SAPIENZA UNIVERSITÀ DI ROMA

Department of Computer, Automation and Management Engineering "A. Ruberti"

Markov Decision Processes

Roberto Capobianco

Reinforcement Learning

Decision Theory

Problem in its simplest form:

- Choosing among actions based on *desirability*
- Desirability is evaluated according to *immediate* outcomes
- Nondeterministic partially observable environments
- $Result(a)$ is the outcome, represented as a random variable
  a represents the event of executing action a

$$P(Result(a) = s' | a, z)$$

Utility Function

Utility function:

- Agent's preferences are captured by an **utility function** *U(s)*
- *U(s)* is a number representing the desirability of a state

Expected utility:

- The expected utility of an action given the evidence is *EU(a|z)*
- Expected utility is the average of *U(s')*, weighted by *P(s'|a,z)*

$$EU(a|z) = \sum_{s'} P(Result(a) = s'|a, z)U(s')$$

Maximum Expected Utility

Principle of maximum expected utility:

A rational agent should choose the action that maximizes the agent's EU

- Agents aim at the highest performance score

Note: hard! Requires perception, learning, KR, inference, complete causal models, etc.

$$action = \operatorname{argmax}_a EU(a|z)$$

Outcomes as a Lottery

- Think of action outcomes as a lottery L (actions are tickets)
- Possible outcomes: $S_1 \ldots S_n$
- Outcomes occur with probabilities $p_1 \ldots p_n$

$$L = [p_1, S_1; p_2, S_2; \ldots p_n, S_n]$$

What are reasonable preference relations between states?

Axioms of Utility Theory

- **Orderability:** given 2 lotteries, agent must prefer one or rate them as equally preferable
  it cannot avoid deciding

- **Transitivity:** given 3 lotteries, if agent prefers *A* to *B*, and *B* to *C*, then it must prefer *A* to *C*

- **Continuity:** if *B* is between *A* and *C* in preference, a *p* exists for which the agent will be indifferent between getting *B* for sure and *A* with probability *p* and *C* with probability *p-1*

- **Substitutability:** if agent is indifferent between *A* and *B*, there is a more complex lottery $[p, A; (1-p), C]$ in which *A* can be substituted with *B*, and agent is indifferent among those

- **Monotonicity:** if 2 lotteries have same possible outcomes, *A* and *B*, if agent prefers *A* to *B*, then agent must prefer lottery with higher probability for *A*

- **Decomposability:** compound lotteries $[p, A; (1-p), [q, B; (1-q), C]]$ can be reduced to simpler ones $[p, A; (1-p)q, B; (1-p)(1-q), C]$ using probability laws
  consecutive lotteries can be compressed in a single one

Axioms of Utility Theory

SAPIENZA
UNIVERSITÀ DI ROMA

- **Orderability:** given 2 lotteries, agent must prefer one or rate them as equally preferable
  it cannot avoid deciding
- **Transitivity:** given 3 lotteries, if agent prefers *A* to *B*, and *B* to *C*, then it must prefer *A* to *C*
- **Continuity:** if *B* is between *A* and *C* in preference, a *p* exists for which the agent will be indifferent between getting *B* for sure and *A* with probability *p* and *C* with probability *p-1*
- **Substitutability:** if agent is indifferent between *A* and *B*, there is a more complex lottery $[p, A; (1-p), C]$ in which *A* can be substituted with *B*, and agent is indifferent among those
- **Monotonicity:** if 2 lotteries have same possible outcomes, *A* and *B*, if agent prefers *A* to *B*, then agent must prefer lottery with higher probability for *A*
- **Decomposability:** compound lotteries $[p, A; (1-p), [q, B; (1-q), C]]$ can be reduced to simpler ones $[p, A; (1-p)q, B; (1-p)(1-q), C]$ using probability laws
  consecutive lotteries can be compressed in a single one

Agents that violate such axioms exhibits irrational behavior in some situations

More on Utility Functions

- Utilities exist, but they are not necessarily unique

  e.g., agent's behavior would not change if it changed $U(s)$ with $U'(s) = aU(s)+b$

- Agent does not *explicitly* maximize utility in its deliberations

- An agent can have any kind of preferences without being irrational

  e.g., having a prime number of dollars, instead of a higher number

Utility of Money

- Suppose you won \$1M
- Either you can take it, or you can gamble on flipping a coin (heads → \$0, tails → \$2.5M)

- Most people would decline. Is that irrational?
- Expected Monetary Value: $\frac{1}{2}(\$0) + \frac{1}{2}(\$2.5\text{M}) = \$1.25\text{M}$ (more than \$1M)
  - Is it better to accept the gamble then?

- Assumptions: current wealth $\$k$, total wealth $\$n$, S_n state of possessing $\$n$

$$EU(Accept) = \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+2.5\text{M}})$$
$$EU(Decline) = U(S_{k+1\text{M}})$$

We need to assign utilities, which are not directly proportional to money

utility of first million is very high, that for an additional million is smaller

Utility of Money (continued)

- Assumptions: current wealth $\$k$, total wealth $\$n$, S_n state of possessing $\$n$

$$EU(Accept) = \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+2.5\text{M}})$$
$$EU(Decline) = U(S_{k+1\text{M}})$$

We need to assign utilities, which are not directly proportional to money

utility of first million is very high, that for an additional million is smaller

- Utility assignments: $U(S_k) = 5$, $U(S_{k+2.5\text{M}}) = 9$, $U(S_{k+1\text{M}}) = 8$ → rational action is decline
- Utility assignment of a millionaire (linear for few millions) → rational action is accept

Expected Utility and Disappointment

$$a^* = \text{argmax}_a EU(a|z)$$

- Model over simplifies real situation and we work with estimates $\widehat{EU}$ of EU
- Estimates are unbiased if $E\left[\widehat{EU}(a|z) - EU(a|z)\right] = 0$
- Even with unbiased estimate, outcome is worse than what estimated
  - Suppose k choices, each of which has true estimated utility 0
  - Suppose also error in each estimate has zero mean and standard deviation 1
  - Sometimes the error is positive (optimistic) and sometimes negative (pessimistic)
  - We choose action with highest estimate, hence we favor optimistic estimates (source of bias)

Agents and Irrationality

- **Normative theory:**
  describes how an agent should act
- **Descriptive theory:**
  describes how an agent does act
- **Certainty effect** in human behavior:
  people are strongly attracted to gains that are certain
- **Regret**:
  give up a certain prize for a high probability of getting a better prize, and loose

Markov Decision Processes

Deterministic environment: solution is easy (obtained with pure search)

Non-deterministic environment:

- Requires a **transition model** $P(s'|s,a)$
- Transitions are **Markovian** (first-order Markov process)
- In each state *s* the agent receives a **reward R(s)** (> 0 or < 0, but bounded)
- **Utility:** sum of received rewards

Markov Decision Process

Problem represented as a **Markov Decision Process (MDP):**

$$< S, A, T, R >$$

- set of states S
- set of actions A
- transition model $T = P(s'|\boldsymbol{s}, a)$
- reward function $R(s)$ (or sometimes $R(s, a, s')$)

Additional important properties:

- fully observable
- Markovian transition model
- additive rewards

Markov Decision Process

Problem represented as a **Markov Decision Process (MDP):**

$$< S, A, T, R >$$

- set of states S
- set of actions A
- transition model $T = P(s'|\mathbf{s}, a)$ ***why not z?***
- reward function $R(s)$ (or sometimes $R(s, a, s')$)

Additional important properties:

