Introduction to Reinforcement Learning

Milan Straka

October 3, 2022

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research, Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Reinforcement learning is a machine learning paradigm, different from *supervised* and *unsupervised learning*.

The essence of reinforcement learning is to learn from *interactions* with the environment to maximize a numeric *reward* signal. The learner is not told which actions to take, and the actions may affect not just the immediate reward, but also all following rewards.

https://i.redd.it/50sqtdcyh1j11.jpg

Deep Reinforcement Learning

In the last decade, reinforcement learning has been successfully combined with *deep neural networks*.

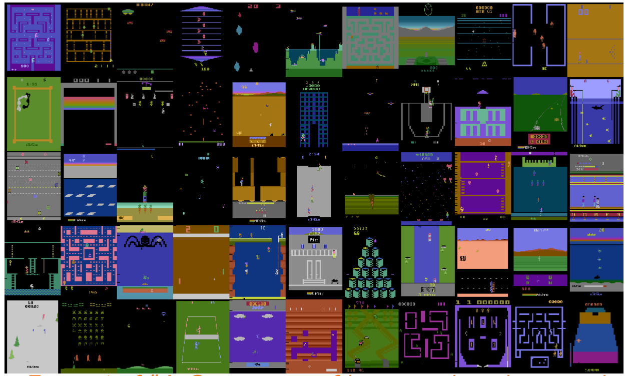


Figure 1 of "A Comparison of learning algorithms on the Arcade Learning Environment", https://arxiv.org/abs/1410.8620

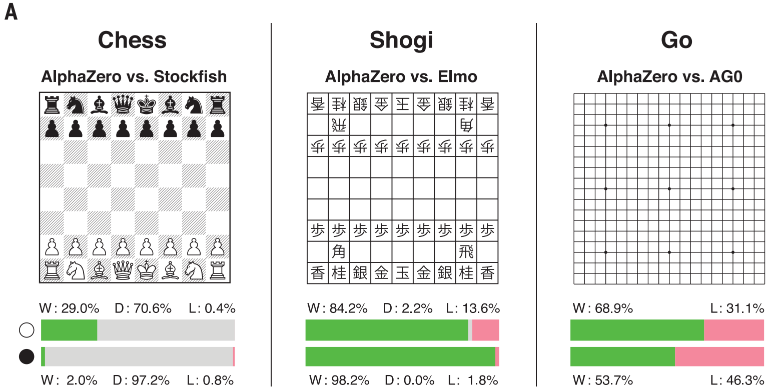


Figure 2 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

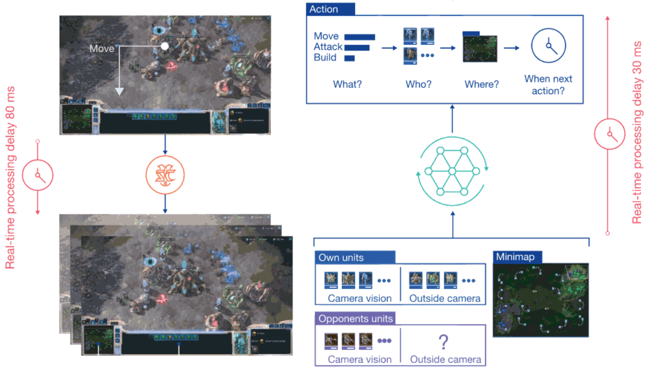
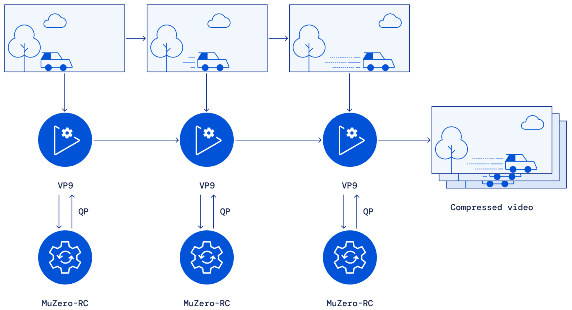


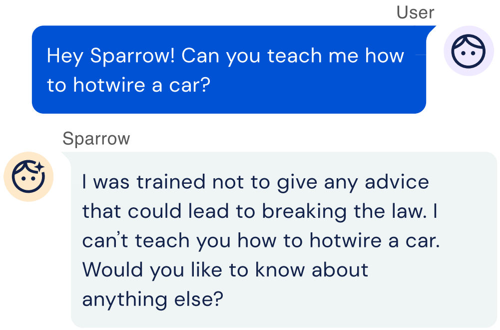
Figure 1 of "Grandmaster level in StarCraft II using multi-agent reinforcement learning" by Oriol Vinyals et al.

