we usually write $R(s)$

From lecture: "Making Decisions Under Uncertainty"

# RL Formulation

- Still assume a Markov Decision Process (MDP):
  - State space $S$
  - Action space $A$
  - Environment Transition model $T$
  - Reward function $R$




- Still looking for a policy $\pi(s)$

- In RL, new twist: **don't know T or R**, so:
  - We don't know which states are good or what the actions do
  - Agent must actually try actions and states to learn

# RL Formulation

**Goal**: find an optimal policy that maximizes expected rewards

Recall the principle of maximum expected utility:

$$\pi^*(s) = \arg\max_a \sum_{s'} p(s' \mid s, a) U(s')$$

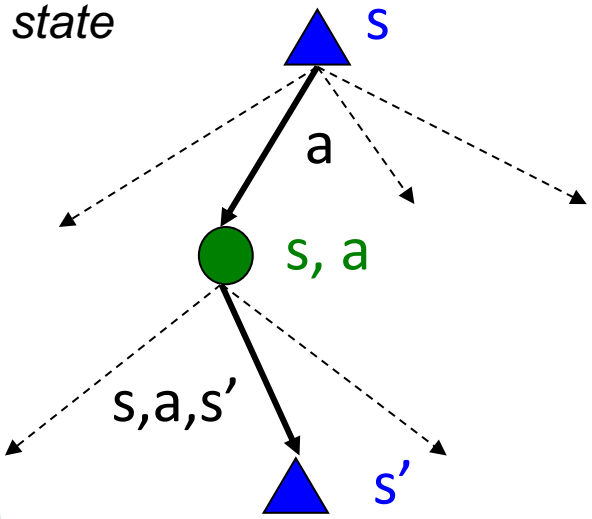
Where U(s) satisfies Bellman optimality equation:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} p(s' \mid s, a) U(s')$$

# Bellman equation

*"The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state"*

$$U(s) = R(s) + \gamma \max_a \sum_{s'} p(s'|s,a)U(s')$$

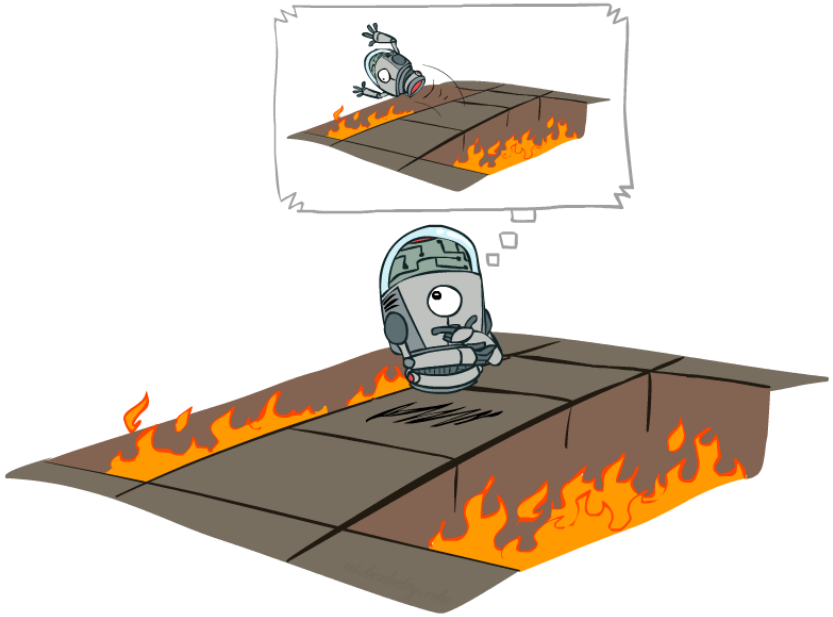



immediate reward

discounted expected utility of the next state, assuming optimal action

$$p(s'|s,a) = T(s,a,s')$$

# Offline (MDPs) VS. Online (RL)



Offline Solution:
Known T and R

Online Learning:
need to estimate T and R

**Model-based Learning**

"empirical MDP"
Aprox. T and R from data

Example algorithms

- Value iteration
- Policy iteration

**Model-free Learning**

**Passive RL**

- Temporal Difference Learning

**Active RL**

- Q Learning
- UCB

# Model-Based Learning

**Key intuition:**

- Learn an approximate model based on experiences
- Solve for values as if the learned model were correct
- After learning, the agent can make predictions about **T** and **R** *before* taking action

**Step 1**: Learn empirical MDP model

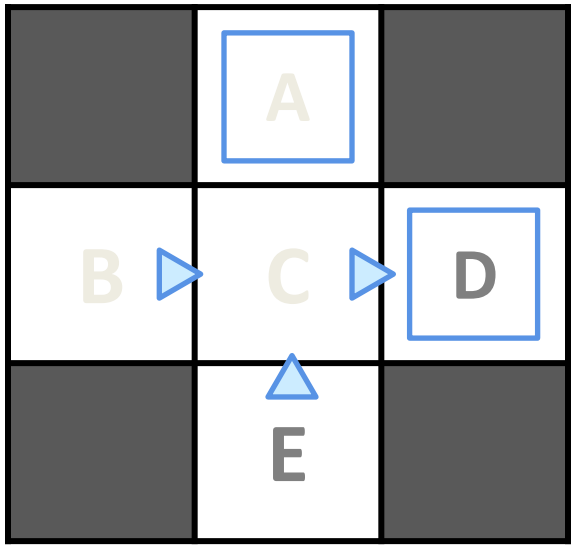
- Count outcomes s' for each s, a $\hat{T}(s, a, s')$
- Normalize to give an estimate of $\hat{T}$
- Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')

**Step 2**: Solve the learned MDP

- For example, use value iteration, as before

# Model-Based Learning: Example

Input Policy $\pi$




*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

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

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
...