- fully observable
- Markovian transition model
- additive rewards

Markov Decision Process

Problem represented as a **Markov Decision Process (MDP):**

$$< S, A, T, R >$$

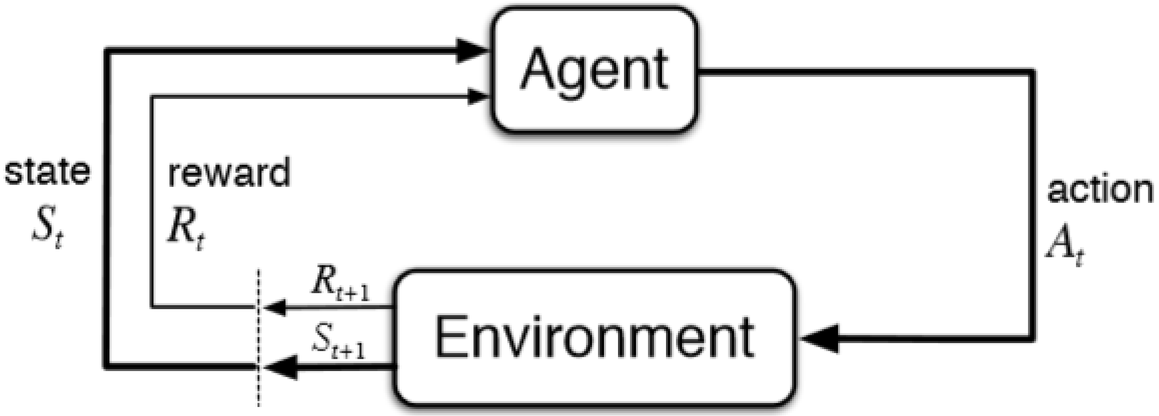
- set of states *S*
- set of actions *A*
- transition model $T = P(s'|\boldsymbol{s}, a)$ ***why not z? because it's fully observable***
- reward function $R(s)$ (or sometimes $R(s, a, s')$)

Additional important properties:

- fully observable
- Markovian transition model
- additive rewards

Finite MDPs

- **Agent** interacts with the **environment** (everything that cannot be changed by agent)
  - Agent selects actions $a \in A(s)$
  - Environment presents situations *s* and gives rewards *r*



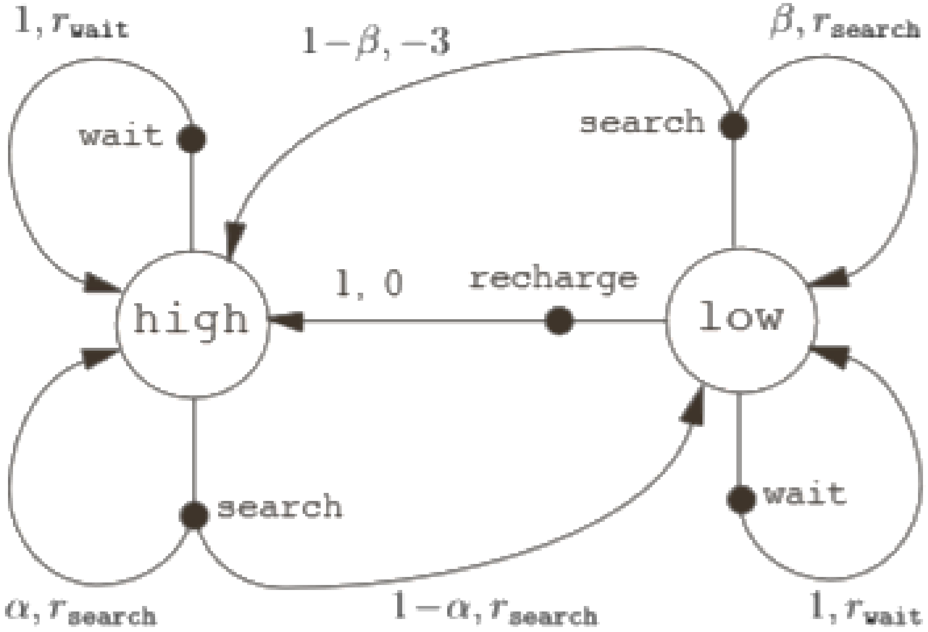
- Interaction at discrete timesteps $t = 0, 1, 2, 3, \ldots$
- MDP and agent generate a **trajectory** $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots$
- In a **finite** MDP, sets of states, actions and rewards have finite number of elements
  - R and S have discrete probability distributions only depending on previous s and a (Markov property)

$$p(s', r | s, a)$$

Finite MDP Example

- Recycling robot that collects empty soda cans in an office
- Set state represents the charge level $S = \{\text{high}, \text{low}\}$
- Action sets are $A(\text{high}) = \{\text{search}, \text{wait}\}$ and $A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-

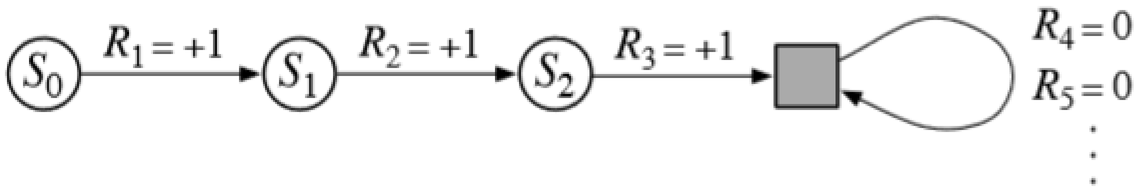


Episodic Tasks

- **Goal**: maximize the total amount of reward the agent receives
  - Not immediate reward, but cumulative reward in the long run
  - Maximization of the expected value of the cumulative sum of reward
  - Reward does not say how to achieve what we want
  - Reward communicates what to do

- Generally we try to maximize **expected return**
  - Return: $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$, where T is a final timestep
  - Requires notion of final timestep:
    - Interaction with environment breaks into **episodes** (i.e., subsequences)
    - Each episode ends in a special state called **terminal state**
    - A terminal state is followed by a reset to a standard starting state or a sample from a distribution of starting states
    - Episodic tasks

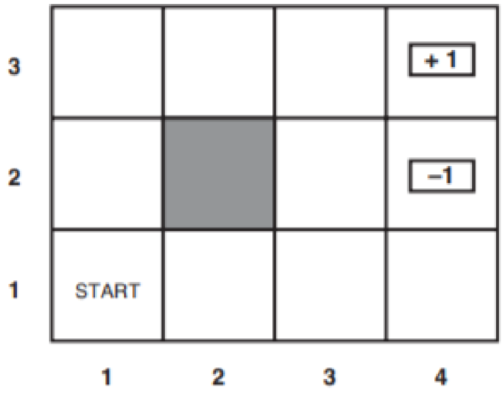
- Return formulation is problematic, because $T = \infty$, and return could be infinite as well
- We use discounting and $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
  - $0 \leq \gamma \leq 1$
  - $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = r_{t+1} + \gamma(r_{t+2} + \gamma^2 r_{t+3} + \cdots) = r_{t+1} + \gamma G_{t+1}$
  - If termination occurs $G_T = 0$
  - If termination state, that's an absorbing state that only generates transitions to itself

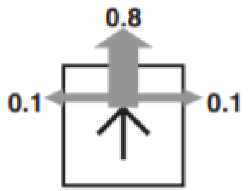


Sequential Decision Problem Example

Example scenario:

- Environment is a 4x3 grid
- Start state, from which agent chooses actions
- Agent interaction terminates in +1 or -1
- Fully observable
- Reward: -0.04 in each state except terminal (+1/-1)