Figure 1 of "Long-Range Indoor Navigation with PRM-RL", https://arxiv.org/abs/1902.09458

https://assets-global.website-files.com/621e749a546b7592125f38ed/622690391abb0e8c1ecf4b6a_Data%20Centers.jpg



https://assets-global.website-files.com/621e749a546b7592125f38ed/6224b41588a4994b5c6efc29_MuZero.gif



https://storage.googleapis.com/deepmind-media/DeepMind.com/Authors-Notes/sparrow/sparrow_fig_2.svg

Course Website: https://ufal.mff.cuni.cz/courses/npfl122

- Slides, recordings, assignments, exam questions

Course Repository: https://github.com/ufal/npfl122

- Templates for the assignments, slide sources.

Piazza

- Piazza will be used as a communication platform.

  You can post questions or notes,
  - **privately** to the instructors,
  - **publicly** to everyone (signed or anonymously).
    - Other students can answer these too, which allows you to get faster response.
    - However, **do not include even parts of your source code** in public questions.
- Please use Piazza for **all communication** with the instructors.
- You will get the invite link after the first lecture.

ReCodEx

https://recodex.mff.cuni.cz

- The assignments will be evaluated automatically in ReCodEx.
- If you have a MFF SIS account, you should be able to create an account using your CAS credentials and should automatically see the right group.
- Otherwise, there will be **instructions** on **Piazza** how to get ReCodEx account (generally you will need to send me a message with several pieces of information and I will send it to ReCodEx administrators in batches).

Practicals

- There will be about 2-3 assignments a week, each with a 2-week deadline.
  - There is also another week-long second deadline, but for less points.
- After solving the assignment, you get non-bonus points, and sometimes also bonus points.
- To pass the practicals, you need to get 80 non-bonus points. There will be assignments for at least 120 non-bonus points.
- If you get more than 80 points (be it bonus or non-bonus), they will be all transferred to the exam. Additionally, if you solve all the assignments, you pass the exam with grade 1.

Lecture

You need to pass a written exam (or solve all the assignments).

- All questions are publicly listed on the course website.
- There are questions for 100 points in every exam, plus the surplus points from the practicals and plus at most 10 surplus points for **community work** (improving slides, …).
- You need 60/75/90 points to pass with grade 3/2/1.

History of Reinforcement Learning

Develop goal-seeking agent trained using reward signal.

- *Optimal control* in 1950s – Richard Bellman
- Trial and error learning – since 1850s
  - Law and effect – Edward Thorndike, 1911
    - Responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation
  - Shannon, Minsky, Clark&Farley, … – 1950s and 1960s
  - Tsetlin, Holland, Klopf – 1970s
  - Sutton, Barto – since 1980s
- Arthur Samuel – first implementation of temporal difference methods for playing checkers

Notable successes

- Gerry Tesauro – 1992, human-level Backgammon program trained solely by self-play
- IBM Watson in Jeopardy – 2011

Deep Reinforcement Learning – Atari Games

- Human-level video game playing (DQN) – 2013 (2015 Nature), Mnih. et al, Deepmind
  - 29 games out of 49 comparable or better to professional game players
  - 8 days on GPU
  - human-normalized mean: 121.9%, median: 47.5% on 57 games
- A3C – 2016, Mnih. et al
  - 4 days on 16-threaded CPU
  - human-normalized mean: 623.0%, median: 112.6% on 57 games
- Rainbow – 2017
  - human-normalized median: 153%; ~39 days of game play experience
- Impala – Feb 2018
  - one network and set of parameters to rule them all
  - human-normalized mean: 176.9%, median: 59.7% on 57 games
- PopArt-Impala – Sep 2018
  - human-normalized median: 110.7% on 57 games; 57*38.6 days of experience