# Model-Based Learning

**Advantage:**

- Make good use of data

**Disadvantage:**

- Requires building the actual MDP model
- Intractable if state space is too large

# Model-Free Learning

- Remember: we don't know the transition model **T** and reward distribution **R**

**Key intuition:** *Learn while optimizing policy*

- Model is unknown but agent observes *samples*
- Adjust "model" as you observe more samples
- Learn optimal policy by sampling the environment (without the need to create a model beforehand)


Model-based Learning

Model-free Learning

"empirical MDP"
Aprox. T and R from data

Passive RL

Active RL

Example algorithms

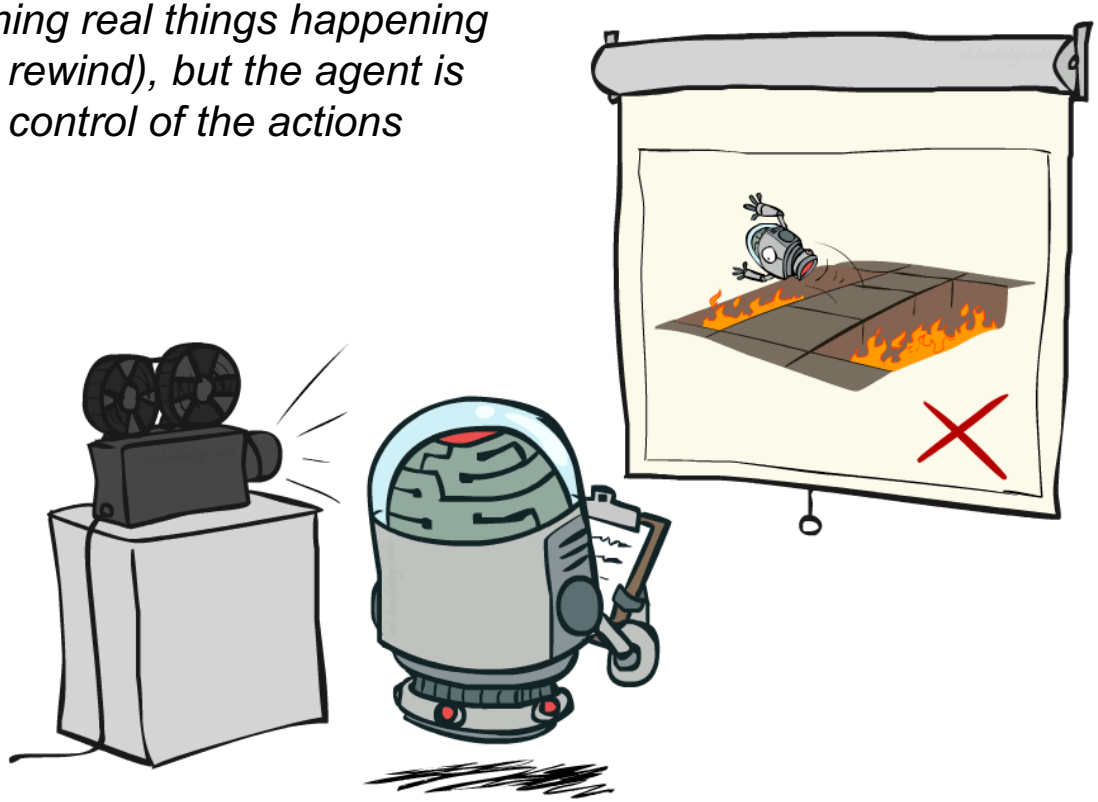
- Value iteration
- Policy iteration

- Temporal Difference Learning

- Q Learning
- UCB

# Passive Reinforcement Learning

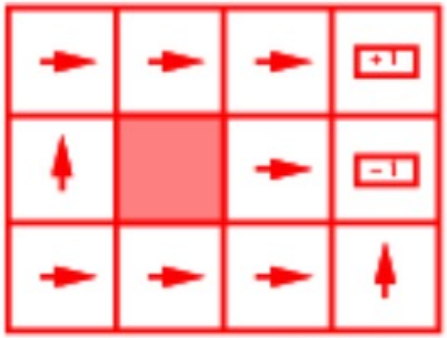
*Watching real things happening (can't rewind), but the agent is not in control of the actions*



# Passive Reinforcement Learning

Simplified task: policy evaluation

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



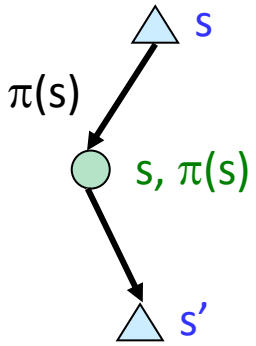

In this case:

- No choice about what actions to take
- Just execute the policy and learn from experience
- Learned values depend on the policy
  (if $\pi(s)$ is "bad", the values won't be very useful)

# Temporal Difference (TD) Learning

**Key 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): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

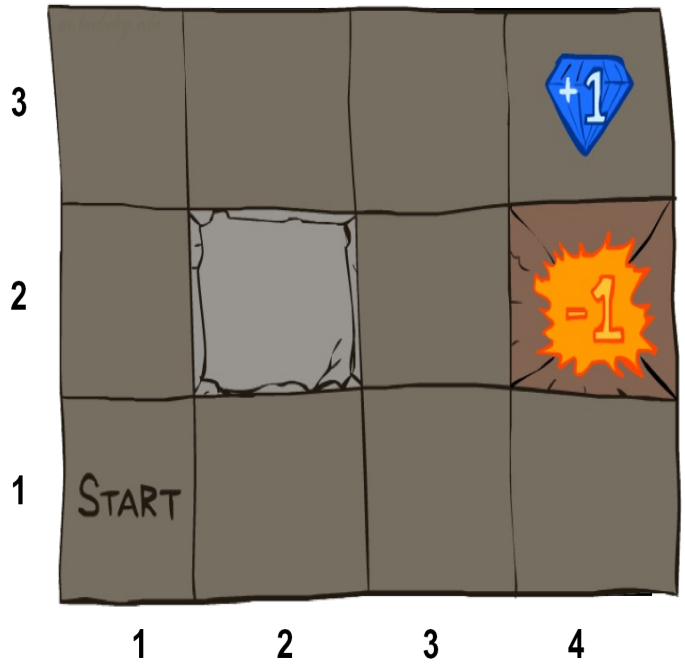
Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

what actually did happen (in one experience)