- Sequence [Up, Up, Right, Right, Right] reaches goal with probability $0.8^5 = 0.32768$
- Total utility if agent receives +1 after 10 steps is 0.6

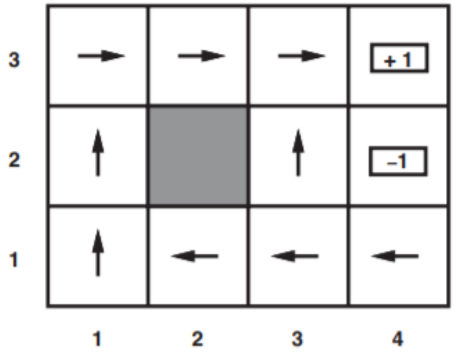
MDP Solutions

Fixed action sequence does not solve the problem

agent might end up in a state different from the goal

Solution: must specify what agent should do for any state

- This kind of solution is called **policy** π: S→A, and an action is obtained as $a = \pi(s)$
- Each time a policy is executed, a different history can be obtained
- Quality of π is measured by the *expected* utility of possible histories generated by π
- An **optimal policy** π^* is a policy that yields to the highest expected utility



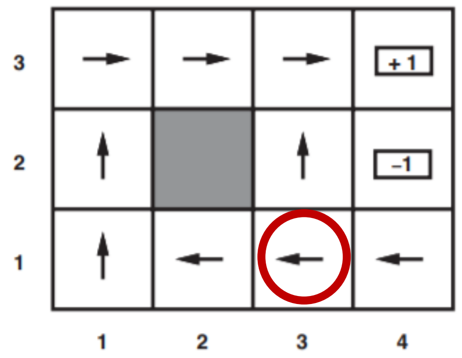
MDP Solutions

Fixed action sequence does not solve the problem

agent might end up in a state different from the goal

Solution: must specify what agent should do for any state

- This kind of solution is called **policy** π, and an action is obtained as $a = \pi(s)$
- Each time a policy is executed, a different history can be obtained
- Quality of π is measured by the *expected* utility of possible histories generated by π
- An **optimal policy** π^* is a policy that yields to the highest expected utility



Why?

Utilities Over Time

- **Finite horizon:** fixed time N after which game is over

$$U_h([s_0, \ldots, s_{N+k}]) = U_h([s_0, \ldots, s_N]) \text{ for all } k > 0$$

  Example:
  - agent starts in (3,1), N=3
  - to reach +1, agent must head directly for it → optimal action: *up* (risky)
  - if N=100, can take safe route → optimal action: *left*

  Optimal policy for a finite horizon could change → it's **nonstationary**

- **Infinite horizon:**

  Simpler, with **stationary** policy

  no reason to behave differently in the same state

Termination

If environment does not reach a terminal state or agent never reaches one:

- History is infinitely long
- Utilities with additive undiscounted rewards are generally infinite

Solutions:

- discounted rewards make an infinite sequence finite:
  - with $\gamma < 1$, with bounded reward $\pm R_{max}$

$$U_h([s_0, s_1, s_2, \dots]) = \frac{R_{max}}{1-\gamma}$$

- if environment contains terminal state, use a proper policy
  - a proper policy is guaranteed to reach a terminal state, usable with additive rewards
- use average reward per timestep for comparing infinite sequences

Rewards

- **Additive rewards:**

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

- **Discounted rewards:**

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

  γ is a **discount factor**, i.e., a number between 0 and 1.

  A discount factor describes the preference of an agent for current VS future rewards.

  - if γ close to 0 → future is insignificant
  - if γ is 1 → same as additive rewards
  - discounting is a good model of animal and human preferences

Policies

- Utility: sum of discounted rewards obtained during sequence
- Expected utility obtained by π when starting in *s*

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

probability distribution of states S_t is determined by s and π

- Expected utility can be used to compare policies
- Policy with highest expected utility (starting in s) → optimal policy

$$\pi_s^* = \text{argmax}\, U^{\pi}(s)$$

- Discounted utilities + infinite horizon → optimal policy independent of s
- True utility of a state is $U^{\pi^*}(s)$
- Expected utility allows agents to select action using maximum expected utility

$$\pi^*(s) = \text{argmax}_{a\in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

How to Find Optimal Policies

Assumption:
agent knows MDP model of the world (i.e., transition model and reward function)

Agent can *plan* its actions before interacting with environment, by using:

- Value iteration

  Idea: calculate utility of each S, then use U to select optimal action in each S

- Policy iteration

  Idea: evaluate policies and try to improve them step after step

Note: if you know and use a model of the world, you are doing planning!

Bellman Equation

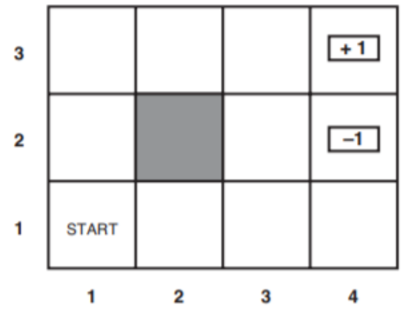
- There is a relation between utility of a state and utility of its neighbors

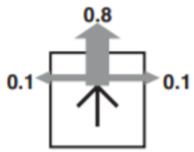
$$U(s) = R(s) + \gamma \max_{a \in A} \sum_{s'} P(s'|s,a)U(s')$$

This is known as **Bellman equation**

Example:

$$\begin{aligned} U(1,1) = -0.04 + \gamma \max[\ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && (Up) \\ & 0.9U(1,1) + 0.1U(1,2), && (Left) \\ & 0.9U(1,1) + 0.1U(2,1), && (Down) \\ & 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\ ]. && (Right) \end{aligned}$$





Value Iteration

- *n* possible states → *n* Bellman equations (one per state) → *n* unknowns
- Equations are nonlinear (max operation is nonlinear)
- Solution is hard, but we can use iterative approaches

Algorithm:

1. Start with arbitrary initial values for utilities
2. Until we reach equilibrium:
   - For each state simultaneously:

$$U_{i+1} \leftarrow R(s) + \gamma \max_{a \in A} \sum_{s'} P(s'|s,a) U_i(s')$$

- Equilibrium is guaranteed to be reached, and this is the solution to the equations
- Solutions are unique and corresponding policy is optimal

Value Iteration Pseudocode

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
   ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Policy Iteration

Algorithm:

- Iterate until utilities don't change anymore:
  1. Policy evaluation
     given a policy π_i, calculate $U_i = U^{\pi_i}$
  2. Policy improvement
     calculate a new MEU policy π_{i+1} using one-step lookahead based on U_i

Easier to do policy evaluation than value iteration (action is fixed):

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

Equation is linear: *n* states, *n* linear equations, *n* unknowns → O(n^3)

Policy Iteration Pseudocode

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Reinforcement Learning is *learning* what to do to maximize a numerical reward signal.

- Learner must discover which actions yield the most reward by trying them
- Actions might affect not only immediate reward, but also subsequent ones

Reinforcement Learning is *learning* what to do to maximize a numerical reward signal.

- Learner must discover which actions yield the most reward by trying them
- Actions might affect not only immediate reward, but also subsequent ones

- **Trial-and-error**
- **Delayed rewards**

What Reinforcement Learning Is

Reinforcement Learning is:

1. A problem (formalized using ideas from dynamical systems theory)
2. A class of solution methods that work well on the problem
3. The field that studies the problem and its solutions

Relation to optimal control:
RL is the optimal control of incompletely known Markov decision processes.

