Reinforcement Learning:

From basic concepts to deep Q-networks

Joelle Pineau
McGill University
Facebook AI Research Montreal

African Masters of Machine Intelligence (Kigali, Rwanda)
January 6-10 2020

Machine Intelligence

Predictions

vs

Decisions

Actions and consequences

- Intelligent agents should not only be *observers*, but also *actors*.
  I.e. they should choose actions in a rational way.

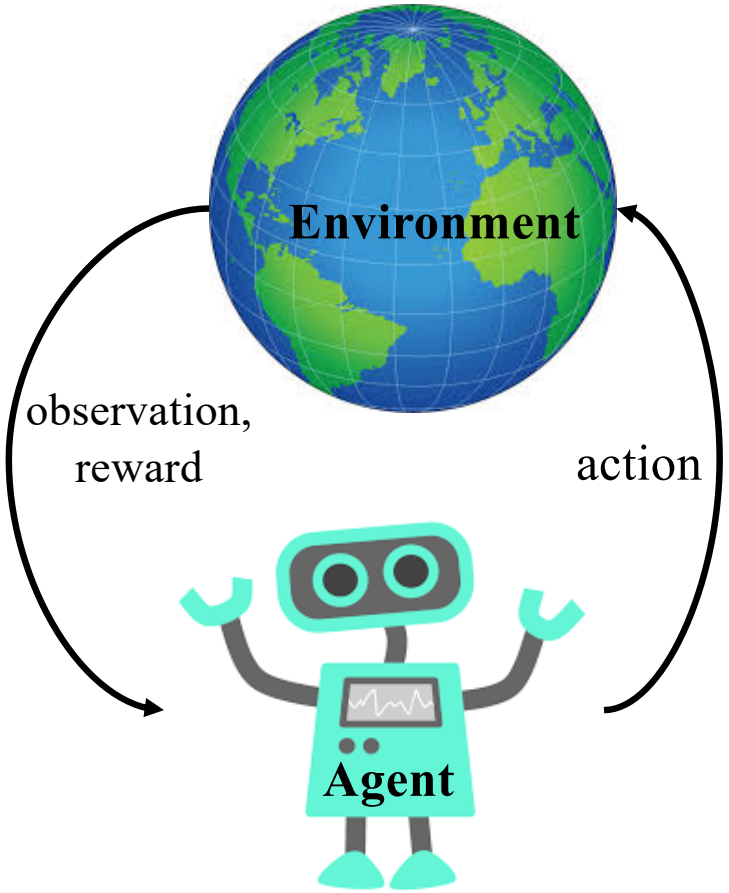
- Most often, actions produce *consequences*, which cause changes in the world.

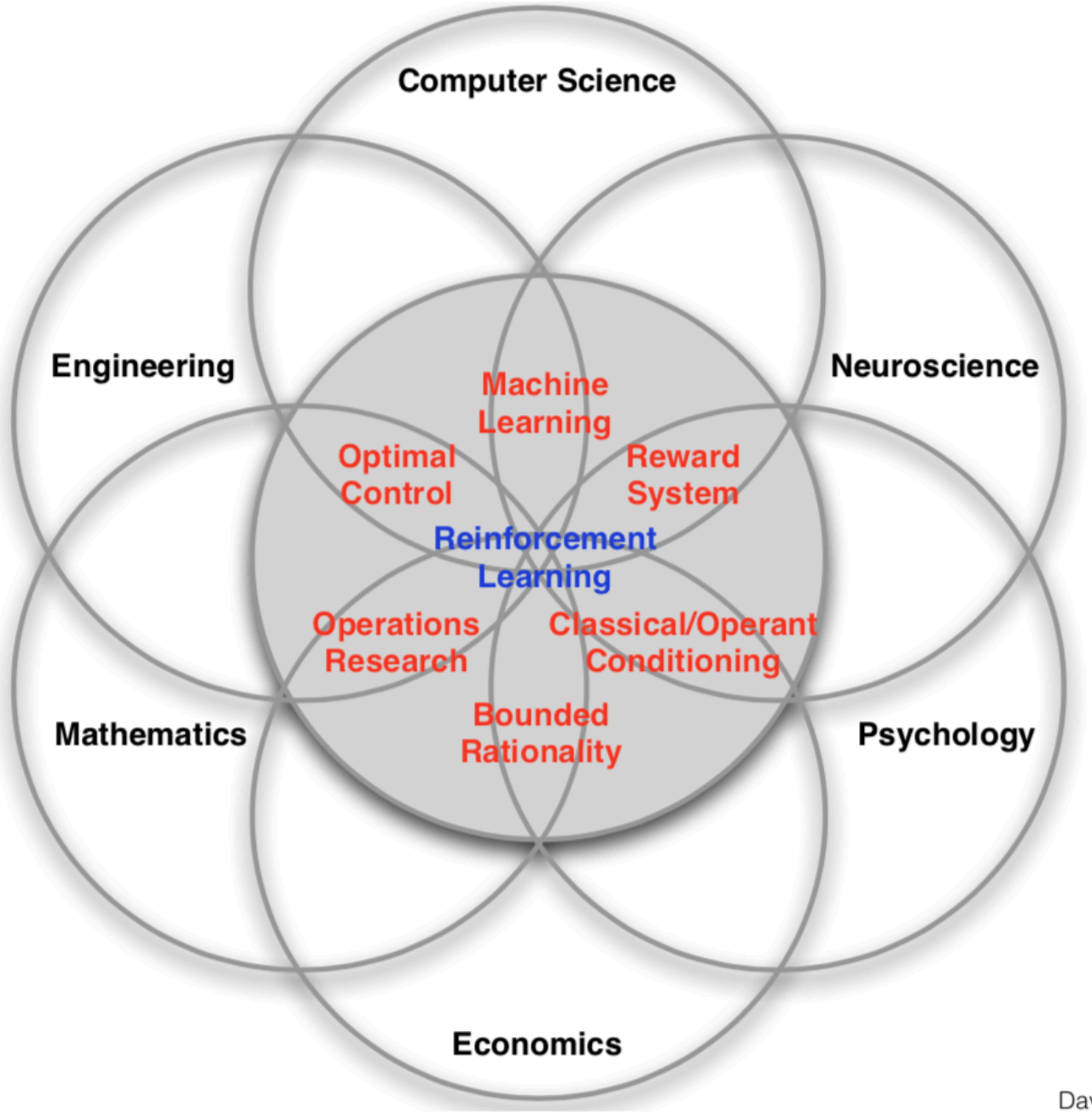
- Decision-making should maximize the overall utility of the agent's actions.

Reinforcement learning

- Inspired by psychology
  - Agent + Environment
  - Agent selects actions to maximize utility function

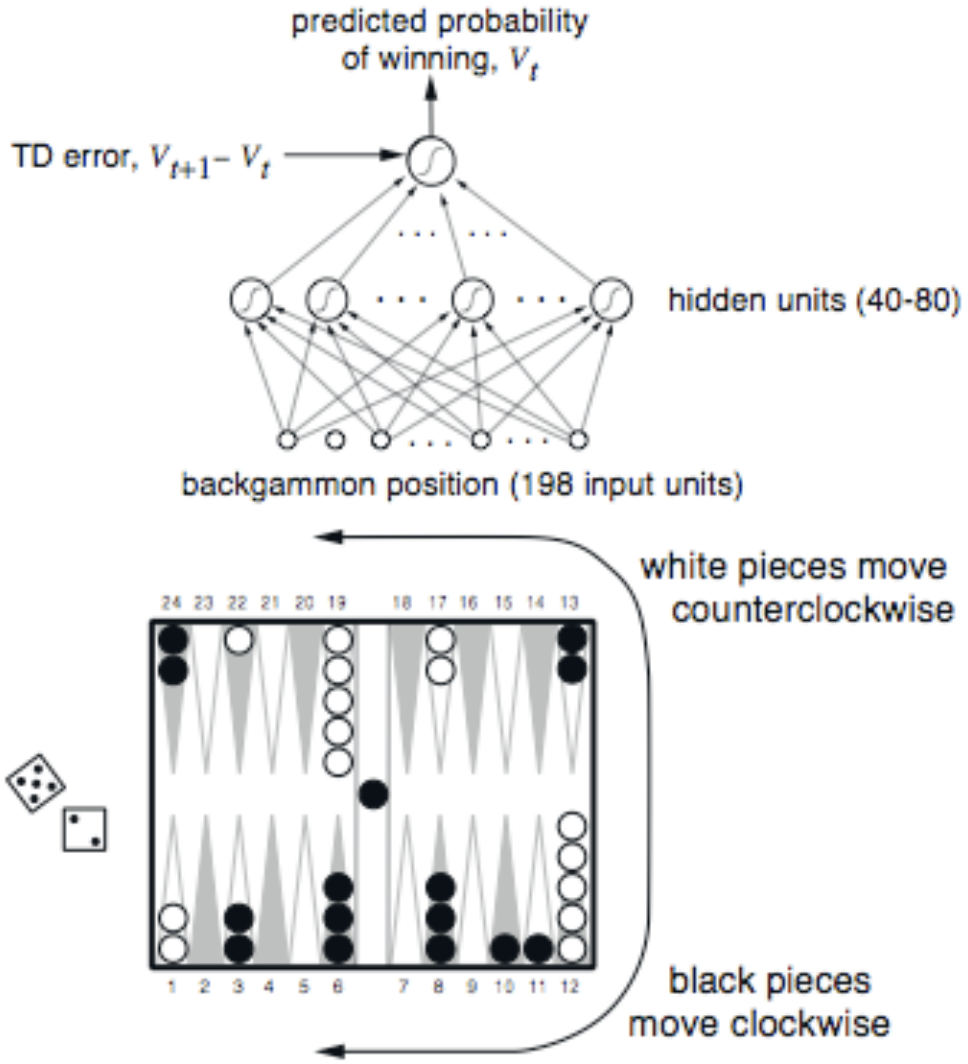
- Learning by trial-and-error, in real-time.
- Improves with experience





David Silver 2015

RL system circa 1990's: TD-Gammon



Reward function:

+100 if win

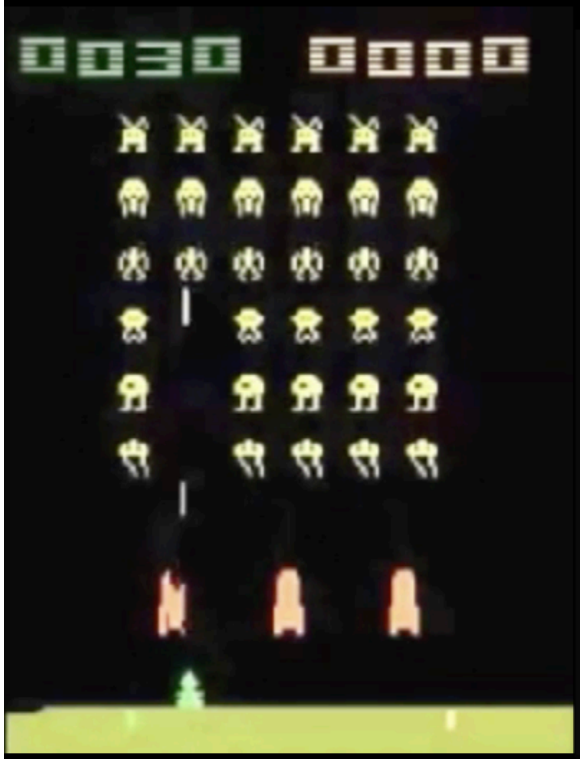
- 100 if lose

0 for all other states

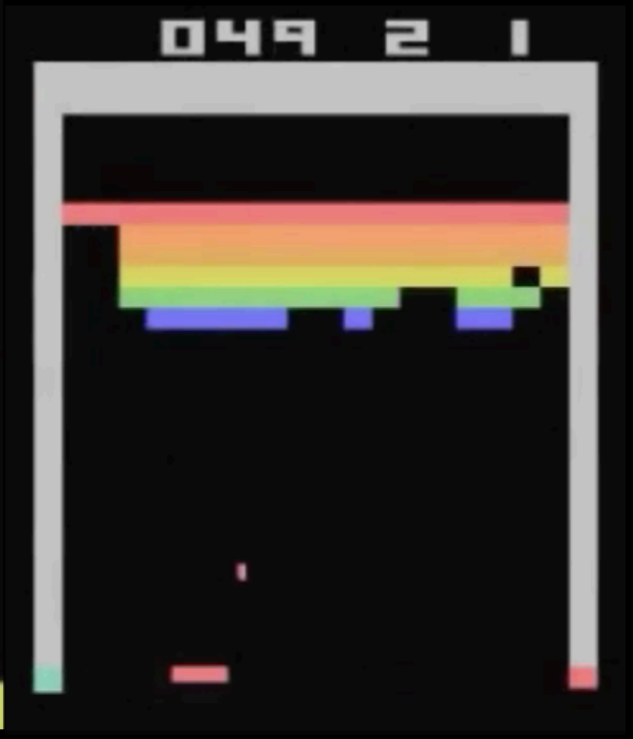
Trained by playing 1.5×10^6 games against itself.

Enough to beat the best human player.

