Lecture 4: Reinforcement learning

Viliam Lisý & Branislav Bošanský

Artificial Intelligence Center
Department of Computer Science, Faculty of Electrical Eng.
Czech Technical University in Prague

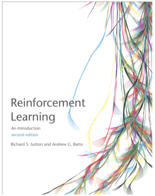
viliam.lisy@fel.cvut.cz

March, 2024

Definition

Wikipedia: Reinforcement learning is "concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward"

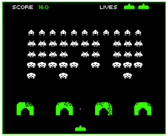
The book: "Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal."

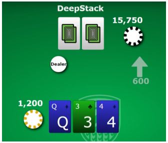


Me: Learning to choose actions to optimize rewards based on experience – trial and errors.

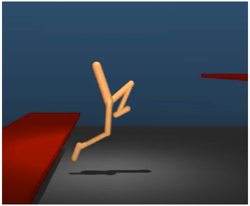
Motivation

Success stories:





ChatGPT

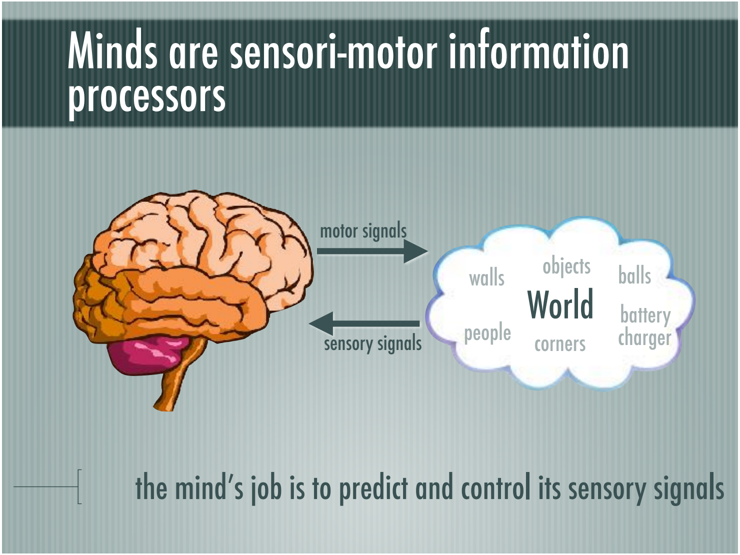


It can solve a diverse set of problems!

Why is most of this in simutlations?

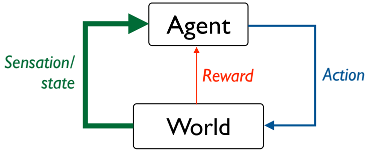
RL currently needs a huge amount of experience, which is easier to obtain in simualtion

Motivation



Taken from R. Sutton's slides.

Reinforcement learning
is *more autonomous learning*

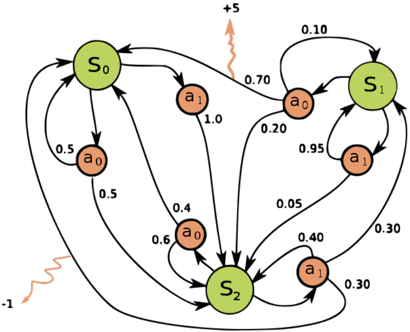


- Learning that requires less input from people
- AI that can learn for itself, during its normal operation

Taken from R. Sutton's slides (and many following are adaptations as well).

Remember MDP

Standard model for Reinforcement Learning problems



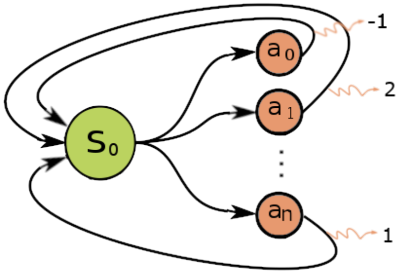
- S – states
- R – rewards
- A – actions
- Discrete steps $t = 0, 1, 2, \ldots$
- Environment *dynamics*

Source: Waldoalvarez @ wikimedia

$$p(s', r|s, a) \leftarrow Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

Single state MDP: Multi-armed Bandit Problem

All actions $a_1, \ldots, a_n$ lead back to the single state of MDP.