What Reinforcement Learning Is Not

Reinforcement Learning is not:

1. Supervised learning (learning from a training set of labeled examples)
   - Important but not adequate for interaction
   - Impractical to obtain correct and representative examples of desired behavior
2. Unsupervised learning (finding structure hidden in unlabeled data)
   - RL uses a reward signal
   - Useful but does not solve the problem

Reinforcement Learning Challenges

Agent has to:

- ***Exploit*** what it has already been experienced to obtain a reward
- ***Explore*** in order to make better action selections in the future

Challenges:

- Exploration-exploitation trade-off (both have to be achieved)
- On stochastic tasks, each action must be tried many times (to correctly estimate expected reward)
- Considers whole problem of a goal-directed agent (not subproblems)
- Significant uncertainty about the environment

Reinforcement Learning Inter-Discipline

Reinforcement Learning benefits from fruitful interactions with other scientific disciplines.

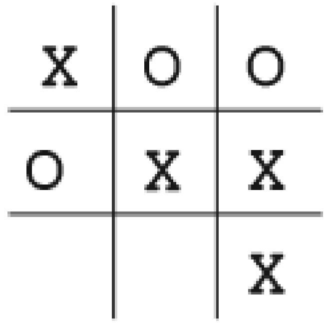
- General machine learning
  - Use of function approximators to address the curse-of-dimensionality
  - Work towards simple general principles for AI
- Psychology
- Neuroscience

Elements of Reinforcement Learning

- **Policy**
  - Defines agent's way of behaving (maps states to actions)
  - Can be a simple function, lookup table, or it may involve extensive computation as search
  - Generally stochastic
- **Reward signal**
  - Defines the goal of a RL problem (*immediate* number sent from environment)
  - Agent's sole objective is maximizing total reward received in the long run
  - Primary basis for altering the policy
  - May be a stochastic function of state and action
- **Value function**
  - Specifies what is good in the *long run*
  - Value of a state is the total reward an agent can expect to accumulate over future, from that state.
- Optionally, a **model of the environment**
  - Mimics the behavior of the environment to allow inference to be made
  - Used for planning (i.e., model-based vs model-free methods)

Reinforcement Learning: Example

- **Game:** Tic-Tac-Toe
- **Assumptions:**
  - Player is not perfect
  - Draws and Losses are equally bad



- **Available classical solutions:**
  - Minimax
    - Wrong assumptions on player (she would never reach a loosing state)
  - Optimization for Markov Decision Processes
    - E.g., dynamic programming. Requires a full specification of the opponent (e.g., her probabilities)
  - Evolutionary methods
    - E.g., hill-climbing in policy space. Hundreds of evaluations for little improvement
    - Each policy change is made only after many games
    - Does not exploit the state-action structure: what happens during the game is ignored

Reinforcement Learning: Example

SAPIENZA UNIVERSITÀ DI ROMA

- **Game:** Tic-Tac-Toe
- **Assumptions:**
  - Player is not perfect
  - Draws and Losses are equally bad

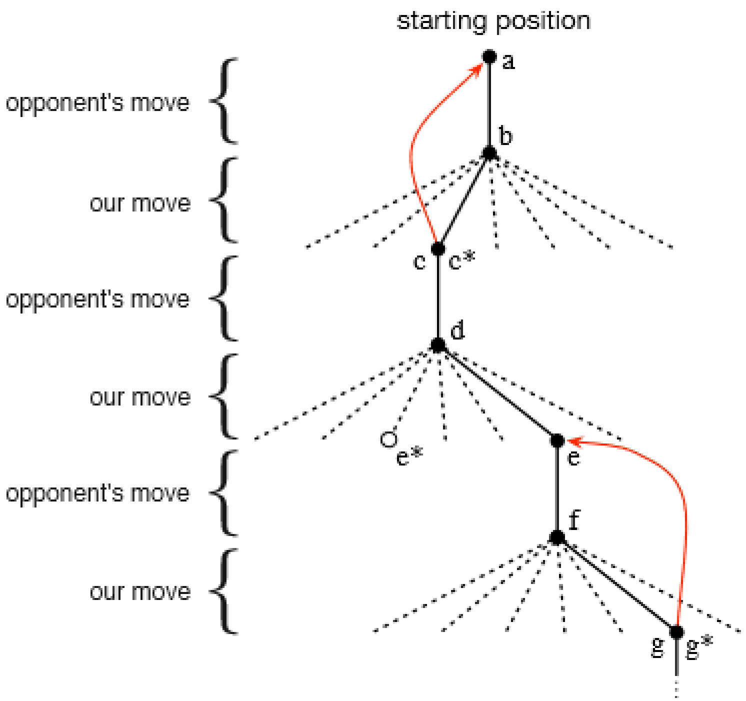
X	O	O
O	X	X
		X

- **Value function methods:**
  - Set up a table of numbers, one per state as the probability of winning from that state
    - The estimate is the state value
    - The table is the value function
    - Initial value is set to 0.5
  - To select our moves, we examine the table and we move *greedily* wrt values
  - Occasionally, we select a random *exploratory* move
  - While playing, we change values of states to make them more accurate estimates
    - We “back up” the value of the state after each greedy move to the state before the move

Reinforcement Learning: Example

- **Game:** Tic-Tac-Toe
- **Assumptions:**
  - Player is not perfect
  - Draws and Losses are equally bad

- **Value function methods:**
  - Perform quite well on this task
  - If appropriate back-up, method converges for any opponent to the true values
  - Optimal against imperfect player
  - Individual states are evaluated



Reinforcement Learning Applicability

- **Reinforcement Learning is applicable to:**
  - Problems without adversaries
  - Problems that do not break down into separate episodes
  - Continuous time problems
  - Problems with very large or infinite state space
  - Problems where part of the state is hidden

Tabular Solutions & Bandit Problems

- We introduce core RL algorithms in their simplest forms
  - State and action spaces are *small* enough to be represented as tables
  - Tabular methods find *exact* solutions (i.e., optimal value function and policy)

- **Bandit problems**
  - Only a single state exists
  - Special case of RL
  - Studies evaluative aspect of RL in simplified setting where you act in one situation
  - Avoids much of the complexity of the full RL problem

k-armed Bandit Problem Description

- You're faced repeatedly with a choice among *k* different actions
- After each choice you receive a reward from a stationary probability distribution
  - Depending on the chosen action
- Objective: maximize expected total reward over some time period (e.g., 1000 timesteps)

- Analogy: slot-machine
  - Each action selection is like a play of one of the slot machine's levers
  - Rewards are payoffs for hitting the jackpot
  - Through repeated action selection you want to maximize winnings by concentrating on the best lever

k-armed Bandit Problem Formulation

- Each of the *k* actions has an expected reward r
- Expected reward corresponds to the *value* $q_*(a)$ of the action *a*

$$q_*(a) = \mathrm{E}[r_t|a_t]$$

- Greedy actions are the ones that have maximum value, and exploit current knowledge
- Nongreedy actions lead to exploration that improves estimates
- If we knew the value, the problem would be solved (choosing action with best value)

Assumption:

- We do not know the action values with certainty, although we might have estimates
- Estimated value of *a* at timestep *t* is $Q_t(a)$

Goal:

- We want $Q_t(a)$ to be as close as possible to $q_*(a)$

Exploitation Vs Exploration

- **Exploitation**:
  - Right thing to do to maximize the expected reward on each step
- **Exploration**:
  - Lower reward in the short run
  - Leads to greater total reward in the long run
  - Uncertainty leads to have at least one action that probably is better than greedy
- Tradeoff depends on many factors:
  - Uncertainty
  - Number of remaining steps
  - Values to estimate