RL + Deep Learing Performance on Atari Games



Space Invaders



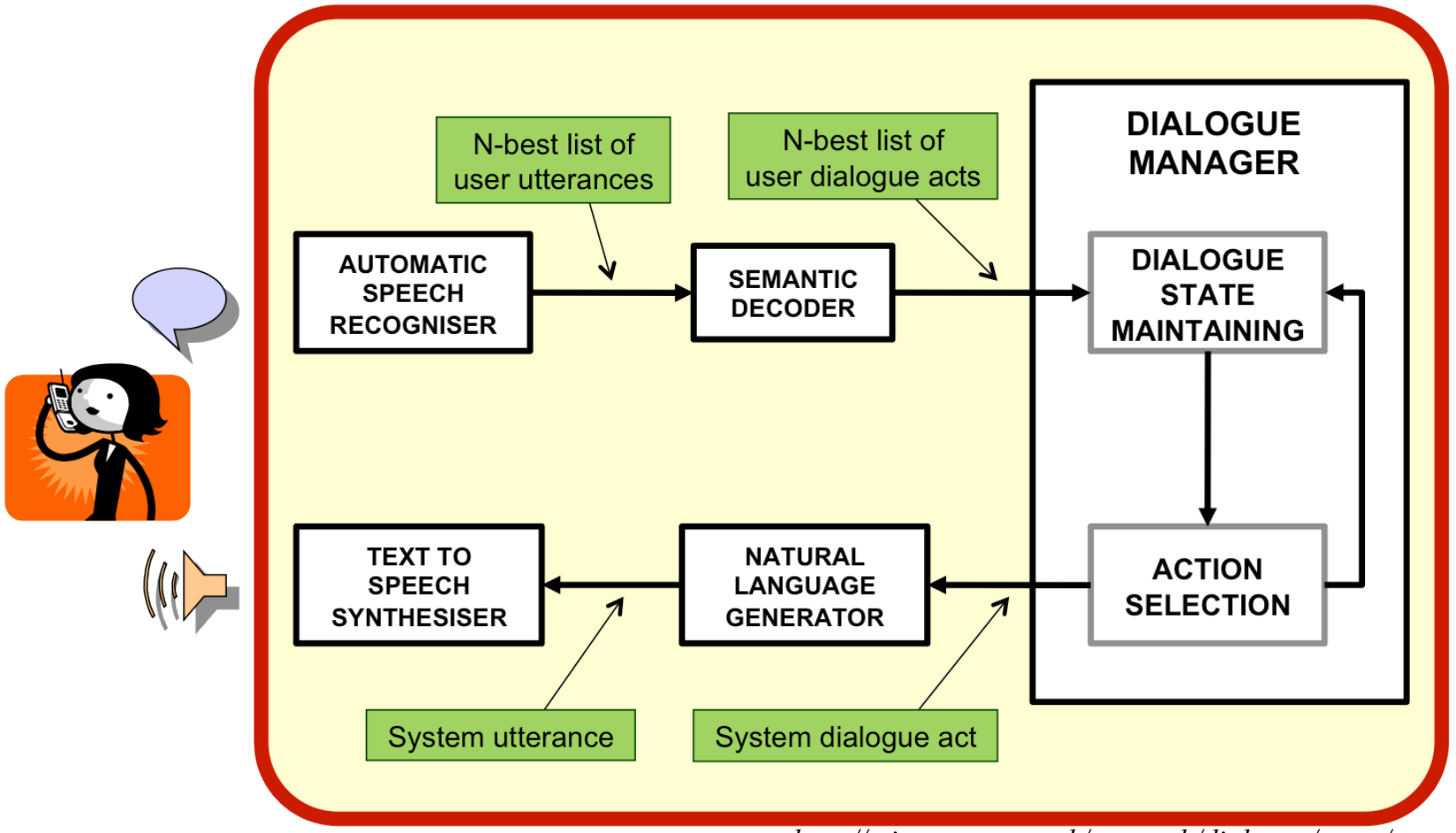
Breakout

Enduro

2016: World Go Champion Beaten by Deep Learning

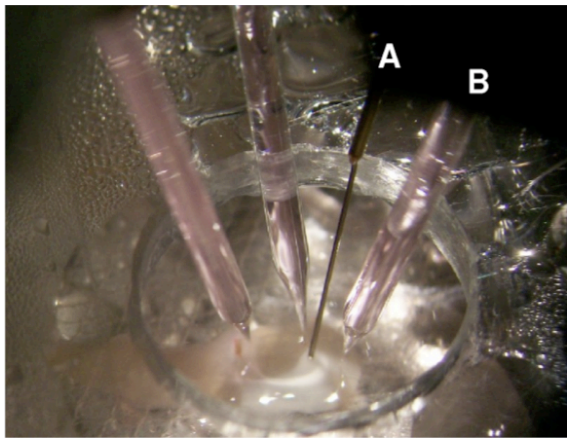
RL and robotics

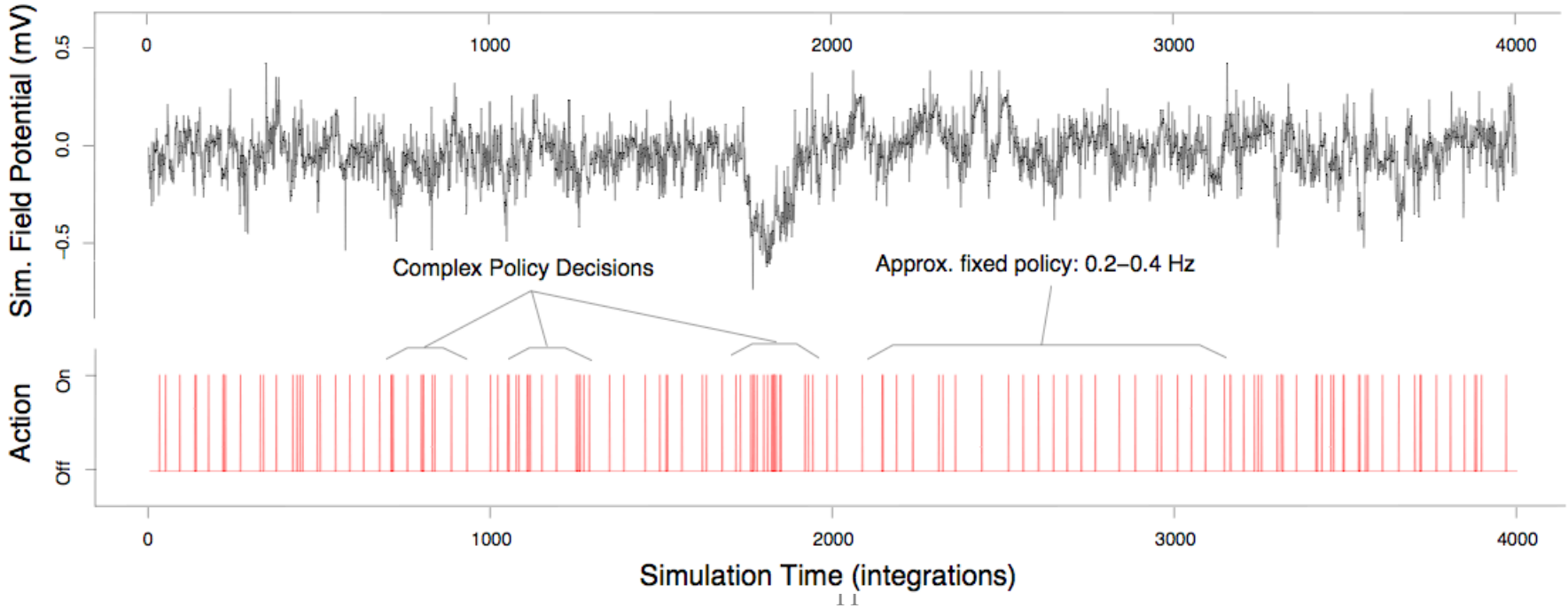
Dialogue systems



http://mi.eng.cam.ac.uk/research/dialogue/epsrc/

Adaptive neurostimulation for epilepsy suppression





Course overview

- **Monday**
  - Utility theory (Russell & Norvig)
  - Multi-armed bandits (Sutton & Barto, Ch.2)
  - Markov decision processes
  - Value iteration, Policy iteration

- **Wednesday**:
  - Basics of reinforcement learning
  - Monte-Carlo methods, Temporal difference
  - Eligibility traces

- **Friday**:
  - Value function approximation
  - Fitted Q-iteration
  - Deep Q networks

Useful references:

Slides:

Doina
Sergey
Hado
COMP-424

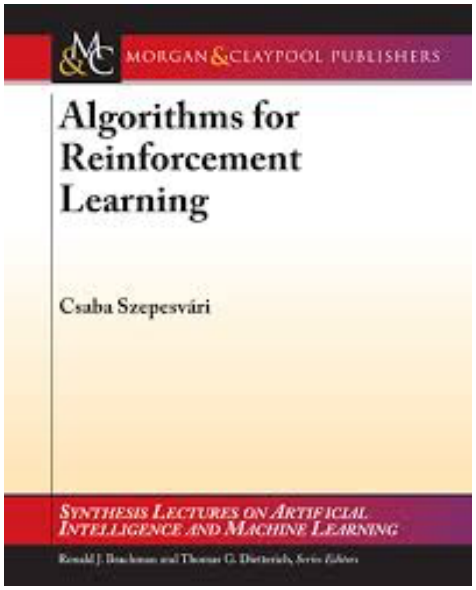
Environments & algorithms:

http://glue.rl-community.org/wiki/Main_Page
https://gym.openai.com
https://github.com/deepmind/lab

Comprehensive list of resources:

https://github.com/aikorea/awesome-rl

Sutton & Barto, 1998



Szepesvari, 2010

Three theories

1. **Probability theory**:
   - Allows us to model an uncertain, stochastic world.
   - Describes what an agent should believe on the basis of evidence.

2. **Utility theory**:
   - Describes what an agent wants.

3. **Decision theory**:
   - Describes what a rational agent should do (based on Probability theory and Utility theory).

     => Reinforcement Learning

Example: Buying a soccer ticket

Many possible consequences:

1. You watch the game, have a good evening, and get back home.
2. You start watching the game, but your neighbour keeps coughing on you, and you end up in hospital with turbeculosis.
3. You watch the game, but when you get back home you find that your cat ate your parrot, but not before eating your homework.
4. You watch the game, when you try to head back home the city bus is not running, but your favorite player passes by and gives you a ride.
5. …

How would you choose between buying and not buying a ticket???

Preferences

- Actions have consequences. We call the consequences of an action payoffs or rewards.

- A rational method would be to evaluate the benefit (desirability, value) of each consequence and weigh it by its probability.

- To compare different actions, denoted a, we need to know for each one:
  - set of consequences $C_a = \{ c_1, \ldots, c_m \}$
  - probability distribution over consequences $P_a(c_i)$, s.t. $\sum_i P_a(c_i) = 1$.

Remember: Expected value

- For a discrete-valued random variable X, with n possible values $\{x_1, \ldots, x_n\}$, occurring with probabilities $p_1, \ldots, p_n$ respectively.

- Then the expected value (mean) of X is:

$$E[X] = \Sigma_{i=1:n} p_i x_i$$

Utilities

- A utility function, $U(x)$, maps states (denoted x) to real values.

- Given a preference behaviour, the utility function is **non-unique**.
  E.g. behavior is invariant w.r.t. additive linear transformations:
  $U'(x) = k_1 U(x) + k_2$, *where* $k_1 > 0$

- Utilities don't need to obey the same laws as expected values.

Money

- Suppose you had to choose between these two scenarios:

  A_1: win \$1M for sure.

  A_2: win \$5M with prob. 0.1
  win \$1M with prob. 0.89
  win \$0 with prob 0.01.

- Which would you choose?

Money (2)

- Suppose you had to choose between these two scenarios:

A_1: win $1M for sure.

A_2: win $5M with prob. 0.1
win $1M with prob. 0.89
lose $1M with prob 0.01.

- Which would you choose?
- What if you were the president of a bank?

Money (3)

- Suppose you had to choose between these two scenarios:

  A_1: win \$5M with prob 0.1
  win \$0 with prob 0.9.

  A_2: win \$1M with prob. 0.3
  win \$0 with prob 0.7.

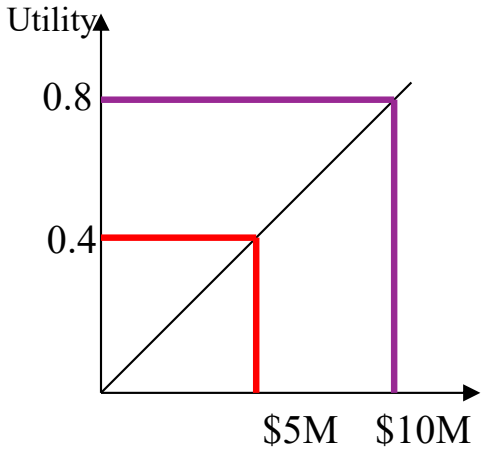
- Which would you choose?

Most people are risk-adverse!