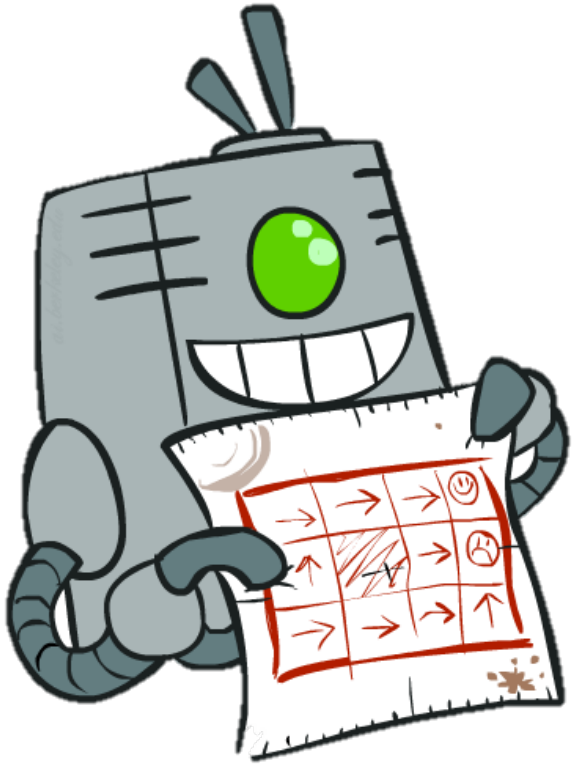
what we thought would happen

# TD Learning Example

You are in an environment (without knowing T and R)




You are given a policy $\pi$



# TD Example

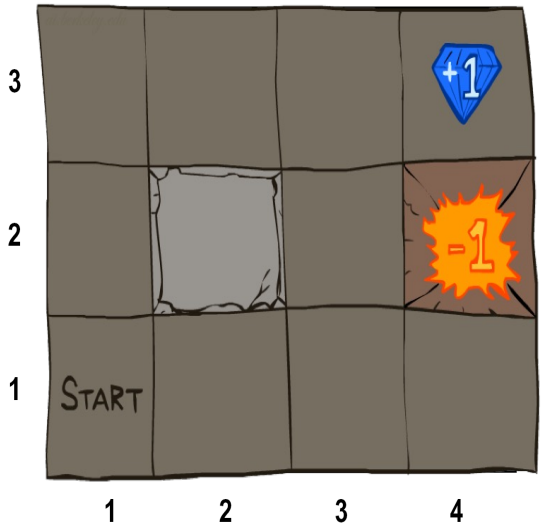
$$\alpha = 0.1, \gamma = 1$$

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha(r + \gamma V(s'))$$

Initialize $V(s)$ to be $0$

| State | $V(s)$ |
|---|---|
| (1,1) | 0 |
| (1,2) | 0 |
| (1,3) | 0 |
| (4,1) | 0 |
| (2,1) | 0 |
| (2,3) | 0 |
| (3,1) | 0 |
| (3,2) | 0 |
| (3,3) | 0 |

Environment




Policy

# TD Example

$$\alpha = 0.1, \gamma = 1$$

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha(r + \gamma V(s'))$$

Start at $(1,1)$

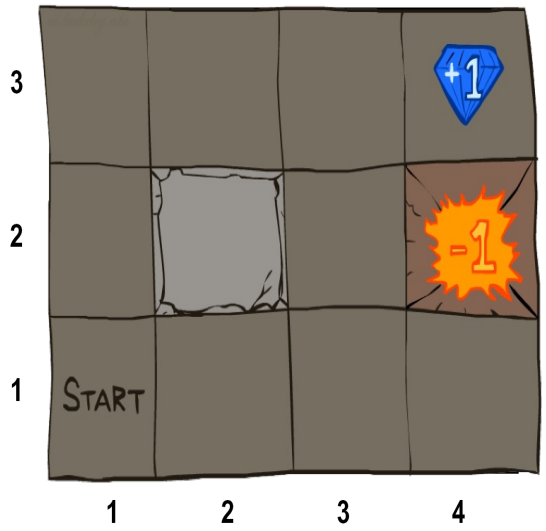
$s = (1,1)$ action=tright

Reward$= -0.01$; End up at $s' = (2,1)$

$V((1,1)) = 0.9 * 0 + 0.1 * (-0.01 + 0) = -0.001$

| State | V($s$) |
|---|---|
| (1,1) | 0 |
| (1,2) | 0 |
| (1,3) | 0 |
| (4,1) | 0 |
| (2,1) | 0 |
| (2,3) | 0 |
| (3,1) | 0 |
| (3,2) | 0 |
| (3,3) | 0 |