Action-value Methods

- **Action-value methods** estimate values of actions to make action selection decisions
- True value of an action is the mean reward when that action is selected
  - This can be obtained by averaging received rewards

$$Q_t(a) = \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}$$

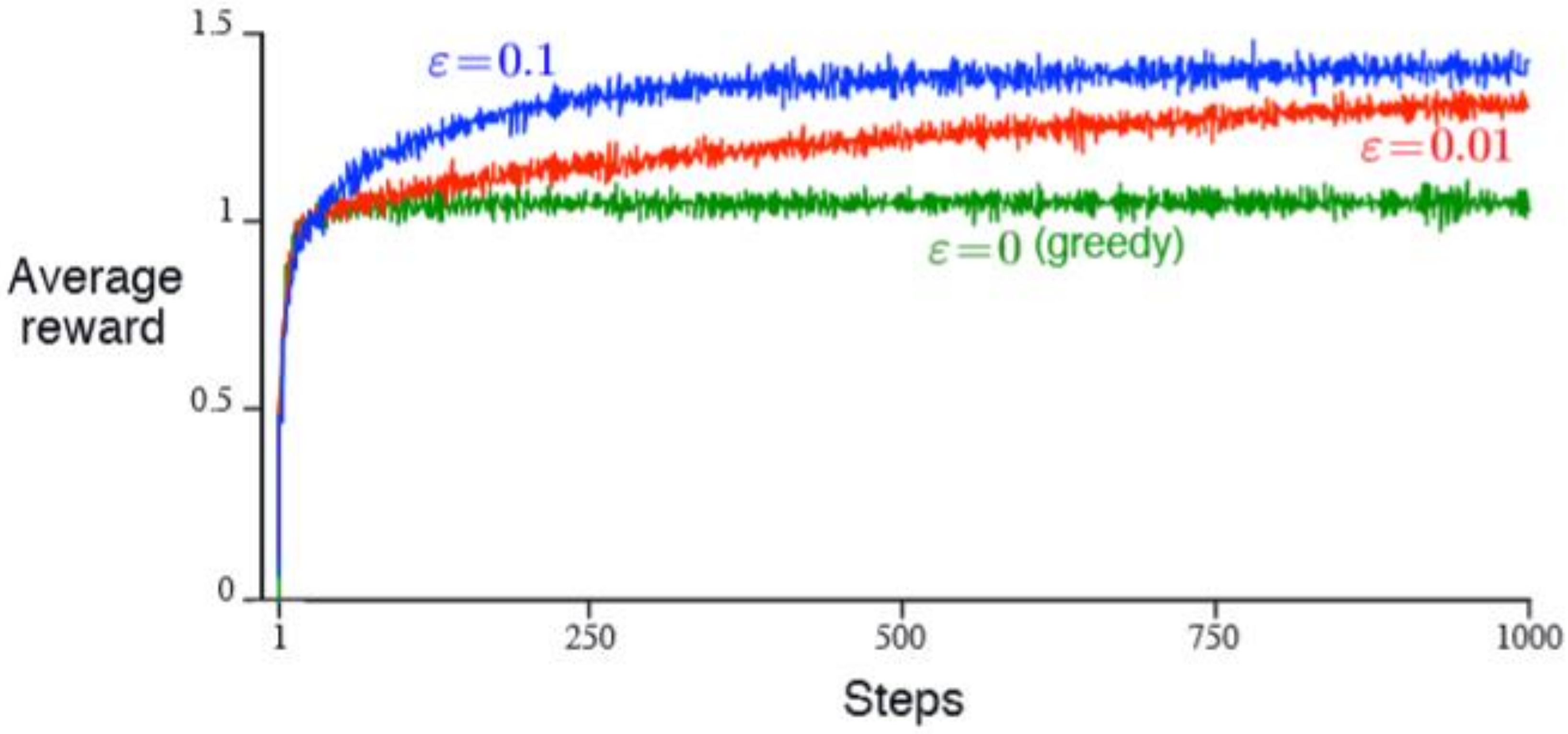
  - If denominator is zero, we define $Q_t(a)$ as a default value (e.g., 0)
  - As denominator goes to infinity, $Q_t(a)$ converges to $q_*(a)$

- Action selection rule is to select actions with highest estimated value
  - If more than one greedy action, selection is made in arbitrary way (e.g., randomly)

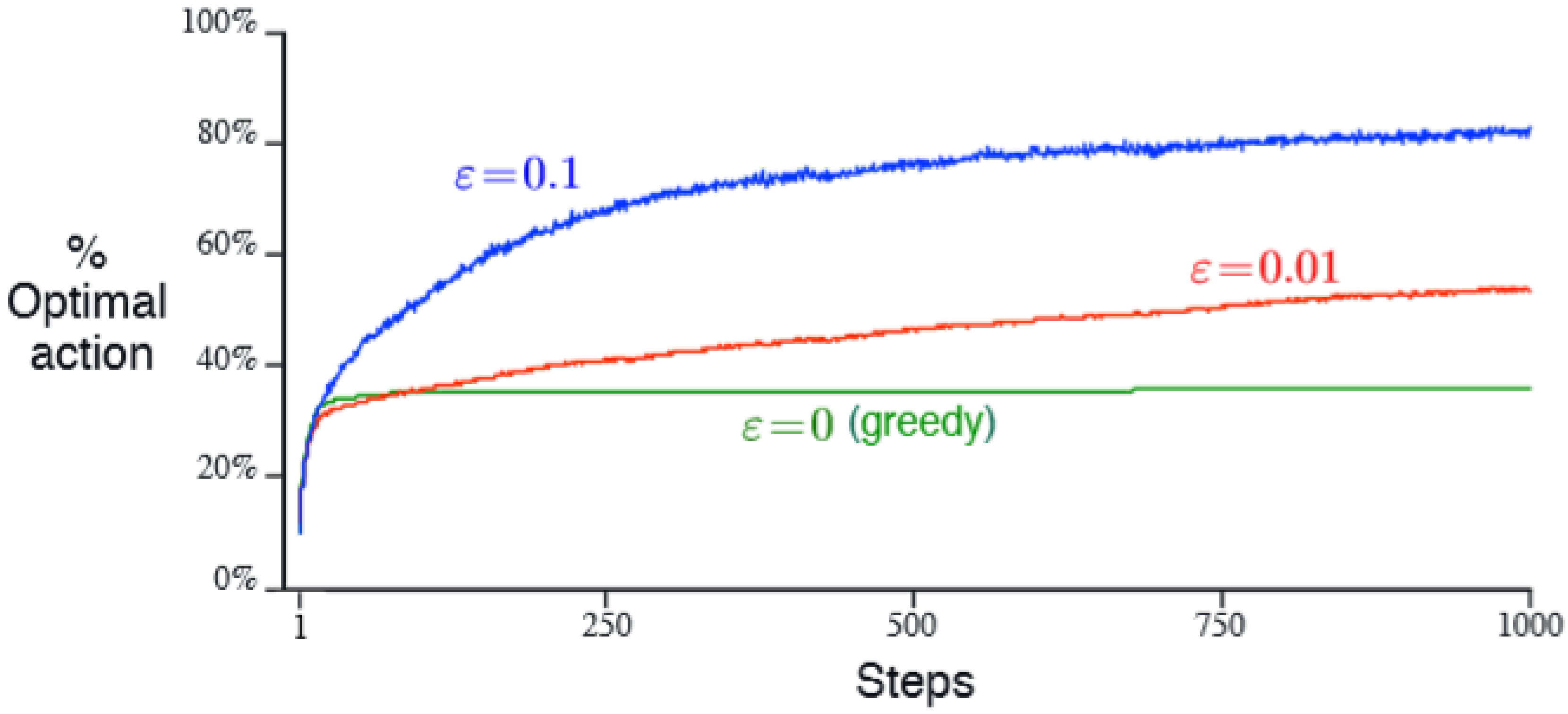
$$a_t = \mathrm{argmax_a Q}_t(a)$$

  - Argmax is purely greedy
  - A simple alternative is ϵ-greedy (mostly greedy, with probability ϵ random)
    - In the limit every action is sampled infinitely, ensuring convergence