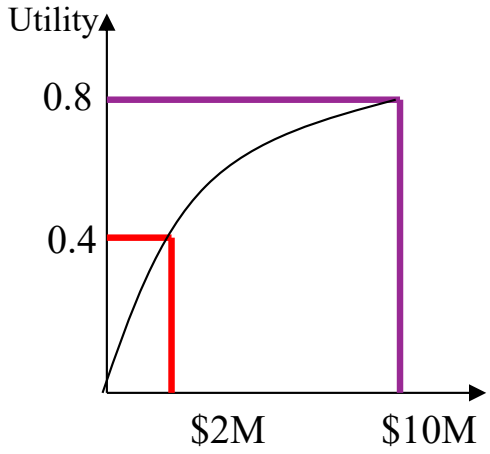
Utility Models

- Capture preferences for rewards and resource consumption.
- Capture risk attitude

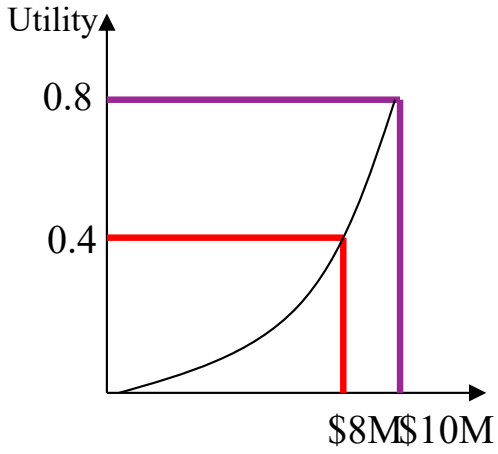
E.g. If risk-neutral, getting $5M has half the utility of getting $10M.



Risk Neutral
(Utility= Expected reward)

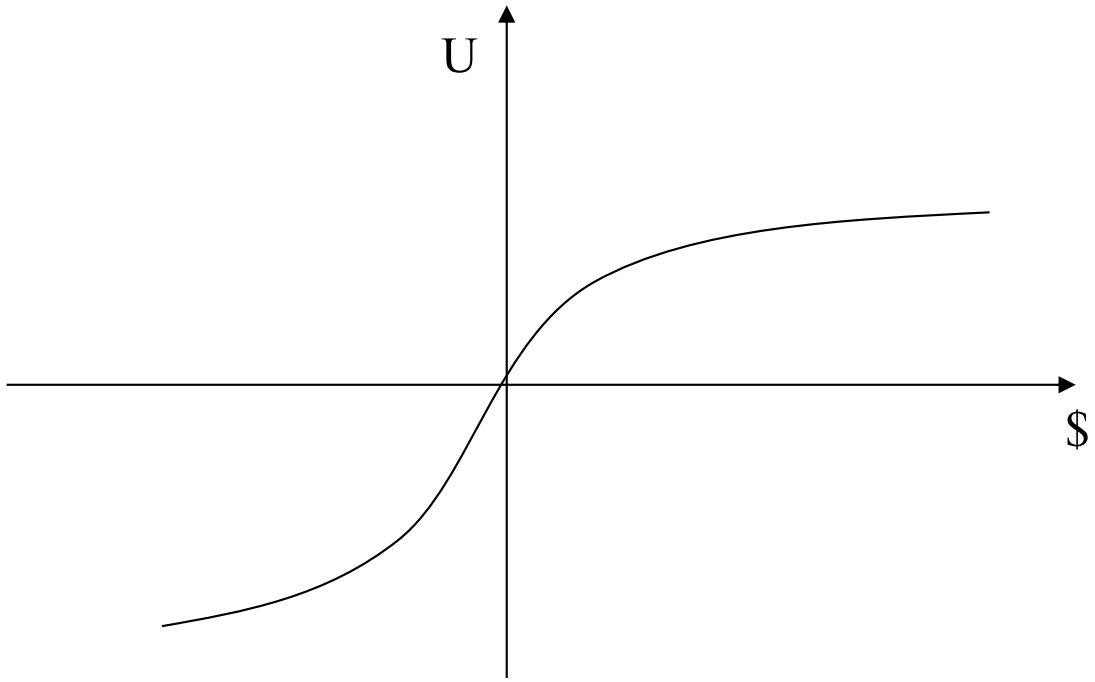


Risk Averse



Risk Seeking

The utility of money



- Decision-theory is normative: describes how rational agents should act.
  => Useful to define an optimization criteria for AI agents.

- People systematically violate the axioms of utility theory!
  => Or maybe we don't understand their utility function?

Maximizing expected utility (MEU)

MEU principle: Choose the action that maximizes expected utility.
Most widely accepted as a standard for rational behavior.

Theorem (Ramsey, 1931; von Neumann and Morgenstern, 1944):

Given preferences that satisfy the axioms of utility theory,
there exists a real-valued function U such that:

$$A \gtrsim B \text{ iff } U(A) \geq U(B)$$

where

$$U([p_1, C_1; \ldots; p_n, C_n]) = \sum_i p_i U(C_i)$$

Example: single-stage decision-making

- One random variable, *X*: does the child have an ear infection or not?
- One decision, *d*: give antibiotic (yes) or not (no)
- **Utility function**: associates a real value to the possible states of the world and possible decisions.

	$X = \text{no}$	$X = \text{yes}$
$d = \text{no}$	0	-50
$d = \text{yes}$	-100	10

- Unfortunately *X* is not directly observable!
- But we know *Pr(X=yes) = 0.1* and *Pr(X=no)=0.9*.
- According to MEU what is the best action?

Maximizing expected utility

- Compute:

$$EU(d = \text{no}) = 0.9 \times 0 + 0.1 \times (-50) = -5$$

$$EU(d = \text{yes}) = 0.9 \times (-100) + 0.1 \times 10 = -89$$

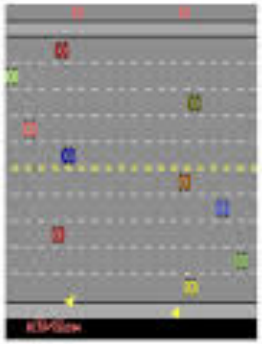
- Best action give this utility function and probability is *d* = no.

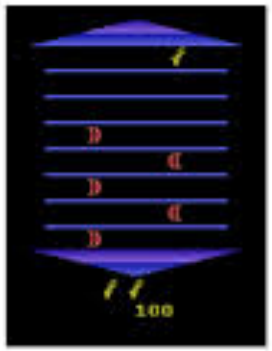
Useful definitions for utility theory

- Utility function: $U(x)$
  - Numerical expression of the desirability of a state
- Expected Utility: $EU(a \mid x) = \sum_i Pr(Effect_i(a) \mid x) U(Effect_i(a))$
  - Utility of an action weighted by the expected outcome of that action
- Maximum Expected Utility: $max_a EU(a \mid x)$
  - Best average payoff that can be achieved in situation x.
- Optimal Action: $argmax_a EU(a \mid x)$
  - Action chosen according to the MEU principle.
- Policy: $\pi(x): X \rightarrow A$
  - A strategy for picking actions in all states.

Many applications of RL: What is the utility?

- Robotics
- Medicine
- Advertisement
- Resource management
- Game playing …





The *k*-armed Bandit Problem

- On each of an infinite sequence of *time steps*, t=1, 2, 3, …, you choose an action A_t from k possibilities, and receive a real-valued *reward* R_t

- The reward depends only on the action taken; it is identically, independently distributed (i.i.d.):

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a], \quad \forall a \in \{1, \ldots, k\}$$

true values

- These true values are *unknown.* The distribution is unknown

- Nevertheless, you must maximize your total reward

You must both try actions to learn their values (**explore**), and prefer those that appear best (**exploit**).

The Exploration/Exploitation Dilemma

- Suppose you form estimates
  $$Q_t(a) \approx q_*(a), \quad \forall a$$
  action-value estimates

- Define the *greedy action* at time *t* as
  $$A_t^* \doteq \arg\max_a Q_t(a)$$

- If $A_t = A_t^*$ then you are *exploiting*
  If $A_t \neq A_t^*$ then you are *exploring*

- You can’t do both, but you need to do both

- You can never stop exploring, but maybe you should explore less with time. Or maybe not.

Action-Value Methods

- Methods that learn action-value estimates and nothing else
- For example, estimate action values as *sample averages*:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbf{1}_{A_i=a}}$$

- The sample-average estimates converge to the true values
  If the action is taken an infinite number of times

$$\lim_{N_t(a)\to\infty} Q_t(a) = q_*(a)$$

The number of times action a has been taken by time t

ε-Greedy Action Selection

- In greedy action selection, you always exploit

- In ε-greedy, you are usually greedy, but with probability ε you instead pick an action at random (possibly the greedy action again)

- This is perhaps the simplest way to balance exploration and exploitation

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

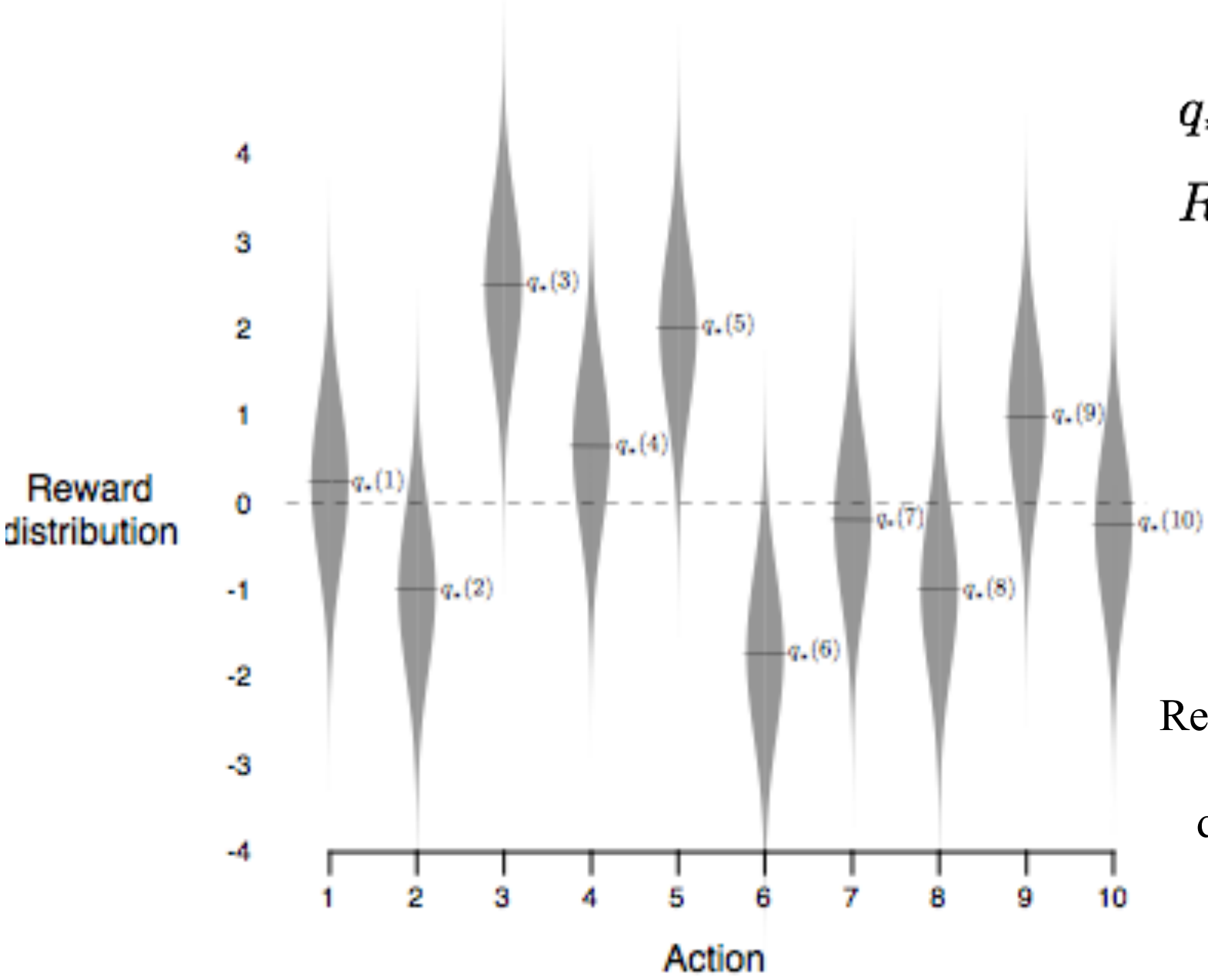
$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