Environment




Policy

# TD Example

$$\alpha = 0.1, \gamma = 1$$

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha(r + \gamma V(s'))$$

Start at $(1,1)$

$s = (1,1)$ action=tright

Reward$= -0.01$; End up at $s' = (2,1)$

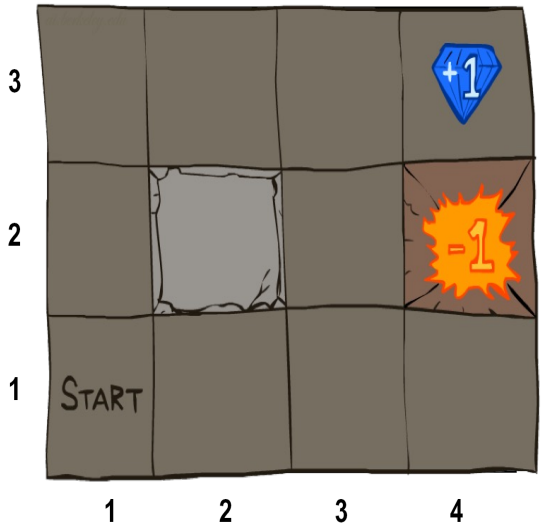
$s = (2,1)$ action=tright

Reward$= -0.01$; End up at $s' = (2,1)$

$$V((2,1)) = 0.9 * 0 + 0.1 * (-0.01 + 0) = -0.001$$

| State | V($s$) |
|---|---|
| (1,1) | -0.001 |
| (1,2) | 0 |
| (1,3) | 0 |
| (4,1) | 0 |
| (2,1) | 0 |
| (2,3) | 0 |
| (3,1) | 0 |
| (3,2) | 0 |
| (3,3) | 0 |

Environment




Policy

# TD Example

$$\alpha = 0.1, \gamma = 1$$

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha(r + \gamma V(s'))$$

Start at $(1,1)$

$s = (1,1)$ action=tright

Reward$= -0.01$; End up at $s' = (2,1)$

$s = (2,1)$ action=tright

Reward$= -0.01$; End up at $s' = (2,1)$

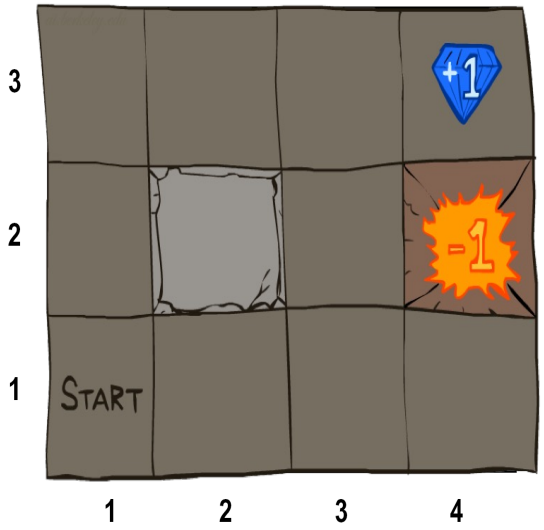
$s = (2,1)$ action=tright

Reward$= -0.01$; End up at $s' = (3,1)$

$$V((2,1)) = 0.9 * -0.001 + 0.1 * (-0.01 + 0) = -0.0019$$

| State | V($s$) |
|---|---|
| (1,1) | -0.001 |
| (1,2) | 0 |
| (1,3) | 0 |
| (4,1) | 0 |
| (2,1) | -0.001 |
| (2,3) | 0 |
| (3,1) | 0 |
| (3,2) | 0 |
| (3,3) | 0 |

Environment




Policy

# TD Example

$$\alpha = 0.1, \gamma = 1$$

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha(r + \gamma V(s'))$$

Start at $(1,1)$

$s = (1,1)$ action=tright

Reward$= -0.01$; End up at $s' = (2,1)$

$s = (2,1)$ action=tright

Reward$= -0.01$; End up at $s' = (2,1)$

$s = (2,1)$ action=tright

Reward$= -0.01$; End up at $s' = (3,1)$

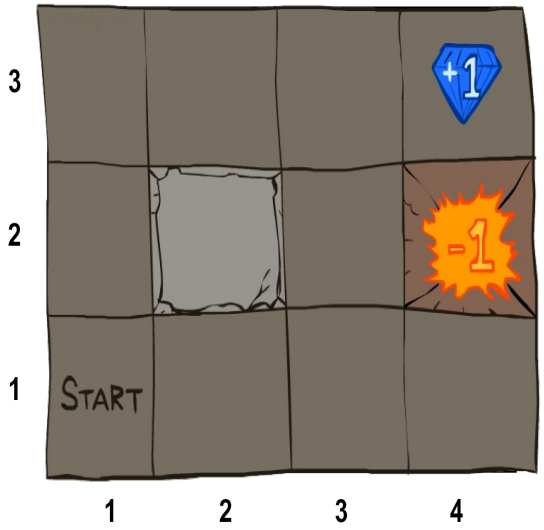
$s = (3,1)$ action=tright

Reward$= -0.01$; End up at $s' = (4,1)$

$V((3,1)) = 0.9 * 0 + 0.1 * (-0.01 + 0) = -0.001$

| State | V($s$) |
|---|---|
| (1,1) | -0.001 |
| (1,2) | 0 |
| (1,3) | 0 |
| (4,1) | 0 |
| (2,1) | -0.0019 |
| (2,3) | 0 |
| (3,1) | 0 |
| (3,2) | 0 |
| (3,3) | 0 |