Greedy Vs Nongreedy Selection



Greedy Vs Nongreedy Selection



Obvious Implementation

Sapienza Università di Roma

- Action-value methods seen so far estimate action values as average of observed rewards

- Let r_i be the reward received after the *i*th selection of a certain action
- Let Q_n denote the estimate of that action value after selecting *n-1* times

$$Q_n = \frac{r_1 + r_2 + \ \dots + r_{n-1}}{n-1}$$

- Obvious implementation: maintain all rewards and then perform computation
  - Expensive! Requires additional memory and computation at every reward

Incremental Implementation

Sapienza Università di Roma

- How can action-values be computed **efficiently** (both in space and time)?

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} r_i = \frac{1}{n}\left(r_n + \sum_{i=1}^{n-1} r_i\right) = \frac{1}{n}\left(r_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} r_i\right) = \frac{1}{n}(r_n + (n-1)Q_n)$$
$$= \frac{1}{n}(r_n + nQ_n - Q_n) = Q_n + \frac{1}{n}[r_n - Q_n]$$

- 1/n is a step size that changes over time

Frequent Update Rule

- $Q_n + \frac{1}{n}[r_n - Q_n]$ is an update rule of a form that we will see often:

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}[\text{Target} \; - \text{OldEstimate}]$$

- [Target – OldEstimate] is the error in the estimate
- Error is reduced by taking a step towards Target (i.e., a desirable direction)
- Step size is generally denoted as α

Simple Bandit Algorithm

Initialize, for $a = 1$ to k:

$$Q(a) \leftarrow 0$$
$$N(a) \leftarrow 0$$

Loop forever:

$$A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$
$$R \leftarrow bandit(A)$$
$$N(A) \leftarrow N(A) + 1$$
$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$$

Stationary Vs Nonstationary Bandits

Stationary bandit problem: problem in which reward probabilities **do not** change over time
Nonstationary bandit problem: problem in which reward probabilities **do** change over time

- Averaging methods are appropriate for stationary bandit problems
- Often RL problems are nonstationary
  - Popular choice: recent rewards should get more weight
  - Can be obtained using constant step size

$$Q_{n+1} = Q_n + \alpha[r_n - Q_n]$$

where $\alpha \in (0, 1]$ is constant

Nonstationary Problems

- For nonstationary problems a popular choice is to use a constant step size
- Q_{n+1} is a weighted average of past rewards and initial estimate Q_1

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha[r_n - Q_n] = \alpha r_n + (1-\alpha)Q_n = \alpha r_n + (1-\alpha)[\alpha r_{n-1} + (1-\alpha)Q_{n-1}] \\
&= \alpha r_n + (1-\alpha)\alpha r_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha r_n + (1-\alpha)\alpha r_{n-1} + (1-\alpha)^2 \alpha r_{n-2} + \cdots + (1-\alpha)^{n-1}\alpha r_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} r_i
\end{aligned}
$$

- The weight of the reward decays exponentially as the number of rewards increases
- If $1-\alpha = 0$ all the weight goes to the last reward ($0^0 = 1$)
- Never completely converges, continues to vary in response to most recent rewards

Optimistic Initial Values

- All discussed methods depend on initial action-value estimate
  - They are biased by initial estimates
- For sample average, bias disappears once all actions are selected at least once
- For constant step size, bias is permanent but decreasing over time

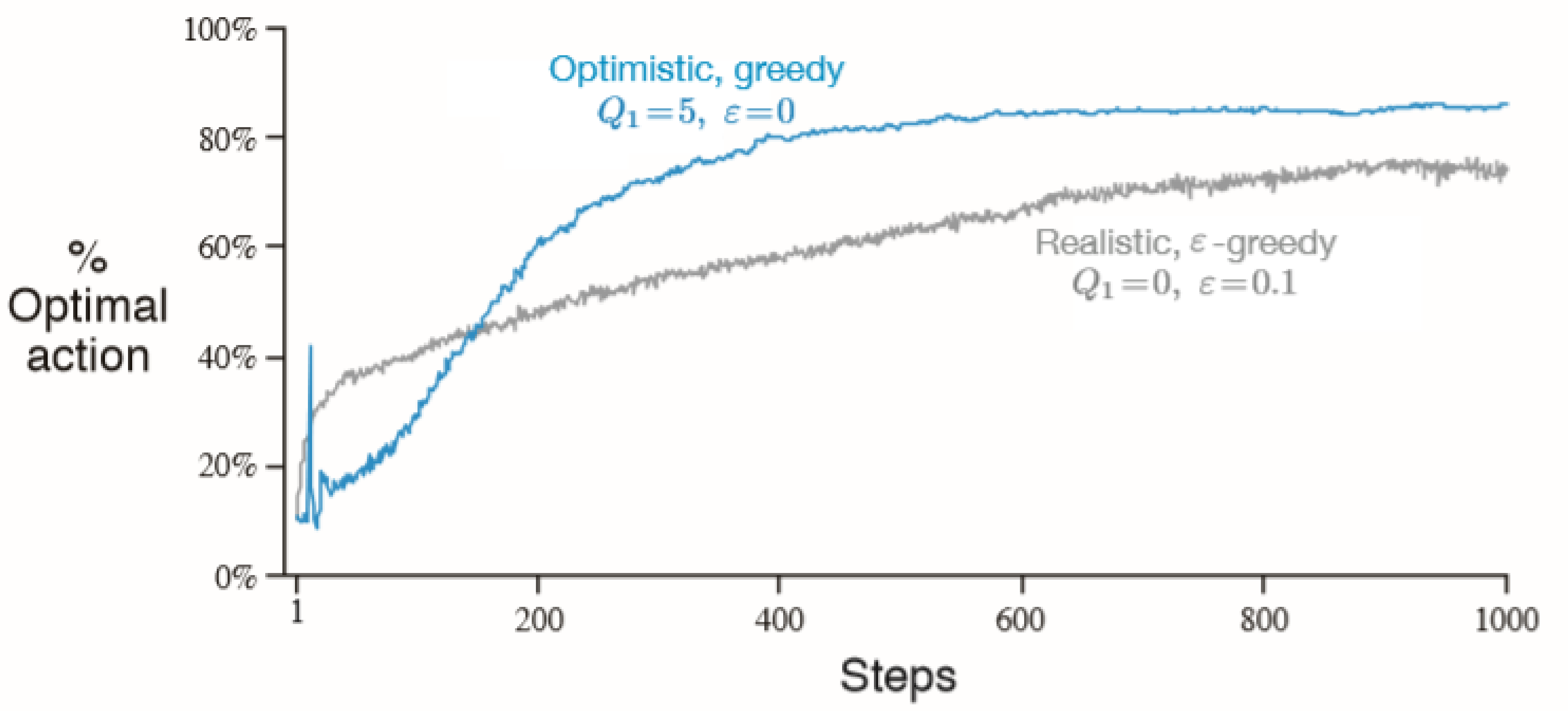
Pros:

- Usually not a problem
- Sometimes can be helpful
- Can provide prior knowledge
- Can be used to encourage exploration
  - Optimistic initialization (i.e., high initial values) can make the agent disappointed
  - Disappointed agent chooses different actions