The 10-armed Testbed



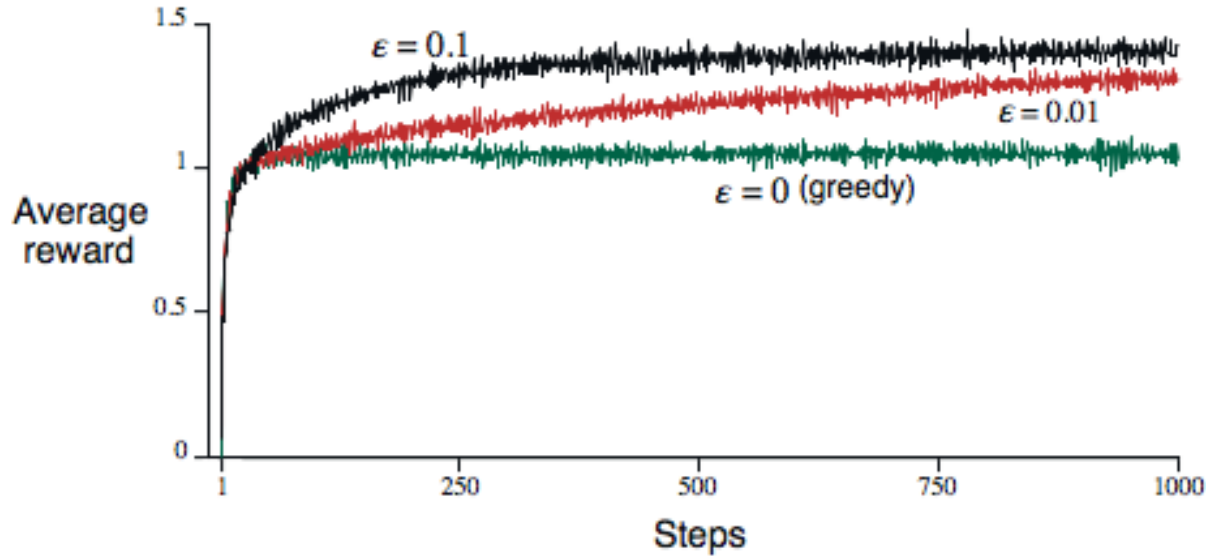
$$q_*(a) \sim \mathcal{N}(0, 1)$$

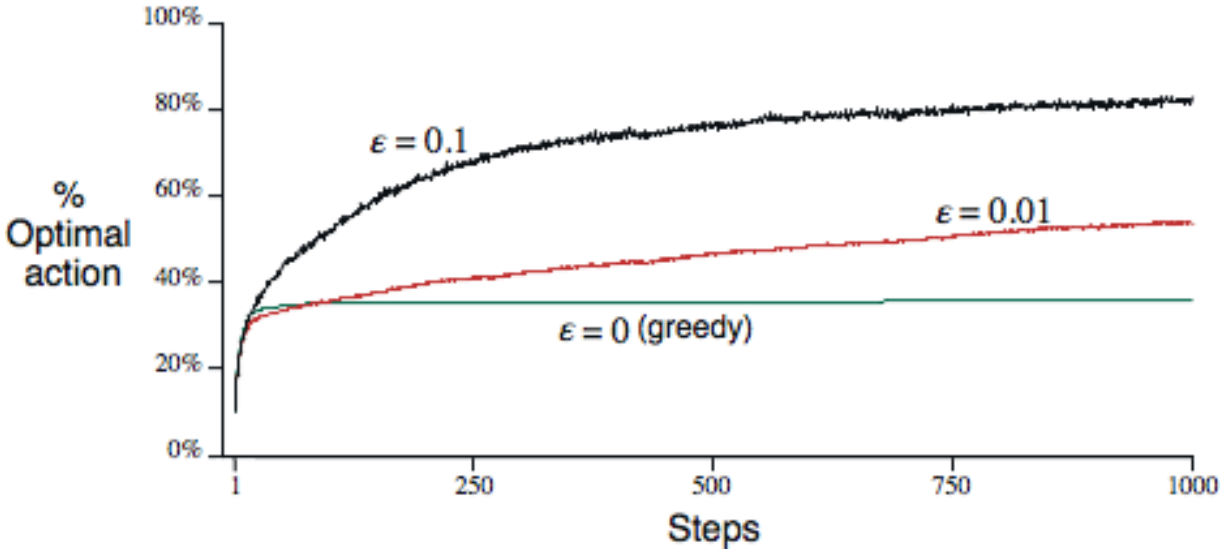
$$R_t \sim \mathcal{N}(q_*(a), 1)$$

Run for 1000 steps

Repeat the whole thing 2000 times with different bandit tasks

ε-Greedy Methods on the 10-Armed Testbed





Averaging ⟶ learning rule

- To simplify notation, let us focus on one action
  - We consider only its rewards, and its estimate after $n+1$ rewards:

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

- How can we do this incrementally (without storing all the rewards)?
- Could store a running sum and count (and divide), or equivalently:

$$Q_{n+1} = Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]$$

- This is a standard form for learning/update rules:

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \; \Big[Target - OldEstimate\Big]$$

Derivation of incremental update

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}$$

Tracking a Non-stationary Problem

- Suppose the true action values change slowly over time
  - then we say that the problem is *nonstationary*
- In this case, sample averages are not a good idea (Why?)

- Better is an "exponential, recency-weighted average":

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

$$= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

where α is a constant, *step-size parameter*, $0 < \alpha \leq 1$

- There is bias due to Q_1 that becomes smaller over time

Standard stochastic approximation convergence conditions

- To assure convergence with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

e.g., $\alpha_n = \frac{1}{n}$

not $\alpha_n = \frac{1}{n^2}$

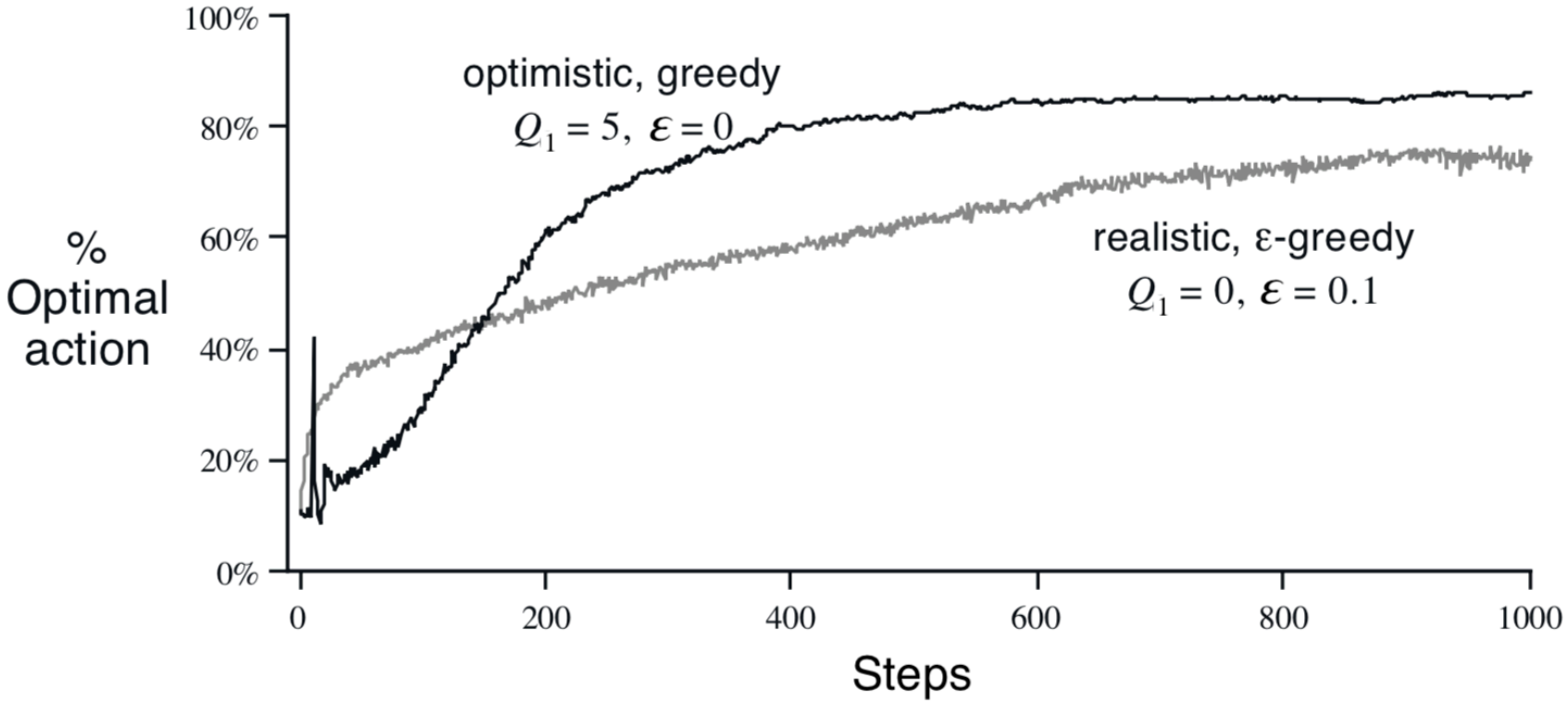
if $\alpha_n = n^{-p}, \quad p \in (0, 1)$
then convergence is at the optimal rate:

$$O(1/\sqrt{n})$$

Optimistic Initial Values

- All methods so far depend on $Q_1(a)$, i.e., they are biased.
  So far we have used $Q_1(a) = 0$

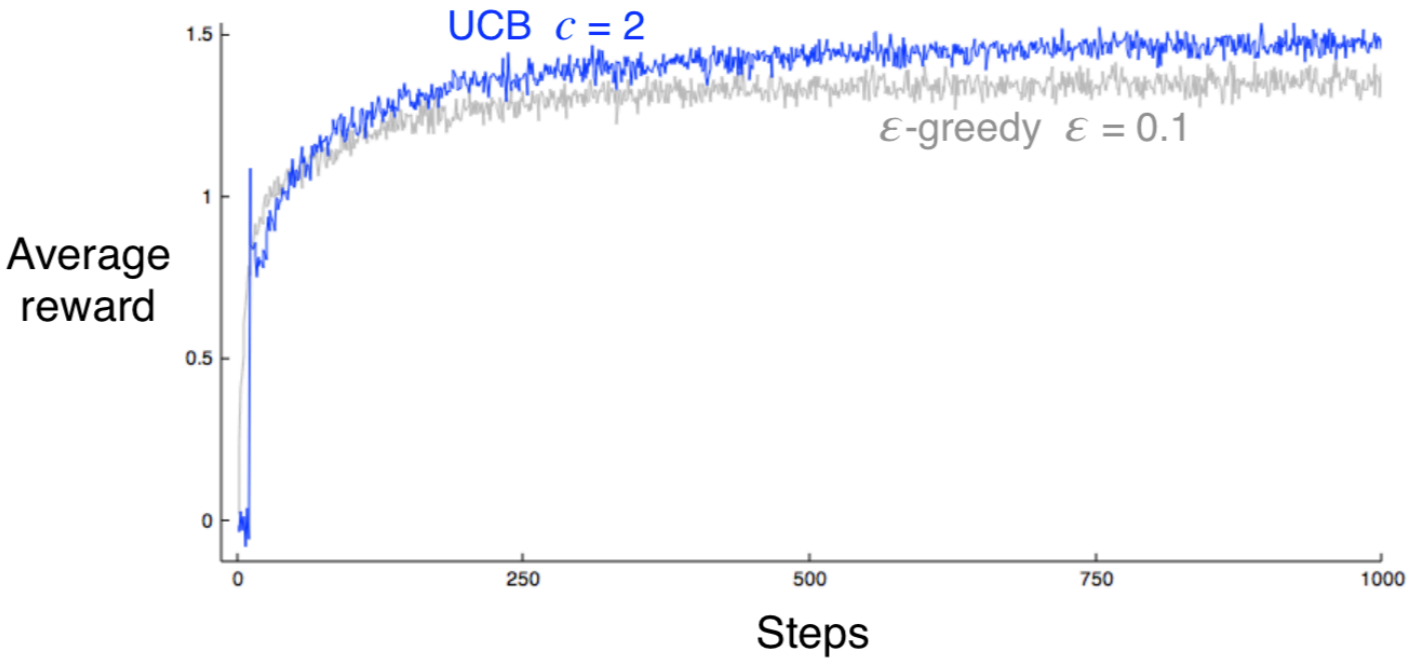
- Suppose we initialize the action values *optimistically* ($Q_1(a) = 5$),
  e.g., on the 10-armed testbed (with $\alpha = 0.1$)



Upper Confidence Bound (UCB) action selection

- A clever way of reducing exploration over time
- Estimate an upper bound on the true action values
- Select the action with the largest (estimated) upper bound

$$A_t \doteq \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$



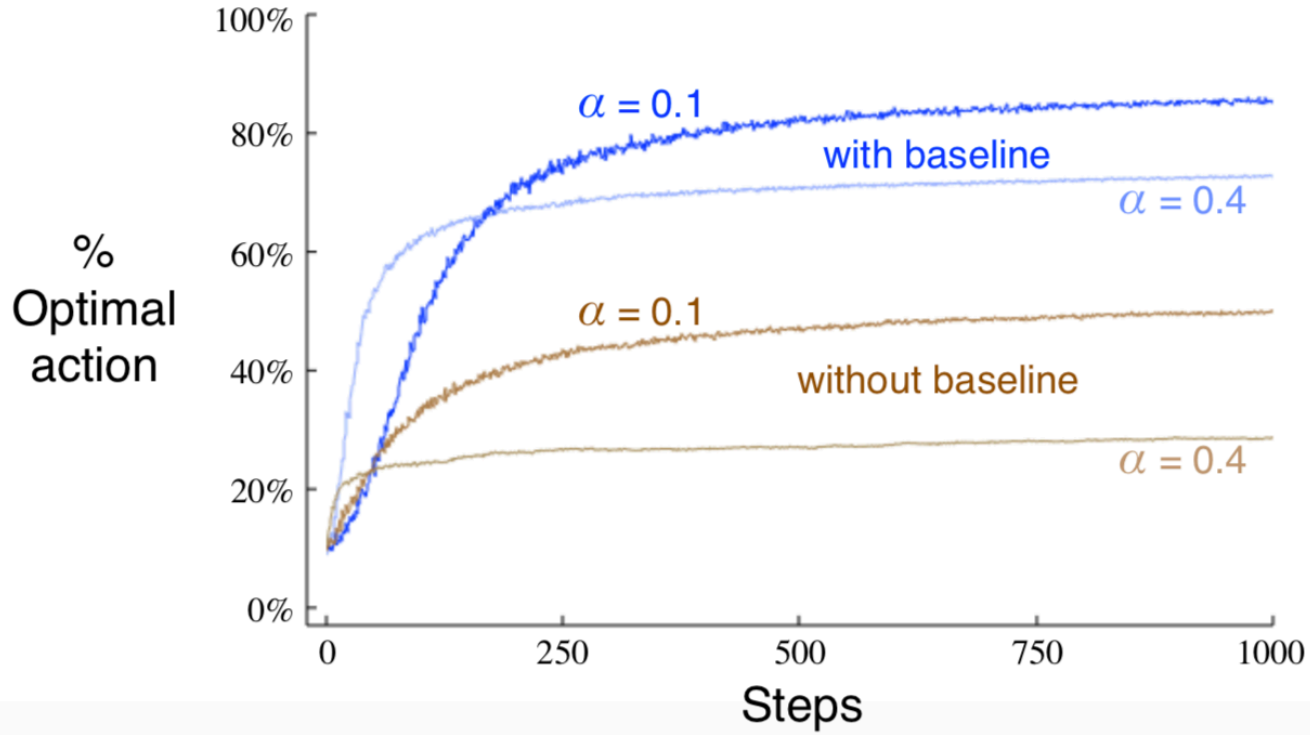
Gradient-Bandit Algorithms

- Let $H_t(a)$ be a learned *preference* for taking action a

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

$$H_{t+1}(a) \doteq H_t(a) + \alpha\Big(R_t - \bar{R}_t\Big)\Big(\mathbb{1}\{A_t = a\} - \pi_t(a)\Big), \qquad \forall a$$

$$\bar{R}_t \doteq \frac{1}{t}\sum_{i=1}^{t} R_i$$



Derivation of gradient-bandit algorithm

In exact *gradient ascent*:

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}\left[R_t\right]}{\partial H_t(a)}, \tag{1}$$

where:

$$\mathbb{E}[R_t] \doteq \sum_b \pi_t(b) q_*(b),$$

$$\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[\sum_b \pi_t(b) q_*(b) \right] \\
&= \sum_b q_*(b) \frac{\partial \pi_t(b)}{\partial H_t(a)} \\
&= \sum_b \left(q_*(b) - X_t \right) \frac{\partial \pi_t(b)}{\partial H_t(a)},
\end{aligned}$$

where X_t does not depend on b, because $\sum_b \frac{\partial \pi_t(b)}{\partial H_t(a)} = 0$.

$$
\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \sum_b \big(q_*(b) - X_t\big) \frac{\partial \pi_t(b)}{\partial H_t(a)} \\
&= \sum_b \pi_t(b) \big(q_*(b) - X_t\big) \frac{\partial \pi_t(b)}{\partial H_t(a)} / \pi_t(b) \\
&= \mathbb{E}\left[\big(q_*(A_t) - X_t\big) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right] \\
&= \mathbb{E}\left[\big(R_t - \bar{R}_t\big) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right],
\end{aligned}
$$

where here we have chosen $X_t = \bar{R}_t$ and substituted R_t for $q_*(A_t)$, which is permitted because $\mathbb{E}[R_t|A_t] = q_*(A_t)$.
For now assume: $\frac{\partial \pi_t(b)}{\partial H_t(a)} = \pi_t(b)\big(\mathbf{1}_{a=b} - \pi_t(a)\big)$. Then:

$$
\begin{aligned}
&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big) \pi_t(A_t) \big(\mathbf{1}_{a=A_t} - \pi_t(a)\big) / \pi_t(A_t) \big] \\
&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big) \big(\mathbf{1}_{a=A_t} - \pi_t(a)\big) \big] .
\end{aligned}
$$

$$
H_{t+1}(a) = H_t(a) + \alpha\big(R_t - \bar{R}_t\big)\big(\mathbf{1}_{a=A_t} - \pi_t(a)\big), \text{ (from (1), QED)}
$$

Thus it remains only to show that

$$\frac{\partial \pi_t(b)}{\partial H_t(a)} = \pi_t(b)\big(\mathbf{1}_{a=b} - \pi_t(a)\big).$$

Recall the standard quotient rule for derivatives:

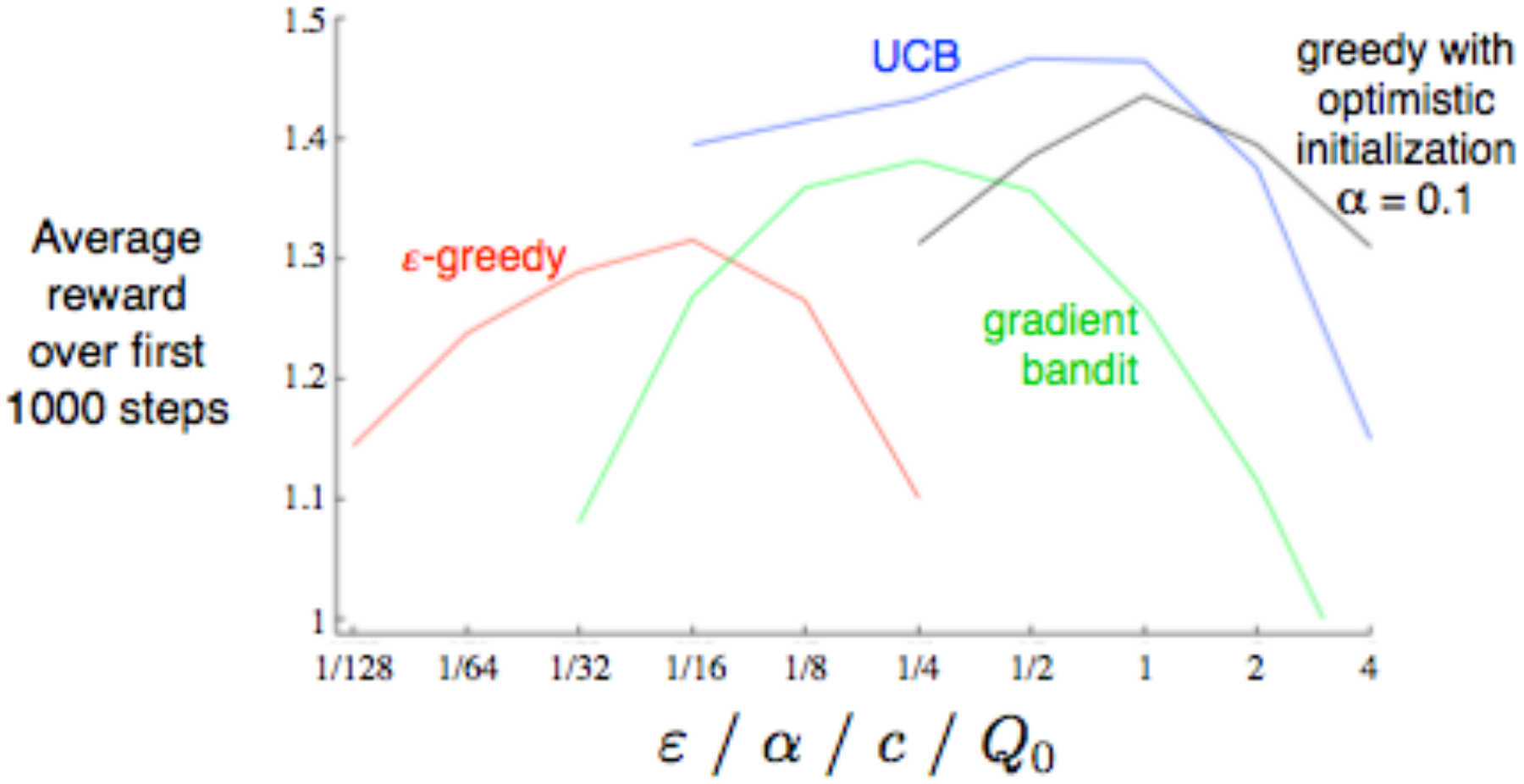
$$\frac{\partial}{\partial x}\left[\frac{f(x)}{g(x)}\right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x)\frac{\partial g(x)}{\partial x}}{g(x)^2}.$$

Using this, we can write...

Quotient Rule: $\frac{\partial}{\partial x}\left[\frac{f(x)}{g(x)}\right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x)\frac{\partial g(x)}{\partial x}}{g(x)^2}$

$$
\begin{aligned}
\frac{\partial \pi_t(b)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \pi_t(b) \\
&= \frac{\partial}{\partial H_t(a)} \left[\frac{e^{h_t(b)}}{\sum_{c=1}^{k} e^{h_t(c)}} \right] \\
&= \frac{\frac{\partial e^{h_t(b)}}{\partial H_t(a)} \sum_{c=1}^{k} e^{h_t(c)} - e^{h_t(b)} \frac{\partial \sum_{c=1}^{k} e^{h_t(c)}}{\partial H_t(a)}}{\left(\sum_{c=1}^{k} e^{h_t(c)} \right)^2} && \text{(Q.R.)} \\
&= \frac{\mathbf{1}_{a=b} e^{h_t(a)} \sum_{c=1}^{k} e^{h_t(c)} - e^{h_t(b)} e^{h_t(a)}}{\left(\sum_{c=1}^{k} e^{h_t(c)} \right)^2} && \left(\frac{\partial e^x}{\partial x} = e^x\right) \\
&= \frac{\mathbf{1}_{a=b} e^{h_t(b)}}{\sum_{c=1}^{k} e^{h_t(c)}} - \frac{e^{h_t(b)} e^{h_t(a)}}{\left(\sum_{c=1}^{k} e^{h_t(c)} \right)^2} \\
&= \mathbf{1}_{a=b} \pi_t(b) - \pi_t(b)\pi_t(a) \\
&= \pi_t(b)\left(\mathbf{1}_{a=b} - \pi_t(a)\right). && \text{(Q.E.D.)}
\end{aligned}
$$

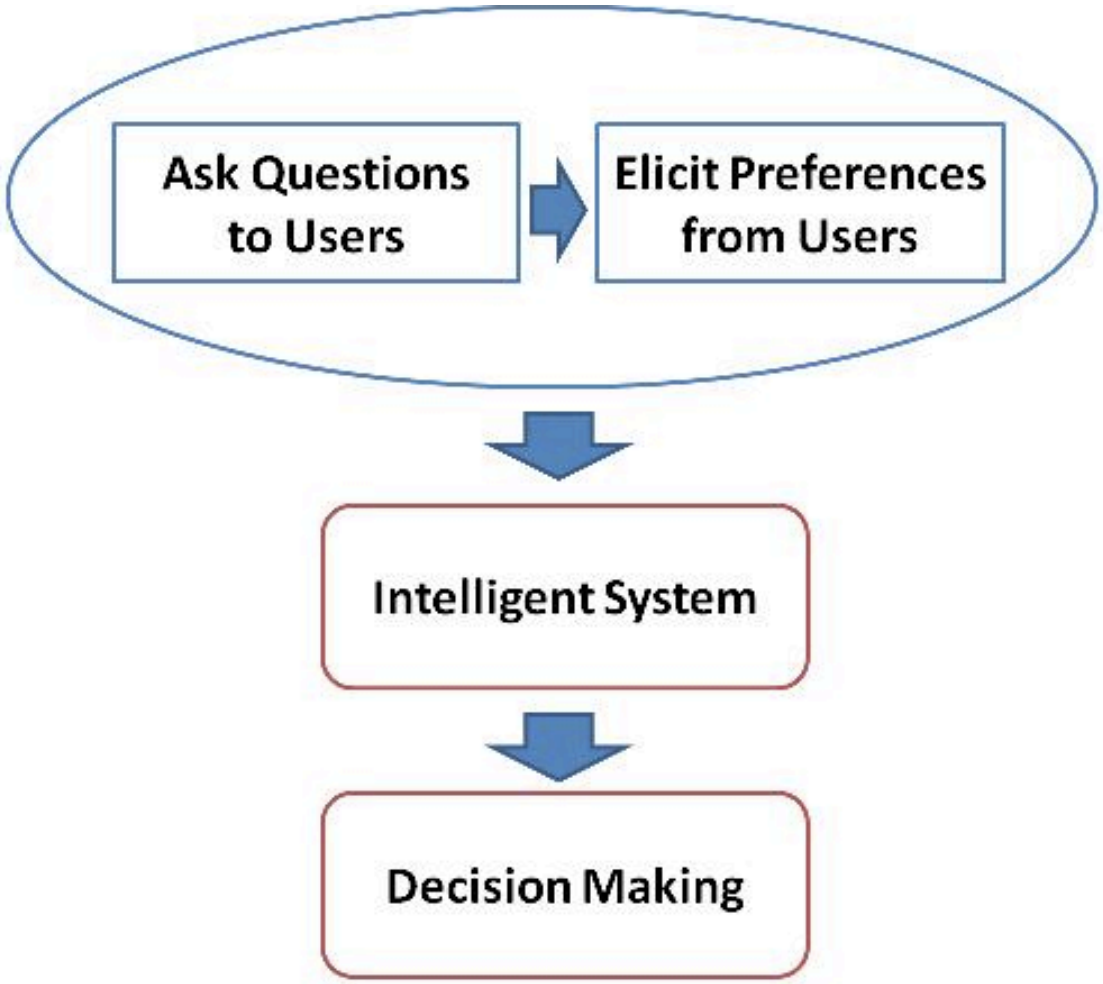
Summary Comparison of Bandit Algorithms



Conclusions

- These are all simple methods
  - but they are complicated enough—we will build on them
  - we should understand them completely
  - there are still open questions
- Most results in the bandit literature are focused on theoretical properties (e.g. bound on # samples necessary to learn true reward function with small error margin)

Related area: Preference elicitation

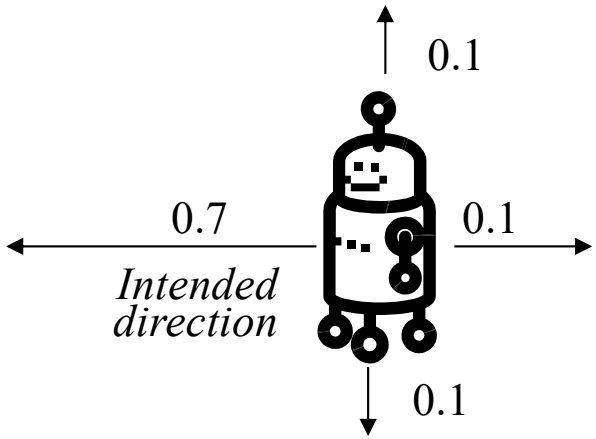


- Deals with very large numbers of users and items.
- Assume for each user we know preference for a small number of items.
- Goal is to infer the value of other unscored items.
- Methods based on matrix completion.