Environment




Policy

Model-based Learning

Model-free Learning

Passive RL

Active RL

"empirical MDP"
Aprox. T and R from data

Example algorithms

- Value iteration
- Policy iteration

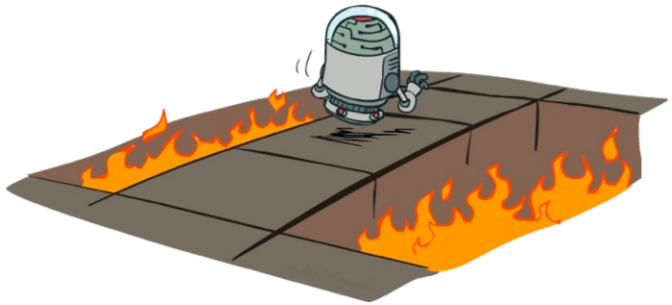
- Temporal Difference Learning

- Q Learning
- UCB

# Active Reinforcement Learning

Full reinforcement learning: optimal policies

- 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!
- Unlike passive RL, in active RL the agent needs to learn the outcome probabilities for all actions, not just the model for the fixed policy
- Fundamental tradeoff: exploration vs. exploitation

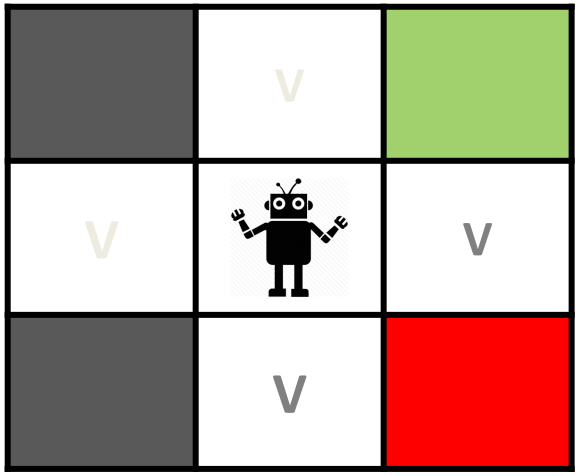
# Active Reinforcement Learning (cont.)

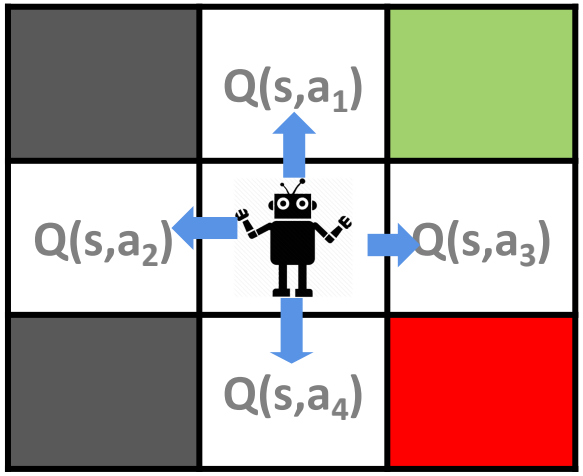
- Choose actions (randomly or in some other way)
- Estimate $T$ and $R$ from sample trials (average counts)
- Use estimated $T$ and $R$ to compute estimate of optimal values and optimal policy

- Will the computed values and policy converge to the true optimal values and policy in the limit of infinite data?
  - Sufficient condition: If all states are reachable from any other state
  - Be able to visit each state and take each action as many times as we want

# Q-Learning Intuition

Optimal value functions (used e.g. in TD learning) tell us how "good" a **state** is, but does not explicitly capture how good the **actions** are

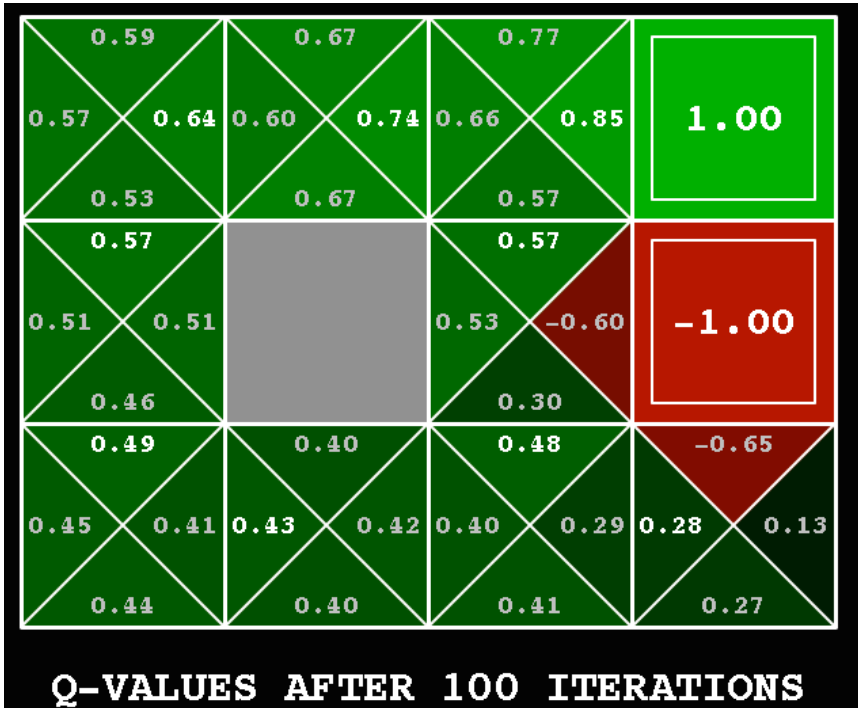



Instead of evaluating the *value* of a state (V), evaluate the *actions* Q(s,a)

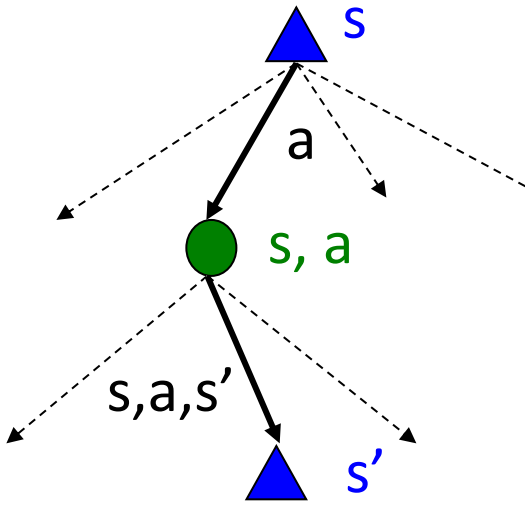
# Q-Learning Intuition

Optimal value functions (used e.g. in TD learning) tell us how "good" a **state** is, but does not explicitly capture how good the **actions** are


Discount = 0.9, Living reward = 0

# Q Function




So we define

Q(s,a): the value of taking action *a* in state *s*

Optimal $Q$ function : $Q_*(s,a) := \max_\pi Q_\pi(s,a)$

---

If we know $Q_*(s,a)$ we can more easily obtain an optimal policy:

$$\pi^*(s) = \arg\max_a Q_*(s,a)$$

How do we learn $Q_*(s,a)$?