Cons:

- Initial estimate must be picked by user

Optimistic Initial Values



Optimistic Initial Values

- Not well suited for nonstationary problems
  - If task changes, new exploration is needed and this does not help
- Beginning occurs only once

Upper-Confidence-Bound Selection

- Greedy actions are those that look best currently
  - Some other might look better
- ϵ-greedy selection forces nongreedy actions to be tried without preference
  - Preference could exist for nearly greedy or particularly uncertain actions
  - It would be better to select according to potential of being optimal

- Potential of being optimal can take into account:
  - Closeness to max
  - Uncertainty

$$a_t = \text{argmax}_a \left[Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}} \right]$$

- $N_t(a)$ is the number of times a has been selected prior to t (if 0, a is considered a max action)
- $c > 0$ controls degree of exploration

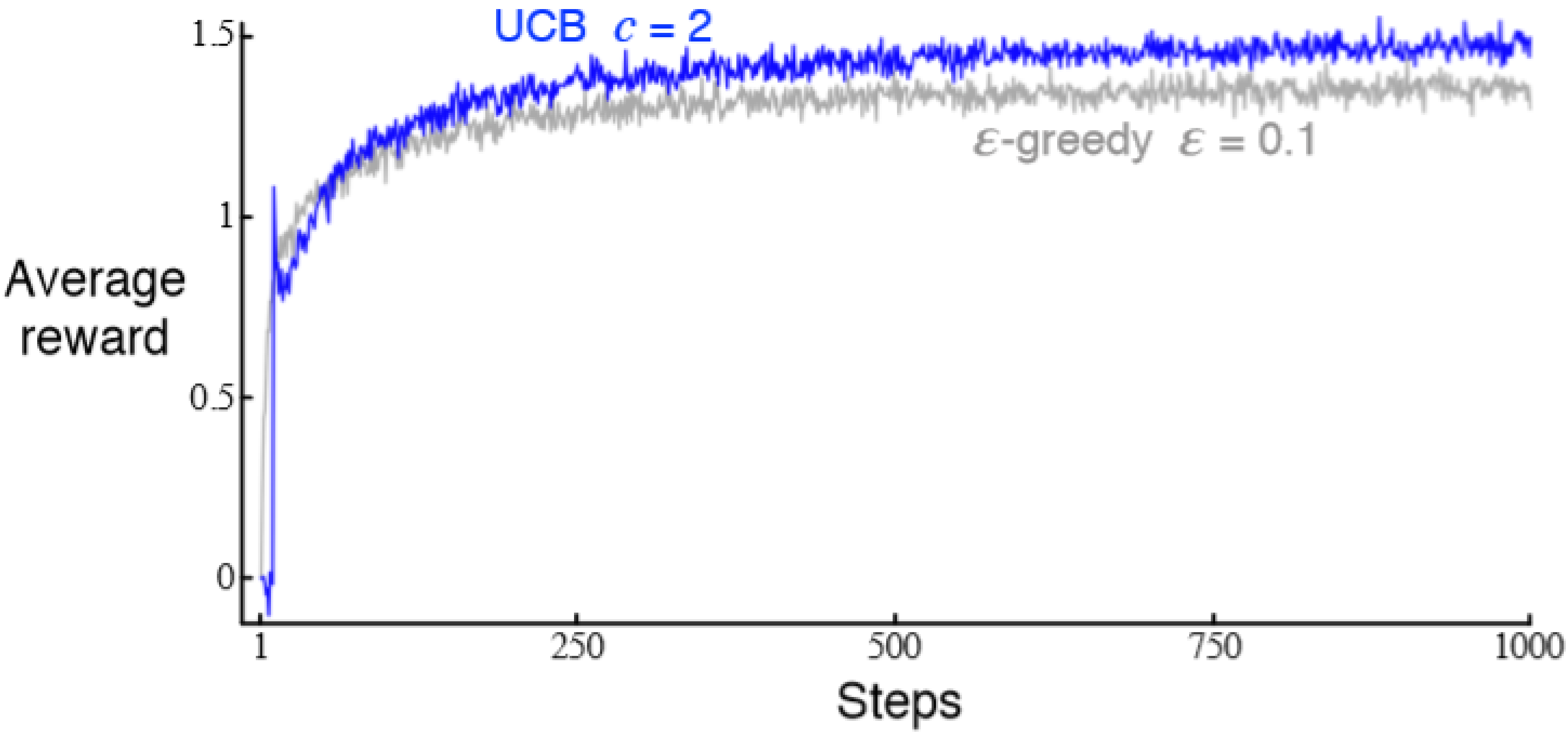
Upper-Confidence-Bound Selection

Upper confidence bound (UCB) selection

$$a_t = \mathrm{argmax}_a \left[Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}} \right]$$

- Square root term measures uncertainty (variance) in estimate of *a*
- Maximizing an upper bound on the possible true value of *a*
- c determines the confidence level
- Every time action is chose, uncertainty is reduced
- If the action is not selected, uncertainty increases (via *t*)
- Difficult to use with large state spaces

Upper-Confidence-Bound Selection



Gradient Bandit Algorithms

- Instead of directly using action values, we can learn a numerical preference $H_t(a)$
- The larger the preference, the more often the action is taken
- Preference has no interpretation in terms of reward
- Only relative preference is important
- Action probabilities are determined according to a *soft-max distribution*

$$p(a_t) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a)$$

- Initially all action preferences are the same
- Natural learning algorithm for this setting is based on stochastic gradient ascent

$$H_{t+1}(a_t) = H_t(a_t) + \alpha(r_t - \overline{r_t})\big(1 - \pi_t(a_t)\big)$$
$$H_{t+1}(a) = H_t(a) - \alpha(r_t - \overline{r_t})\pi_t(a) \text{ for all } a \neq a_t$$