Picture from: *http://students.washington.edu/hyunggu/cs886pe.html*

From single-state problems:

To multiple states:



S			+1
			-10

Pause

Markov Decision Process (MDP)

Agent and environment interact at discrete time steps: $t = 0, 1, 2, \mathrm{K}$

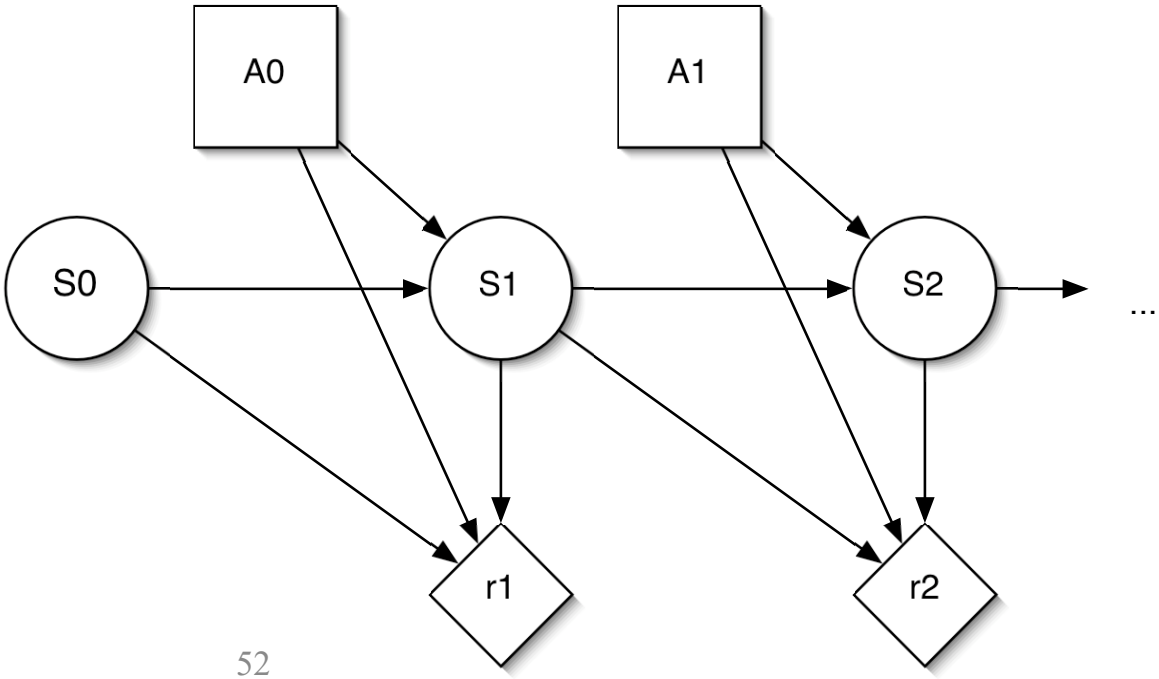
Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t : $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$

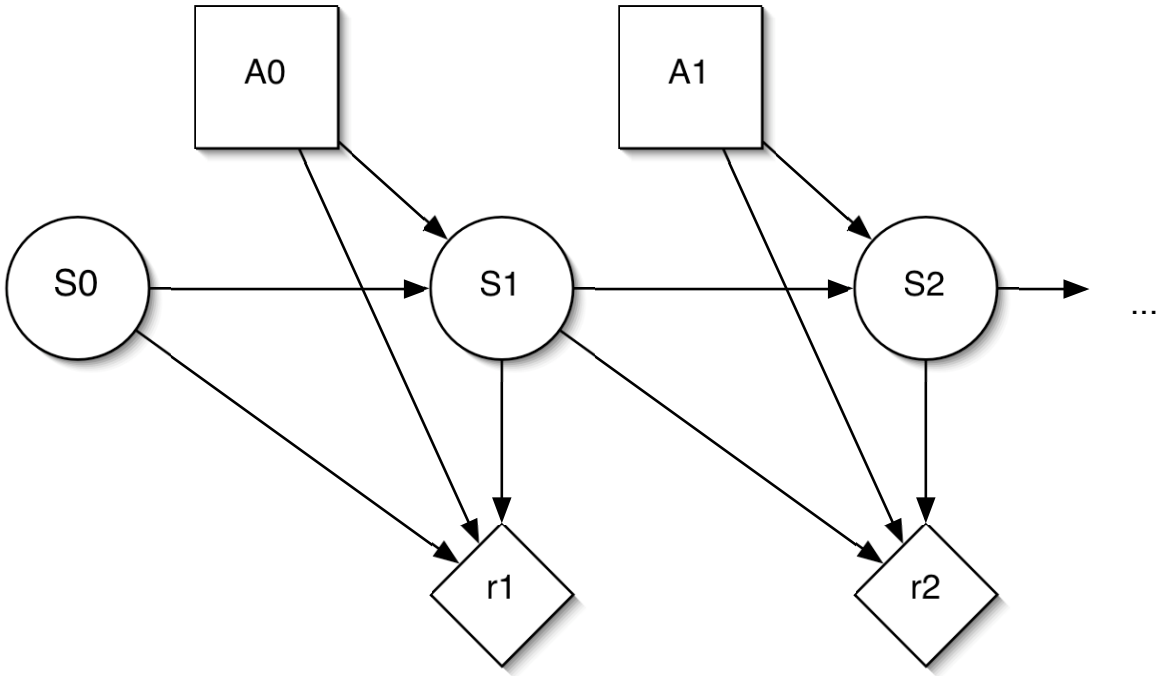
Also require the transition function: T(s,a,s') = Pr(s' | s,a)



The **Markov** property

The distribution over future states **depends only on the present state**, not on any previous events.

$$Pr(s_t \mid s_{t-1}, \ldots, s_0) = Pr(s_t \mid s_{t-1})$$



The **Markov** property

- Traffic lights?

- Chess?

- Poker?

Maximizing utility

- Define: G_t , the utility for a trajectory, starting from step t.

- Episodic tasks : Tasks with an end (e.g. games, mazes, etc.)

$$G_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

Maximizing utility

- Define: G_t , the utility for a trajectory, starting from step t.

- Episodic tasks : Tasks with an end (e.g. games, mazes, etc.)

$$G_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

- Continuing tasks : Tasks that go on forever (e.g. juggling, balancing)

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \ldots = \sum_{k=0:\infty} \gamma^k r_{t+k+1}$$

The discount factor, γ

- Discount factor, $\gamma \in [0, 1)$ (usually close to 1).

- Intuition:

  - Receiving \$900 today is worth the same as \$1000 tomorrow (assuming a discount factor of factor of $\gamma = 0.9$).

  - At each time step, there is a $1-\gamma$ chance that the agent dies, and does not receive rewards afterwards.

The policy

A policy defines the action-selection strategy at every state:

$$\pi(s,a) = P(a_t=a \mid s_t=s)$$

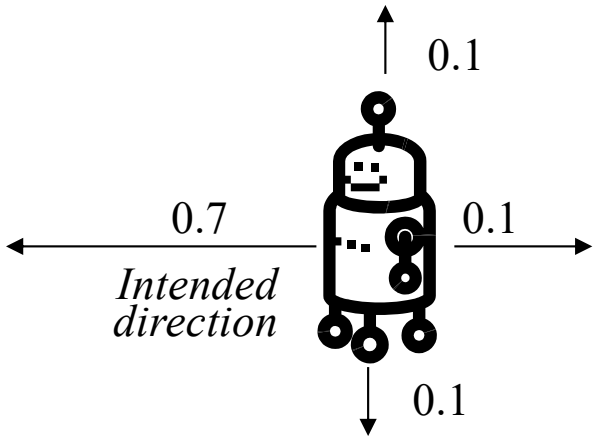
(Can be stochastic as above, or deterministic, $S \rightarrow A$.)

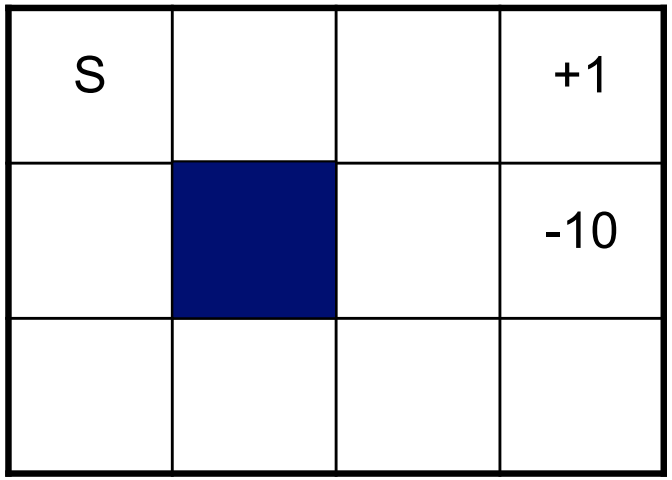
Goal: **Find the policy that maximizes expected total reward.**
(But there are many policies!)

$$\mathbf{argmax}_\pi \, \mathbf{E}_\pi [\, r_1 + \ldots + r_T \mid s_0 \,]$$

Quick problem

- Define:
  - States
  - Actions
  - Rewards
  - Policy





Value functions

Goal: **Find a policy** that maximizes the expected return.

The **expected return of a policy** (for every state) is called the **value function**: $V^{\pi}(s) = E_{\pi}[r_{t+1} + r_{t+2} + \ldots + r_T \mid s_t = s]$

How?

Value functions

Goal: **Find a policy** that maximizes the expected return.

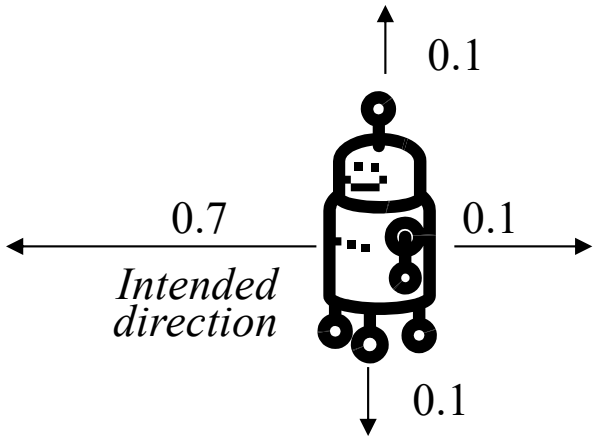
The **expected return of a policy** (for every state) is called the **value function**: $V^{\pi}(s) = E_{\pi}[r_{t+1} + r_{t+2} + \ldots + r_T \mid s_t = s]$

Simple strategy:

1. Enumerate the space of all possible policies.
2. Estimate the expected return of each one.
3. Keep the policy that has maximum expected return.

Quick problem

- What is the space of all policies?



S			+1
			-10

Getting confused with terminology?

- **Reward**?
- **Return**?
- **Value**?
- **Utility**?

Getting confused with terminology?

- **Reward**: 1 step numerical feedback
- **Return**: Sum of rewards over the agent's trajectory.
- **Value**: Expected sum of rewards over the agent's trajectory.
- **Utility**: Numerical function representing preferences.

- In RL, we assume **Utility** = **Value**.

The value of a policy

$$V^{\pi}(s) = E_{\pi}[r_{t+1} + r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = E_{\pi}[r_{t+1}] + E_{\pi}[r_{t+2} + \ldots + r_T \mid s_t = s]$$

The value of a policy

$$V^{\pi}(s) = E_{\pi}[r_{t+1} + r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = E_{\pi}[r_{t+1}] + E_{\pi}[r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = \underbrace{\sum_{a \in A} \pi(s,a) R(s,a)}_{\text{Immediate reward}} + \underbrace{E_{\pi}[r_{t+2} + \ldots + r_T \mid s_t = s]}_{\text{Future expected sum of rewards}}$$

The value of a policy

$$V^{\pi}(s) = E_{\pi}[r_{t+1} + r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = E_{\pi}[r_{t+1}] + E_{\pi}[r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a)R(s,a) + E_{\pi}[r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a)R(s,a) + \underbrace{\sum_{a \in A} \pi(s,a)\sum_{s' \in S} T(s,a,s')}_{\text{Expectation over 1-step transition}} E_{\pi}[r_{t+2} + \ldots + r_T \mid s_{t+1} = s']$$

The value of a policy

$$V^{\pi}(s) = E_{\pi}[r_{t+1} + r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = E_{\pi}[r_{t+1}] + E_{\pi}[r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) R(s,a) + E_{\pi}[r_{t+2} + \ldots + r_T \mid s_t = s]$$

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) R(s,a) + \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s') E_{\pi}[r_{t+2} + \ldots + r_T \mid s_{t+1} = s']$$

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) R(s,a) + \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s') V^{\pi}(s')$$

By definition

This is a **dynamic programming** algorithm.