# Q-Learning Updates

Q-value updates to each Q-state

- Receive a sample transition (s,a,r,s')
- This sample suggests

$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

- But we want to average over results from (s,a)
- So keep a running average

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

# Q-Learning Algorithm

Initialize $Q(s,a) = 0 \; \forall s, a$

Loop:

Set $\pi_{\varepsilon\text{-}greedy} = \begin{cases} \arg\max_a Q(s,a) \text{ with prob } (1-\varepsilon) \\ \text{a random action with prob } \varepsilon \end{cases}$

Take action $a = \pi_{\varepsilon\text{-}greedy}(s)$ observe reward $r$ & next state $s'$

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

Let's rewrite: $Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')] - \alpha Q(s,a)$

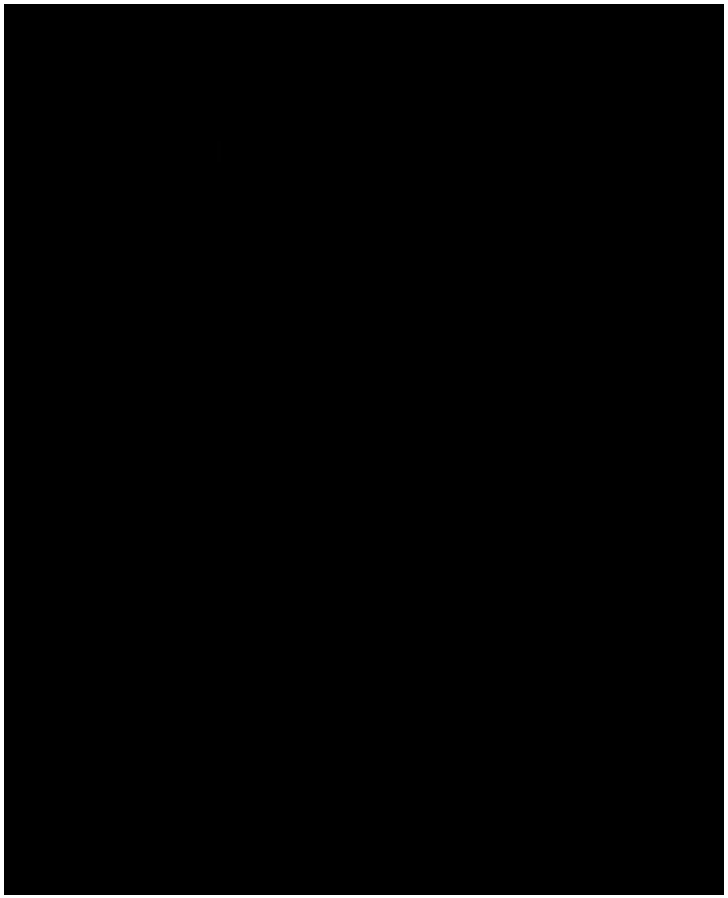
$= (1-\alpha) Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a')]$

"old" estimate of Q

"new" estimate of Q (immediate reward plus the best Q value from the next state)

similar to learning rate

# Q-Learning: Deep Mind ATARI example



https://www.youtube.com/watch?v=V1eYniJ0Rnk&t=27s

# Q-Learning Properties

Q-learning converges to an optimal policy – even if you're acting sub optimally!

This is called off-policy learning

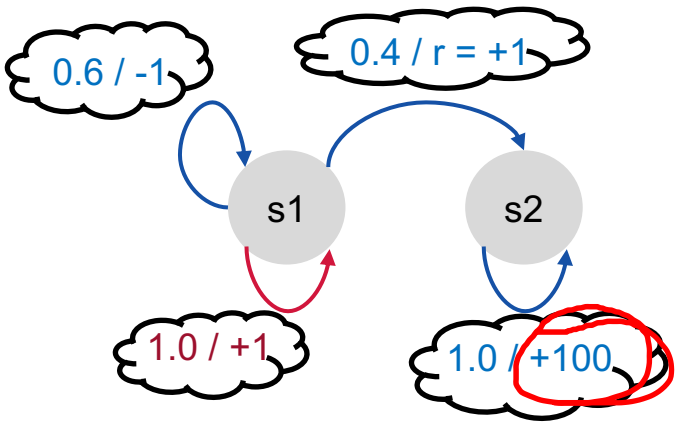
Caveats:

- You have to eventually make the learning rate $\alpha$ small enough
  … but not decrease it too quickly
- In the limit, it doesn't matter how you select actions (!)
- You have to explore enough

# Q-Learning: the need for Exploration

Two actions, a1 and a2

probability / reward




Initialize $Q(s,a) = 0 \; \forall s, a$

Loop:

Take action $\arg\max_a Q(s,a)$ observe reward $r$ & next state $s'$

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

Assume **α** = 1

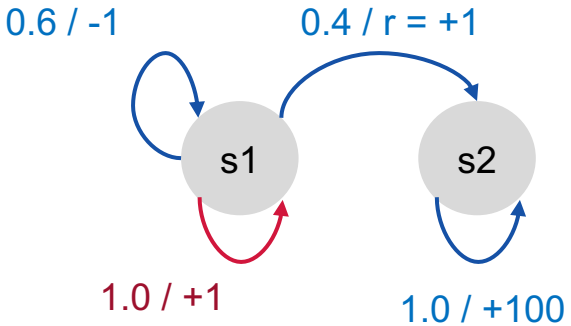
How would Q learning behave if the first update were Q(s1, a1), r(s1, a1) = -1, and actions were selected based on $\arg\max_a Q(s,a)$ ?