A simple case with many of the RL's fundamental problems.

utility estimation, exploration-exploitation, (non-stationarity)

Why is it called Multi-Armed Bandit Problem

Example problem

Action 1: Reward is always 8

Expected reward: $q_*(1) = 8$

Action 2: 88% chance of 0, 12% chance of 100

Expected reward: $q_*(2) = 12$

Action 3: Uniformly random between -10 and 35

Expected reward: $q_*(3) = 12.5$

Action 4: a third 0, a third 20, and a third from 8-18

Expected reward: $q_*(4) = 13/3 + 20/3 = 11$

Multi-armed Bandit Problem

On each of a sequence of time steps, $t = 1, 2, \ldots, T$ 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 indentically, independently distributed (i.i.d.):

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

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

$$Q_t(a) \approx q_*(a), \forall a \quad \text{action-value estimates}$$

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

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

Where $N_t(a)$ is 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

Algorithm ϵ-Greedy:

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

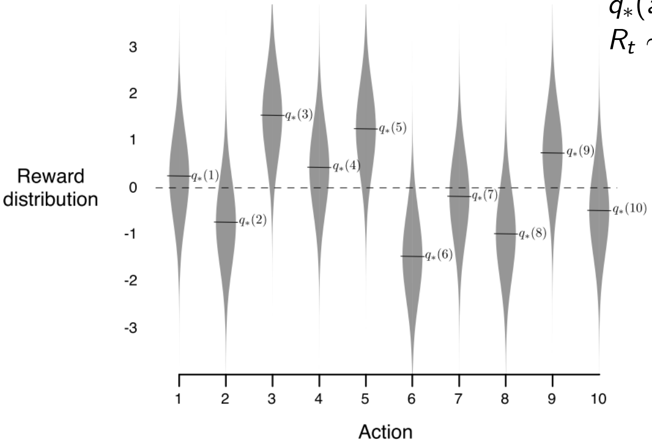
$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

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

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

One Task from the 10-armed Testbed

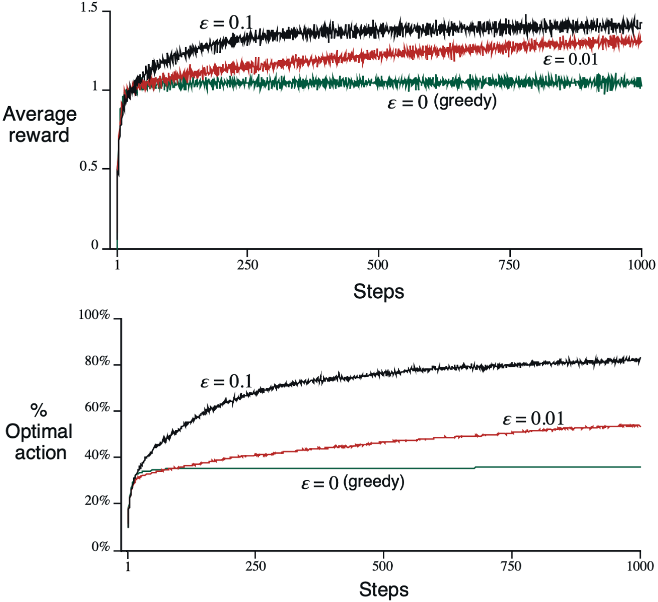


$$q_*(a) \sim \mathcal{N}(0, 1)$$
$$R_t \sim \mathcal{N}(q_*(a), 1)$$

Run for 1000 steps

Repeat the whole thing 2000 times with different bandit tasks

ϵ-Greedy Methods on the 10-Armed Testbed



Averaging $\rightarrow$ Learning Rule

To simplify notation, let us focus on one action

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

How can we do this incrementally (without storing all the rewards)?

Could store a running sum and count (and divide), or equivalently:

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

This is a standard form for learning/update rules:

$$\mathit{NewEstimate} \leftarrow \mathit{OldEstimate} + \mathit{StepSize}\left[\mathit{Target} - \mathit{OldEstimate}\right]$$

Derivation of incremental update

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

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

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., $\alpha_n \doteq \frac{1}{n}$
not $\alpha_n \doteq \frac{1}{n^2}$

if $\alpha_n \doteq n^{-p}, p \in (0.5, 1]$ then convergence is at the optimal rate $O(1/\sqrt{n})$

Tracking a Non-stationary Problem

Suppose the true action values change (slowly) over time then we say that the problem is **nonstationary** (not i.i.d.)