The value of a policy

State value function (for a **fixed** policy):

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \left[\underbrace{R(s,a)}_{\text{Immediate}} + \gamma \underbrace{\sum_{s' \in S} T(s,a,s') V^{\pi}(s')}_{\text{Future expected sum of rewards}} \right]$$

State-action value function:

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'} T(s, a, s') \, Q^{\pi}(s', \pi(s'))$$

These are (two forms of) **Bellman's equation**.

The value of a policy

State value function:

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi}(s') \right)$$

When S is a **finite set of states**, this is a **system of linear equations** (one per state) with a unique solution V^{π}.

Bellman's equation in matrix form: $V^{\pi} = R^{\pi} + \gamma T^{\pi} V^{\pi}$

Which can solved exactly: $V^{\pi} = (I - \gamma T^{\pi})^{-1} R^{\pi}$

Quick problem

- Can you write down the transition matrix for the policy $\pi(s)=Search,\ \forall s$
- Can you write the value of this policy using a system of linear eqns?

Recycling Robot MDP

$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$

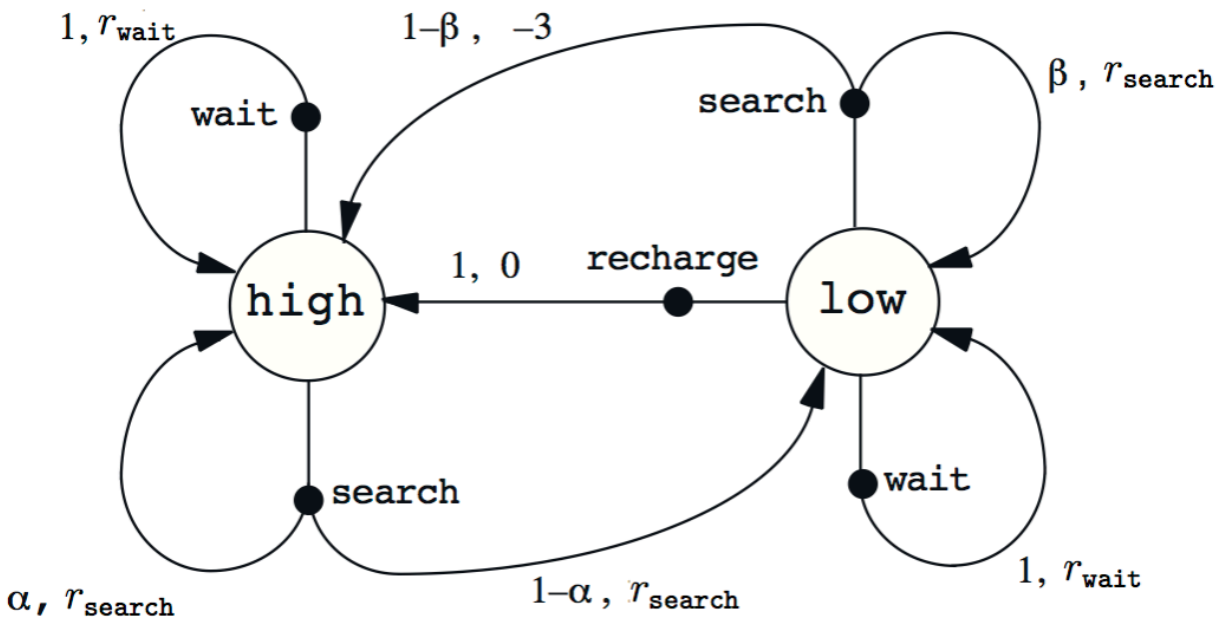
$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$r_{\texttt{search}} > r_{\texttt{wait}}$



Iterative Policy Evaluation

Main idea: turn Bellman equations into update rules.

1. Start with some initial guess $V_0(s), \forall s.$ (Can be 0, or $r(s,\cdot)$.)

Iterative Policy Evaluation

Main idea: turn Bellman equations into update rules.

1. Start with some initial guess $V_0(s), \forall s.$ (Can be 0, or $r(s, \cdot)$.)

2. During every iteration k, update the value function for all states:

$$V_{k+1}(s) \leftarrow \Big(R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_k(s') \Big)$$

Iterative Policy Evaluation

Main idea: turn Bellman equations into update rules.

1. Start with some initial guess $V_0(s), \forall s.$ (Can be 0, or $r(s, \cdot)$.)

2. During every iteration k, update the value function for all states:

$$V_{k+1}(s) \leftarrow \Big(R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_k(s') \Big)$$

3. Stop when the maximum changes between two iterations is smaller than a desired threshold (the values stop changing.)

Convergence of Iterative Policy Evaluation

- Consider the absolute error in our estimate $V_{k+1}(s)$:

$$
\begin{aligned}
|V_{k+1}(s) - V^{\pi}(s)| &= \left| \sum_a \pi(s,a)(R(s,a) + \gamma \sum_{s'} T(s,a,s') V_k(s')) \right. \\
&\quad \left. - \sum_a \pi(s,a)(R(s,a) + \gamma \sum_{s'} T(s,a,s') V^{\pi}(s')) \right| \\
&= \gamma \left| \sum_a \pi(s,a) \sum_{s'} T(s,a,s')(V_k(s') - V^{\pi}(s')) \right| \\
&\leq \gamma \sum_a \pi(s,a) \sum_{s'} T(s,a,s') |V_k(s') - V^{\pi}(s')|
\end{aligned}
$$

- As long as $\gamma<1$, the error **contracts** and eventually goes to 0.

Quick problem

- Can you find the value of the policy $\pi(s)=Search$ using the iterative method? Assume $V_0(s)=0, \forall s$.

Recycling Robot MDP

$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$

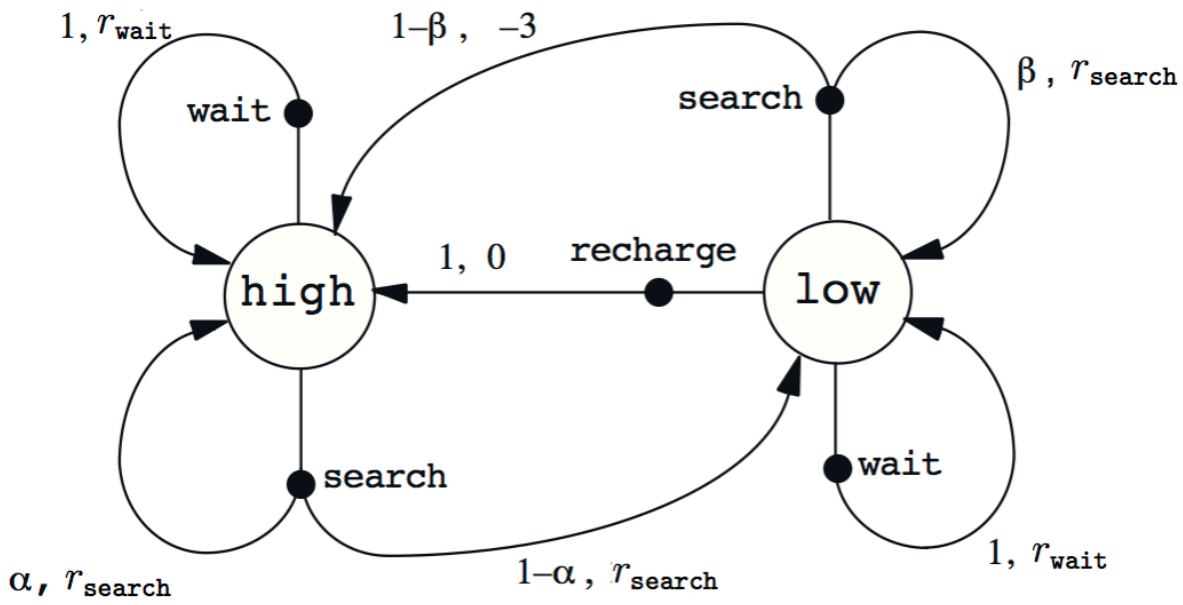
$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$r_{\texttt{search}} > r_{\texttt{wait}}$



Assume

$\gamma = 0.99$

$\alpha = 0.75$

$\beta = 1$

$r_{search} = 5$

$r_{wait} = 1$

Optimal policies and optimal value functions

- The **optimal value function** V^* is defined as the best value that can be achieved at any state:

$$V^*(s) = \max_\pi V^\pi(s)$$

- Any policy that achieves the optimal value function is called an **optimal policy**, denoted π^*.

- There exists a unique optimal **value function** *(Bellman, 1957)*.
- The optimal policy is not necessarily unique.

Optimal policies and optimal value functions

- If we know V^* (and R, T, γ), then we can compute π^* easily:

$$\pi^*(s) = \text{argmax}_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right)$$

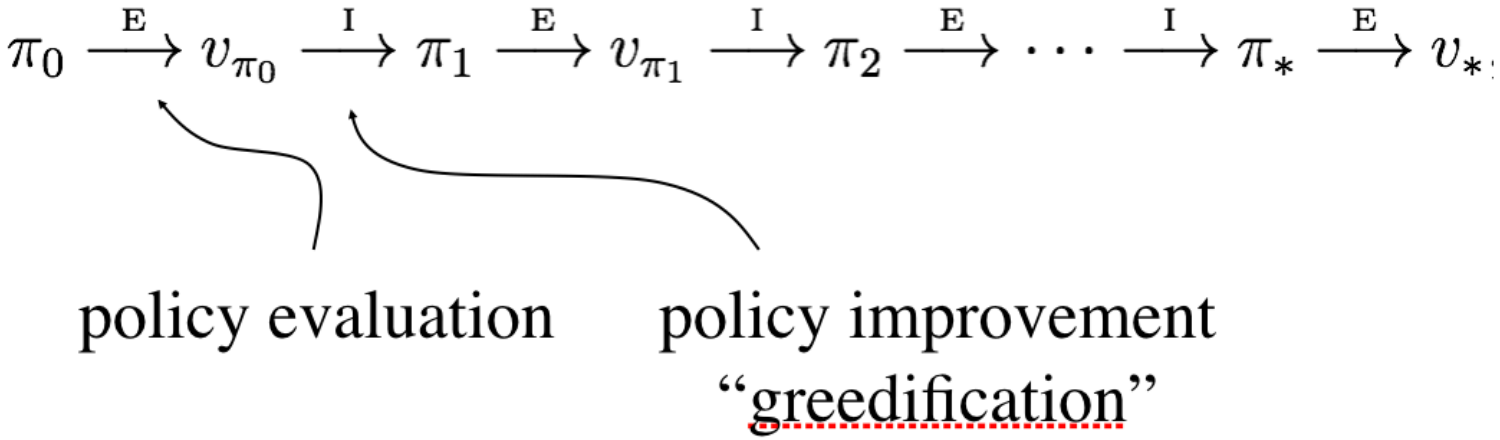
- If we know π^* (and R, T, γ), then we can compute V^* easily:

$$V^*(s) = \sum_{a \in A} \pi^*(s,a) \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right)$$

$$V^*(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^*(s')$$

Use policy evaluation to get V^*

Finding a good policy: **Policy Iteration**



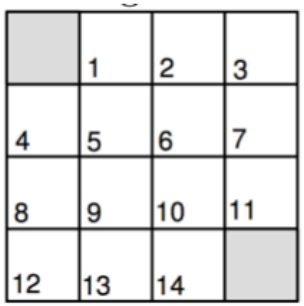
Policy iteration algorithm:

- Start with an initial policy π_0 (e.g. random)
- Repeat:
  - Compute V^{π}, using policy evaluation.
  - Compute a new policy π' that is greedy with respect to V^{π}
- Terminate when $\pi = \pi'$

Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices

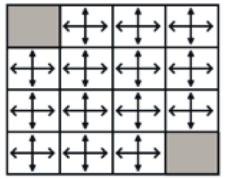
actions



$R = -1$
on all transitions

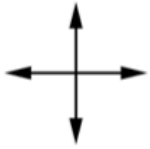
$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

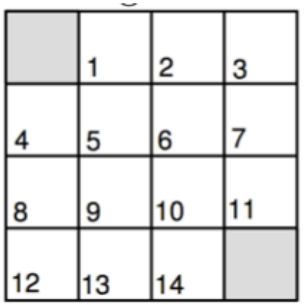
	V_k for the Random Policy	Greedy Policy w.r.t. V_k
$k = 0$	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	 random policy
$k = 1$	0.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 0.0	
$k = 2$	0.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 0.0	
$k = 3$	0.0 -2.4 -2.9 -3.0 -2.4 -2.9 -3.0 -2.9 -2.9 -3.0 -2.9 -2.4 -3.0 -2.9 -2.4 0.0	
$k = 10$	0.0 -6.1 -8.4 -9.0 -6.1 -7.7 -8.4 -8.4 -8.4 -8.4 -7.7 -6.1 -9.0 -8.4 -6.1 0.0	
$k = \infty$	0.0 -14. -20. -22. -14. -18. -20. -20. -20. -20. -18. -14. -22. -20. -14. 0.0	

Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices



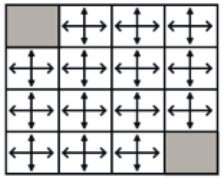
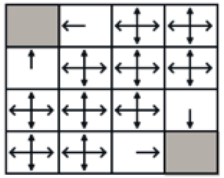
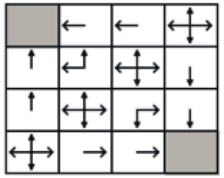
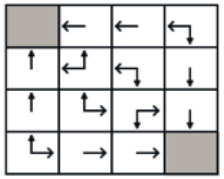
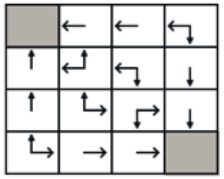
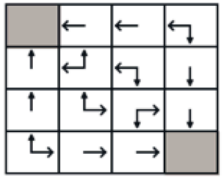
actions