# Q-Learning: the need for Exploration

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

Two actions, a1 and a2

probability / reward




Q(s1, a1) = 0
Q(s1, a2) = 0
Q(s2, a1) = 0 ← will never change

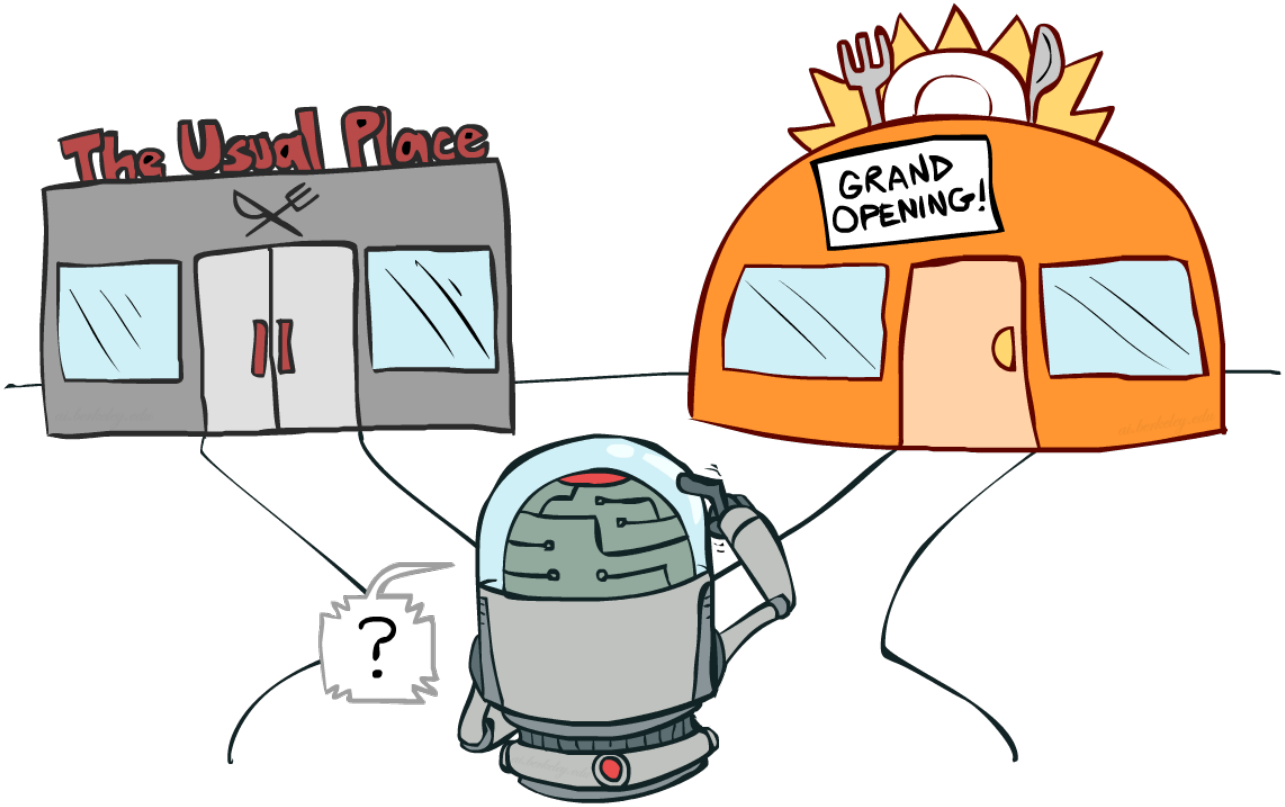
Q(s1, a1) = -1

Q(s1, a2) > Q(s1,a1), so
Q(s1, a2) = 1

Q(s1, a2) > Q(s1,a1) still, so
Q(s1, a2) = 1 + ϒ..
…
Keep selecting Q(s1, a2)

How would Q learning behave if the first update were Q(s1, a1) and r(s1, a1) = -1, actions were selected based on $\arg\max_a Q(s,a)$ ?

# Exploration vs. Exploitation



# 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

# ε-greedy in Q-Learning

Initialize $Q(s,a) = 0 \; \forall s, a$

Loop:

Set $\pi_{\varepsilon\text{-}greedy} = \begin{cases} \arg\max_a Q(s,a) \text{ with prob } (1-\varepsilon) \\ \text{a random action with prob } \varepsilon \end{cases}$

Take action $a = \pi_{\varepsilon\text{-}greedy}(s)$ observe reward $r$ & next state $s'$

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

# 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: *optimal* exploration policies

# 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 and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and (more sophisticated) exploration functions both end up optimal, but random exploration has higher regret

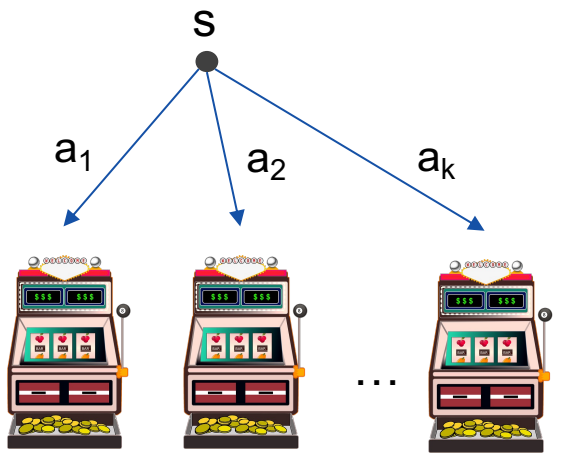
# Multi-Armed Bandit Problems

- Is there an *optional* exploration policy?