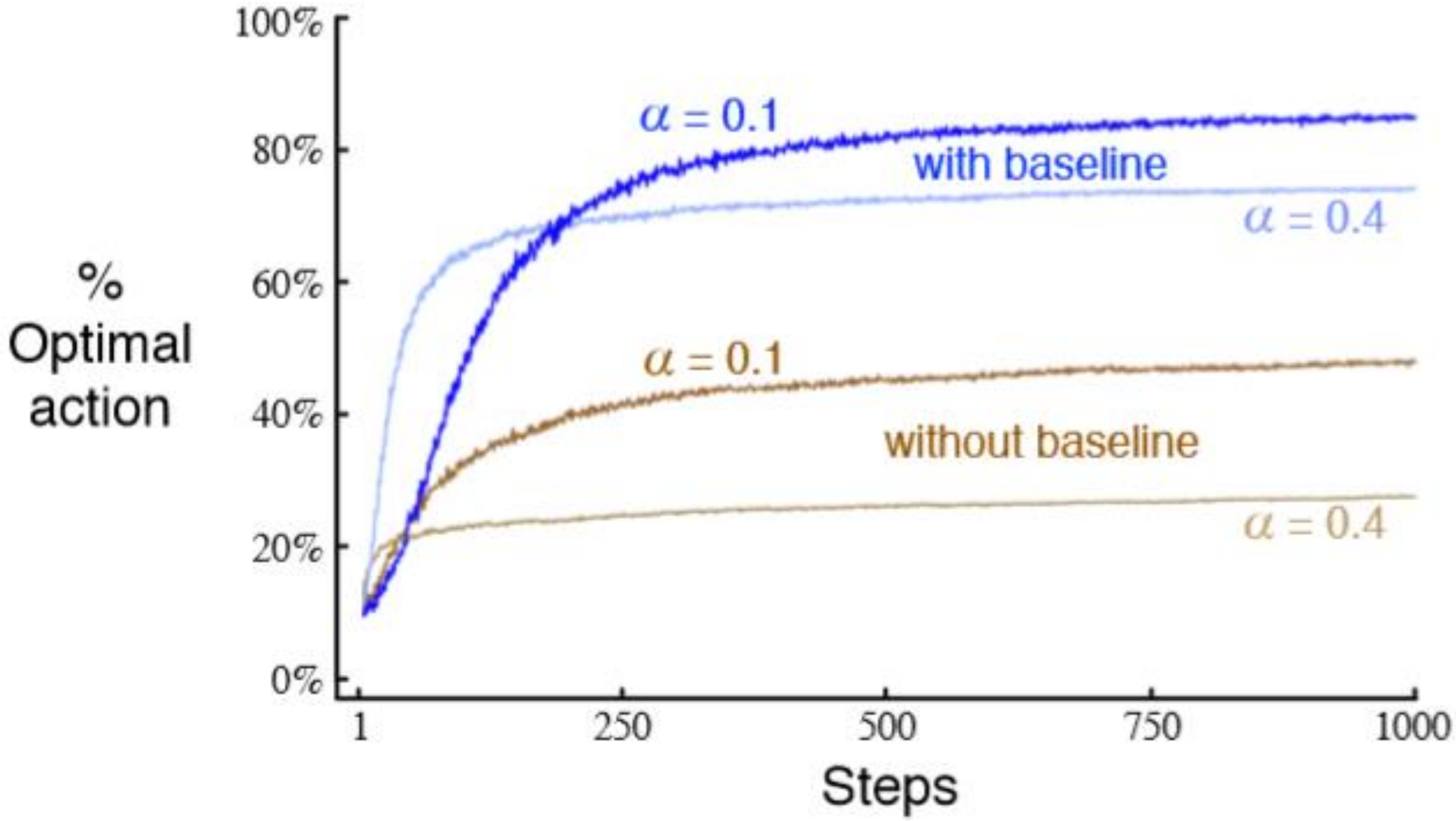
Baseline

$$H_{t+1}(a_t) = H_t(a_t) + \alpha(r_t - \overline{r_t})\big(1 - \pi_t(a_t)\big)$$
$$H_{t+1}(a) = H_t(a) - \alpha(r_t - \overline{r_t})\pi_t(a) \text{ for all } a \neq a_t$$

- $\bar{r}$ is the average reward, and serves as baseline for comparing reward
  - If reward is higher than baseline, probability of a_t is increased
  - If reward is lower than baseline, probability is decreased
  - Non-selected actions move in the opposite direction

- α is the step size

Gradient Bandit Algorithms



Gradient Derivation

- In exact gradient ascent, each action preference would be updated as

$$H_{t+1}(a) = H_t(a) + \alpha \frac{d\mathrm{E}[r_t]}{dH_t(a)}$$

$$\mathrm{E}[r_t] = \sum_x \pi_t(x) q_*(x)$$

$$\frac{d\mathrm{E}[r_t]}{dH_t(a)} = \frac{d}{dH_t(a)}\left[\sum_x \pi_t(x) q_*(x)\right] = \sum_x q_*(x) \frac{d\pi_t(x)}{dH_t(a)} = \sum_x (q_*(x) - B_t) \frac{d\pi_t(x)}{dH_t(a)}$$

- B is the baseline, and can be any scalar
- Baseline can be included without changing equality
  - $\sum_x \frac{d\pi_t(x)}{dH_t(a)} = 0$ over all actions
  - As $H_t(a)$ changes, some action probabilities go up and some other go down (must sum to 1)

Gradient Derivation

- It is not possible to implement gradient ascent exactly, because we do not know $q_*(x)$

$$\frac{d\mathrm{E}[r_t]}{dH_t(a)} = \sum_x (q_*(x) - B_t)\frac{d\pi_t(x)}{dH_t(a)} = \sum_x \pi_t(x)(q_*(x) - B_t)\frac{d\pi_t(x)}{dH_t(a)}/\pi_t(x)$$

$$= \mathrm{E}\left[(\mathrm{q}_*(\mathrm{a_t}) - \mathrm{B_t})\frac{\mathrm{d}\pi_t(a_t)}{dH_t(a)}/\pi_t(a_t)\right] = \mathrm{E}\left[(r_t - \bar{r_t})\frac{\mathrm{d}\pi_t(a_t)}{dH_t(a)}/\pi_t(a_t)\right]$$

- Remember: $\mathrm{E}[r_t|a_t] = q_*(a_t)$, and the expected choice of the policy is a_t
- If we assume that $\frac{d\pi_t(x)}{dH_t(a)} = \pi_t(x)\big(\mathrm{I}_{a=x} - \pi_t(a)\big)$, where $\mathrm{I}_{a=x}$ is 1 if *a=x,* else 0

$$\mathrm{E}\left[(r_t - \bar{r_t})\frac{\mathrm{d}\pi_t(a_t)}{dH_t(a)}/\pi_t(a_t)\right] = \mathrm{E}\left[(r_t - \bar{r_t})\pi_t(a_t)\left(\mathrm{I}_{a=a_t} - \pi_t(a)\right)/\pi_t(a_t)\right] = \mathrm{E}\left[(r_t - \bar{r_t})\left(\mathrm{I}_{a=a_t} - \pi_t(a)\right)\right]$$

Gradient Derivation

- Substituting expectation with samples (as we get from environment), we get

$$H_{t+1}(a) = H_t(a) + \alpha(r_t - \bar{r}_t)\left(\mathrm{I}_{a=a_t} - \pi_t(a)\right)$$

- How do we obtain $\frac{d\pi_t(x)}{dH_t(a)} = \pi_t(x)\left(\mathrm{I}_{a=x} - \pi_t(a)\right)$? (Remember $\frac{de^x}{dx} = e^x$)

$$\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{\frac{df(x)}{dx}g(x) - f(x)\frac{dg(x)}{dx}}{g(x)^2}$$

$$\frac{d\pi_t(x)}{dH_t(a)} = \frac{d}{dH_t(a)}\pi_t(x) = \frac{d}{dH_t(a)}\left[\frac{e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}}\right] = \frac{\frac{de^{H_t(x)}}{dH_t(a)}\sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)}\frac{d\sum_{y=1}^{k} e^{H_t(y)}}{dH_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2}$$

$$= \frac{\mathrm{I}_{\mathrm{a=x}}\mathrm{e}^{H_t(x)}\sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)}e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2} = \frac{\mathrm{I}_{\mathrm{a=x}}\mathrm{e}^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}} - \frac{e^{H_t(x)}e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2} = \mathrm{I}_{\mathrm{a=x}}\pi_t(x) - \pi_t(x)\pi_t(a)$$

$$= \pi_t(x)\left(\mathrm{I}_{a=x} - \pi_t(a)\right)$$

Contextual Bandits

- **Nonassociative tasks**: no need to associate different actions with different situations
- **General RL task**: more then one situation, with policy mapping situations to actions

- We need to extend nonassociative tasks to associative settings
- Suppose there are different k-armed bandits, each of them clearly identified
  - Its action value is not given
  - A policy can be learned that maps each task to the best action for that task

- Associative search tasks are known as **contextual bandits**
  - Like a full RL problem (with multiple states/situations)
  - Like k-armed bandits, each action affects only immediate reward
  - If actions affect also next situation, we have a full RL problem

State Values Vs State-Action Values