In this case, sample averages are not a good idea (Why?)

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

$$
\begin{aligned}
Q_{n+1} &\doteq Q_n + \alpha \left[R_n - Q_n \right] \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i,
\end{aligned}
$$

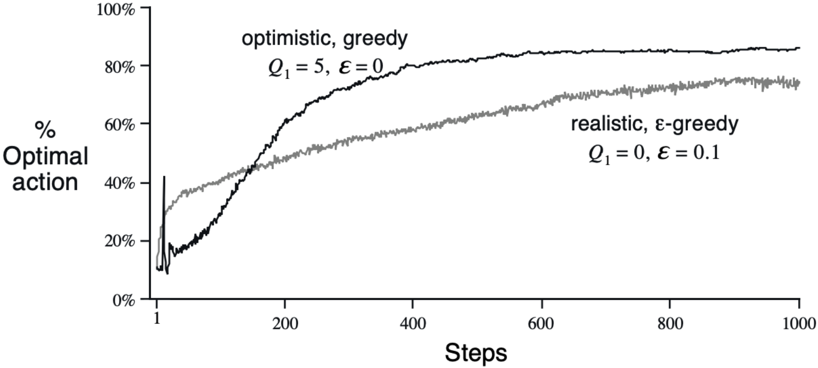
where α is a constant step-size parameter, $\alpha \in (0, 1]$

There is bias due to Q_1 that becomes smaller over time

Optimistic Initial Values

The estimates 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** ($Q_1(a) = 5$),



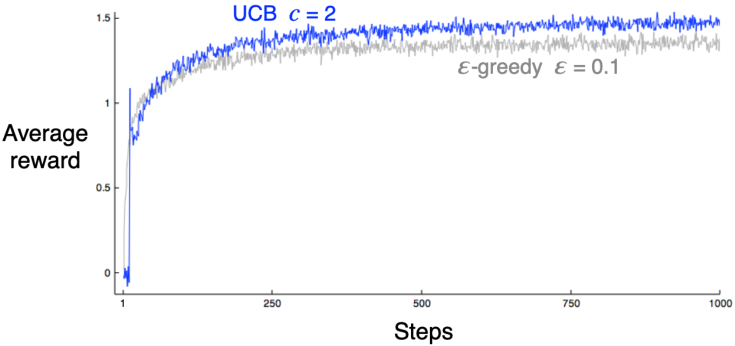
Upper Confidence Bound (UCB) action selection

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

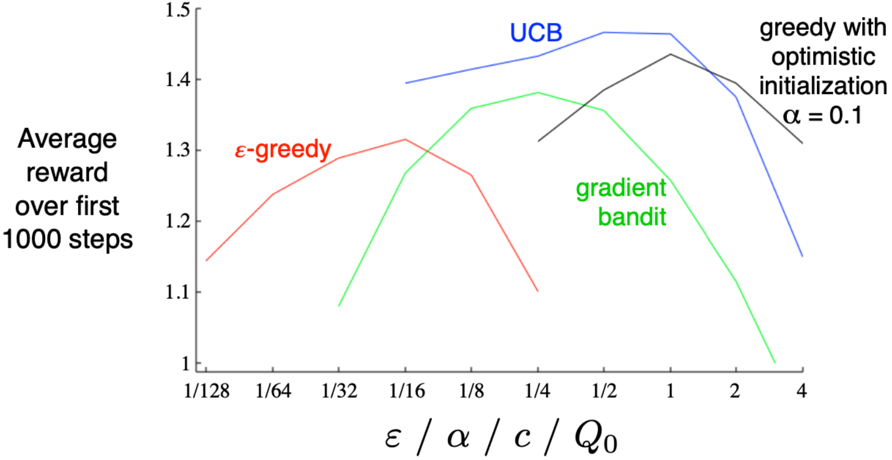
$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$