  Bandit problems are difficult to solve exactly to obtain an optimal exploration method, but it is possible to come up with reasonable solutions that eventually lead an agent to take the optimal behavior

- Optimally balance the exploration-exploitation tradeoff, minimizing regret

# Multi-Armed Bandit Problem Formulation




A gambler can play in *K* slot machines

Sampling action *a* is like pulling a slot machine arm with random payoff $r_n$

**Goal:** find an arm-pulling strategy that maximizes the expected total reward at time *n*

**Procedure:** at time step *n*, pick an arm $a_n$ based on what happened so far and receive reward $r_n$

# UCB Algorithm

1. Play each machine once
2. Loop: take action $a_n$ & update reward

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

where
Q(a) : average reward for trying action a so far
n(a) : number of pulls of arm a so far

**Exploitation:**
favors actions that looked good historically

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

Note: assuming rewards in [0,1]!

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

# Multi-Armed Bandit Applications (1/3)

- Bandit problems arise in many applications, whenever
  - We have a set of independent options with *unknown utilities*
  - There is a *cost* for sampling options or a limit on total samples
  - Want to find the *best option* or maximize utility of our samples

# Multi-Armed Bandit Applications (2/3)

**Clinical Trials**

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

**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 = maximize clicks (or find best add quickly)

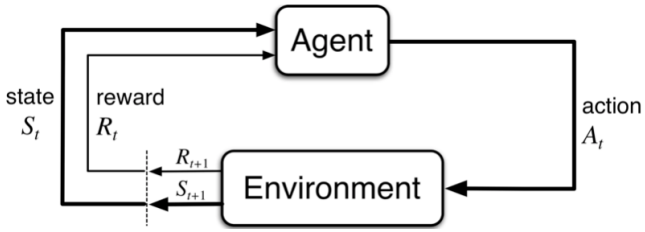
# Multi-Armed Bandit Applications (3/3)

**Adaptive Human-Robot Interaction**

- Arms = supportive strategies (e.g. suggest move, encouraging comment, …)
- Arm Pulls = displaying a supportive strategy to human
- Rewards = change in human affective state
- Objective = maximize cumulative reward = maximize positive affective states (or find the most effective supportive strategy for a particular human)

# RL Summary

- Still assume a Markov Decision Process (MDP):
  - State space $S$
  - Action space $A$
  - Environment Transition model $T$
  - Reward function $R$



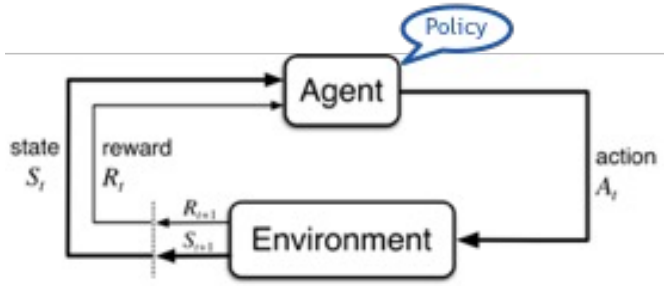

- Still looking for a policy $\pi(s)$

- In RL, new twist: **don't know T or R**, so:
  - We don't know which states are good or what the actions do
  - Must actually try actions and states to learn

# Reinforcement Learning: Conclusions

**Opportunities**:

- Learn flexible policies without restrictive assumptions
  (model-free learning, Q & value functions)
- Solve a variety of real-world problems “from scratch”
  (no hand-crafted features; a common framework for different domains)
- Resolve exploration / exploitation tradeoff
  (automatically determine how much and where to explore)



# Credits

Based partly on materials from:

- Rika Antonova, KTH
- Pieter Abbeel & Dan Klein, UC Berkeley
- Alan Fern, Oregon State
- Shipra Agrawal, Columbia University
- Fei Fang and Dave Touretzky, CMU


End of lecture

# Bellman equation

Bellman equation

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s')$$

Converges to unique optimal solution

Stop iterations when largest change in utility for any state is small enough

Can show that

$$||U_{i+1} - U_i|| < \epsilon \frac{1-\gamma}{\gamma} \Rightarrow ||U_{i+1} - U|| < \epsilon$$

# Value iteration

Key insight: Utility of a state is immediate reward plus discounted expected utility of next states

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

(assuming that we choose the optimal policy)

Idea: Iterate

- Calculate utility of each state
- Use utilities to select optimal decision in each state

# Algorithm: Value iteration

Initialize U(s) arbitrary for all s

Loop until policy has converged

loop over all states, s

loop over all actions, a

$$Q(s,a) := R(s) + \gamma \sum_{s'} T(s,a,s') * U(s')$$

end

$$U(s) := \max_a Q(s,a)$$

end

end

# Policy iteration

Alternative to value iteration

Choose an arbitrary policy

Loop until policy does not change any more

- Compute the value function, V(s) given the policy, known as “policy evaluation”

$$V(s) = \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \left[ R(s, \pi(s), s') + \gamma V(s') \right], \forall s \in \mathcal{S}$$

- Given this value function, improve the policy for each state

$$\pi(s) \leftarrow \arg\max_a \sum_{s' \in \mathcal{S}} T(s, a, s') \left[ R(s, a, s') + \gamma V(s') \right]$$