$R = -1$
on all transitions

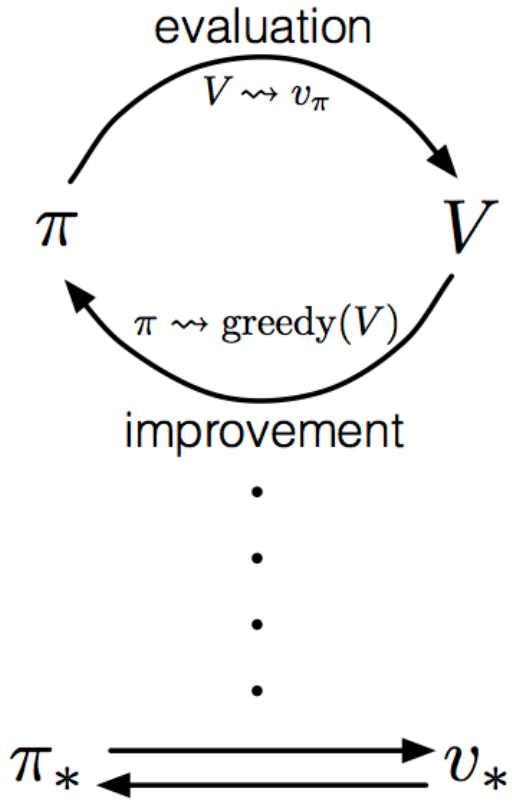
$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

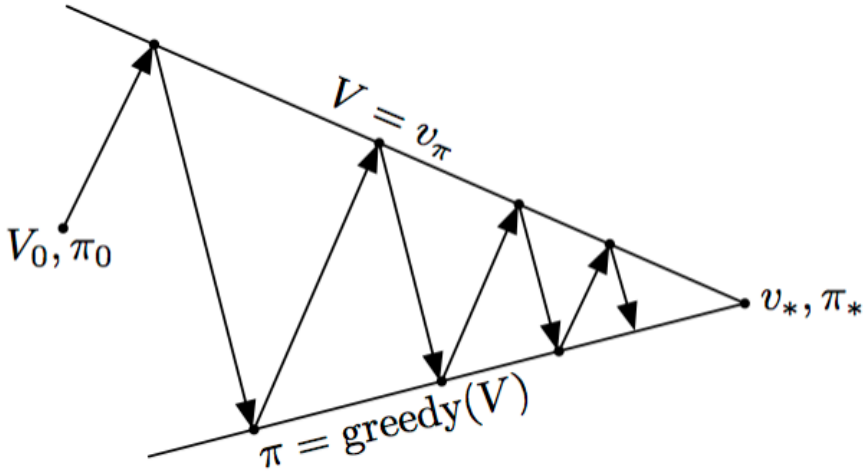
	V_k for the Random Policy	Greedy Policy w.r.t. V_k
$k = 0$	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	 ← random policy
$k = 1$	0.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 0.0	
$k = 2$	0.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 0.0	
$k = 3$	0.0 -2.4 -2.9 -3.0 -2.4 -2.9 -3.0 -2.9 -2.9 -3.0 -2.9 -2.4 -3.0 -2.9 -2.4 0.0	
$k = 10$	0.0 -6.1 -8.4 -9.0 -6.1 -7.7 -8.4 -8.4 -8.4 -8.4 -7.7 -6.1 -9.0 -8.4 -6.1 0.0	 ← optimal policy
$k = \infty$	0.0 -14. -20. -22. -14. -18. -20. -20. -20. -20. -18. -14. -22. -20. -14. 0.0	

Generalized policy iteration (GPI)

any interaction of policy evaluation and policy improvement, independent of their granularity.



A geometric metaphor for convergence of GPI:



Finding a good policy: **Value iteration**

Main idea: Turn the Bellman optimality equation into an iterative update rule (same as done in policy evaluation)

- Start with an arbitrary initial approximation $V_0(s)$
- Repeat:
  - Update the value function estimate:

    $V_k(s) = max_{a \in A} (R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{k-1}(s'))$
- Terminate the Bellman residual is below some desired threshold.

  $| V_k(s) - V_{k-1}(s) | < \varepsilon$

The algorithm converges (in the limit) to the true V^*.

Quick problem

- Can you find the optimal value using the value iteration method?
  - Assume $V_0(s)=0, \forall s$.

Recycling Robot MDP

$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$

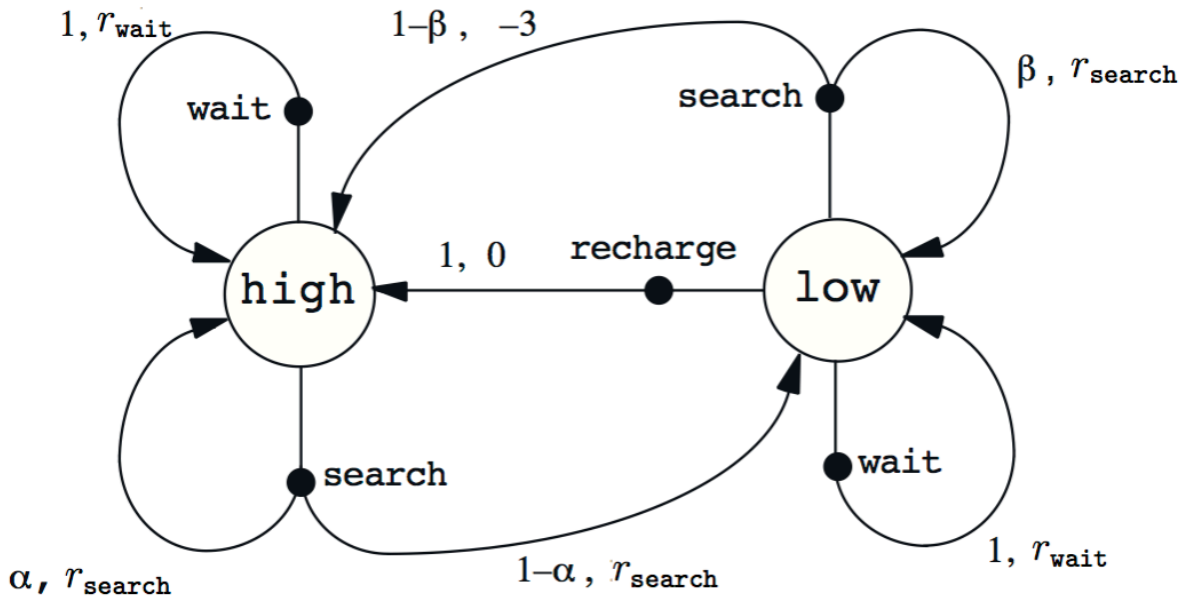
$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$r_{\texttt{search}} > r_{\texttt{wait}}$



Assume

$\gamma = 0.99$

$\alpha = 0.75$

$\beta = 1$

$r_{search} = 5$

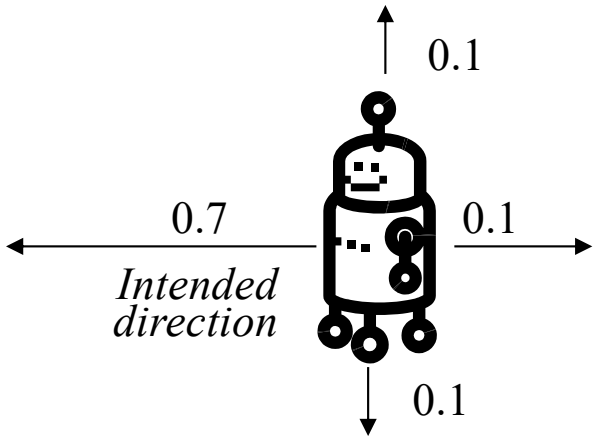
$r_{wait} = 1$

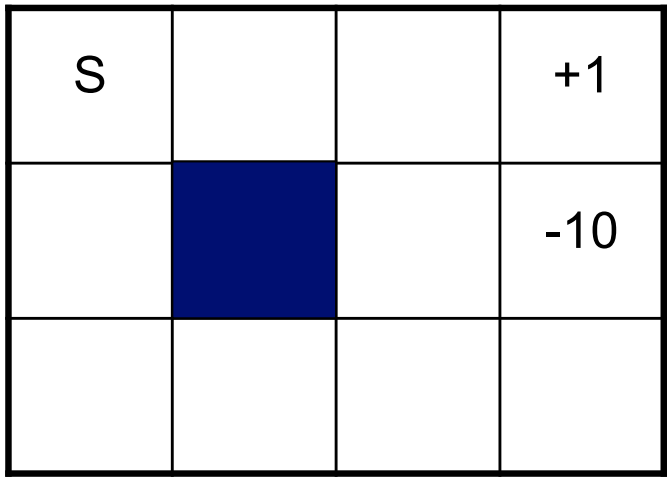
Getting confused?

- Policy evaluation
- Policy iteration
- Value iteration

Try this example: 4x3 gridworld

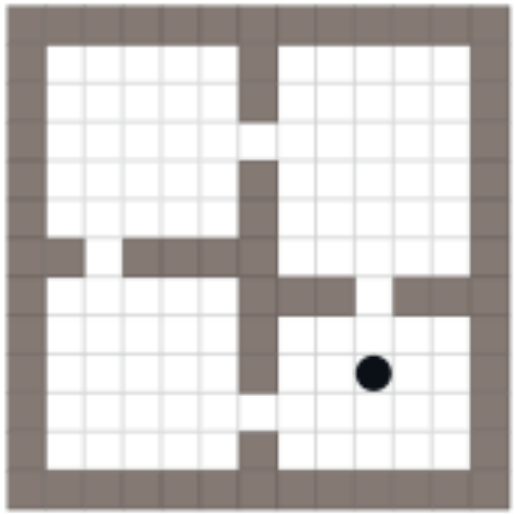
- 11 discrete states, 4 motion actions (N, S, E, W) in each state.
- Transitions are mildly **stochastic**.
- Reward is +1 in top right state, -10 in state directly below, -0 elsewhere.
- Episode terminates when the agent reaches +1 or -10 state.
- Discount factor $\gamma = 0.99$.



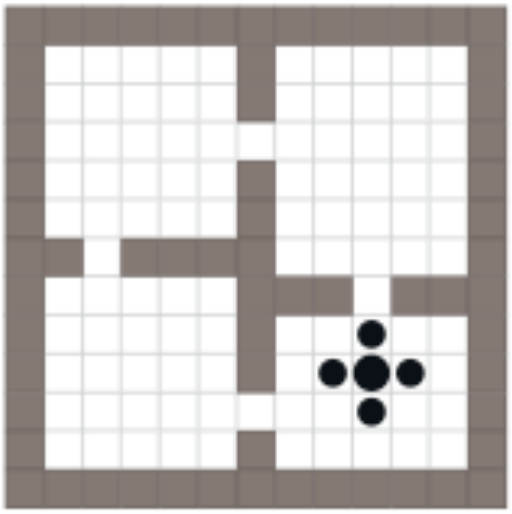


Another example: Four Rooms

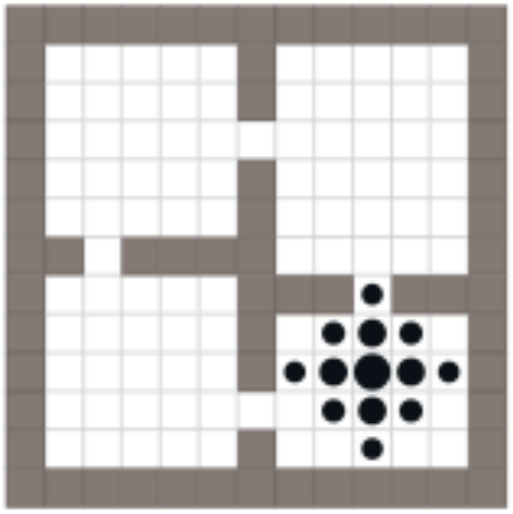
- Four actions, fail 30% of the time.
- No rewards until the goal is reached, $\gamma = 0.9$.
- Values propagate backwards from the goal.



Iteration #1



Iteration #2



Iteration #3

Complexity analysis

- Policy iteration:
  - Per iteration: $O(S^3 + S^2A)$
  - N^0 iteration: At most $|A|^{|S|}$

- Value iteration??

Complexity analysis

- Policy iteration:
  - Per iteration: $O(S^3 + S^2A)$
  - N^0 iteration: At most $|A|^{|S|}$
  - ✓ Fewer iterations

- Value iteration:
  - Per iteration: $O(S^2A)$ (down to $O(SA)$ if few successor states)
  - N^o iterations: Polynomial in $1 / (1 - \gamma)$
  - ✓ Faster per iteration

Alternative: Asynchronous value iteration

- Instead of updating all states on every iteration, focus on *important states*.
  - E.g., board positions that occur on every game, rather than just once in 100 games.

- Asynchronous dynamic programming algorithm:
  - Generate trajectories through the MDP.
  - Update states whenever they appear on such a trajectory.

- Focuses the updates on states that are actually possible.

Limitations of MDPs

1. Finding an optimal policy is polynomial in the number of states.
   - Number of states is often astronomical.
   - Dynamic programming can solve problems up to 10^7 states.

2. Value iteration and policy iteration assume the model (transitions and rewards) is known in advance.

3. State is sometimes not observable (similar to HMM case).
   - Some states may “look the same”.
   - Sensors data may be noisy.