Demo

Interactive demo no longer active, let me know if you find one:
https://pavlov.tech/2019/03/02/animated-multi-armed-bandit-policies/

Comparison of Bandit Algorithms



Bandits Summary

These are all simple methods

- but they are complicated enough – we will build on them
- we should understand them completely
  - there is a lot of theory, e.g., upper/lower bounds
- there are still open questions

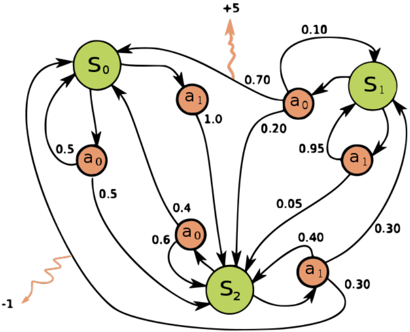
Our first algorithms that learn from evaluative feedback

- and thus must balance exploration and exploitation

Our first algorithms that appear to have a goal

- that learn to maximize reward by trial and error

Standard model for Reinforcement Learning problems



- S – states
- R – rewards
- A – actions
- Discrete steps $t = 0, 1, 2, \ldots$
- Environment *dynamics*

Source: Waldoalvarez @ wikimedia

$$p(s', r|s, a) \leftarrow Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

The Agent Learns a Policy

Policy at step t, denoted π_t, maps from states to actions.

$$\pi_t(a|s) = \text{ probability that } A_t = a \text{ when } S_t = s$$

Special case are **deterministic** policies.

$$\pi_t(s) = \text{ the action taken with } prob = 1 \text{ when } S_t = s$$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience
- Roughly, the agent's goal is to get as much reward as it can **over the long run**.

Return

Suppose the sequence of rewards after step t is:

$$R_{t+1}, R_{t+2}, R_{t+3}, \ldots$$

What do we maximize?

At least three cases, but in all of them, we seek to maximize the **expected return**, $\mathbb{E}\, G_t$, on each step t.

- **Total reward**, $G_t =$ sum of all future reward in the episode
- **Discounted reward**, $G_t =$ sum of all future *discounted* reward
- **Average reward**, $G_t =$ average reward per time step

Episodic Tasks

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze

In episodic tasks, we almost always use simple total reward:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Continuing Tasks

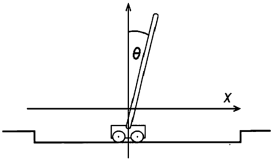
Continuing tasks: interaction does not have natural episodes, but just goes on and on...

In this class, for continuing tasks we will always use *discounted return*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where $0 \leq \gamma \leq 1$, is the **discount rate**.
shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted
Typically, $\gamma = 0.9$

An Example: Pole Balancing



(image from Ma&Likharev 2007)

Avoid **failure**: the pole falling beyond a critical angle or the cart hitting end of track

As an **episodic task** where episode ends upon failure:

reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

As a **continuing task** with discounted return:

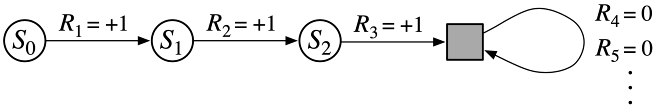
reward = -1 upon failure; 0 otherwise

$\Rightarrow$ return = $-\gamma^k$, for k steps before failure

In either case, return is maximized by avoiding failure for as long as possible.

A Trick to Unify Notation for Returns

- In episodic tasks, we number the time steps of each episode starting from zero.
- We usually do not have to distinguish between episodes, so instead of writing for states in episode j, we write just S_t
- Think of each episode as ending in an absorbing state that always produces reward of zero:



- We can cover **all** cases by writing $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$,

where γ can be 1 only if a zero rewards absorbing state is always reached.

What about average reward?

Tasks that continue forever, but later rewards are not substantially less important than the earlier.

- Patrolling an area against patient intruders
- Controlling vibrations of an airplane

Not very common in AI problems.

Summary

RL is a set of methods to learn a policy from an interaction with environment

The goal is to maximise return derived from immediate rewards

The simplest RL problem is the multi-armed bandit problem

- exploration vs. exploitation problem
- ϵ-greedy, optimistic initialisation, UCB

Canonical model of RL problems is MDP