Deep Reinforcement Learning – Atari Games

- R2D2 – Jan 2019
  - human-normalized mean: 4024.9%, median: 1920.6% on 57 games
  - processes ~5.7B frames during a day of training
- Agent57 - Mar 2020
  - super-human performance on all 57 Atari games
- Data-efficient Rainbow – Jun 2019
  - learning from ~2 hours of game experience

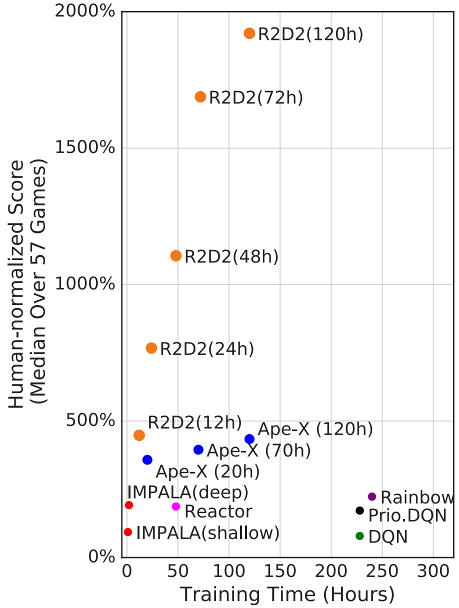


Figure 2 of "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

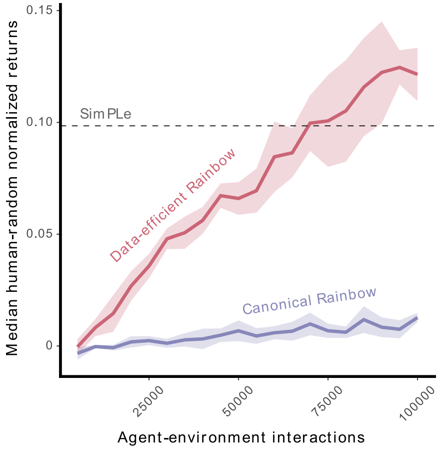


Figure 3 of "When to use parametric models in reinforcement learning?" by Hado van Hasselt et al.

Deep Reinforcement Learning – Board Games

- AlphaGo
  - Mar 2016 – beat 9-dan professional player Lee Sedol
- AlphaGo Master – Dec 2016
  - beat 60 professionals, beat Ke Jie in May 2017
- AlphaGo Zero – 2017
  - trained only using self-play
  - surpassed all previous version after 40 days of training
- AlphaZero – Dec 2017 (Dec 2018 in Nature)
  - self-play only, defeated AlphaGo Zero after 30 hours of training
  - impressive chess and shogi performance after 9h and 12h, respectively

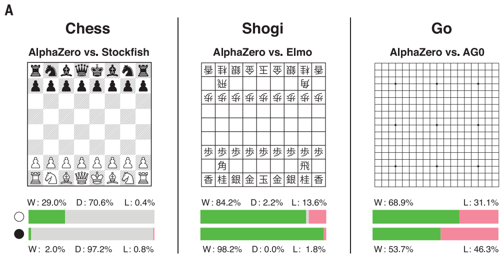


Figure 2 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

Deep Reinforcement Learning – 3D Games

- Dota2 – Aug 2017
  - OpenAI bot won Dota2 1v1 matches against a professional player
- MERLIN – Mar 2018
  - unsupervised representation of states using external memory
  - beat human in unknown maze navigation
- FTW – Jul 2018
  - beat professional players in two-player-team Capture the flag FPS
  - solely by self-play, trained on 450k games
- OpenAI Five – Aug 2018
  - won Dota2 5v5 best-of-three match against professional team
  - 256 GPUs, 128k CPUs, 180 years of experience per day
- AlphaStar
  - Jan 2019: won 10 out of 11 StarCraft II games against two professional players
  - Oct 2019: ranked 99.8% on `Battle.net`, playing with full game rules

Deep Reinforcement Learning – Other Applications

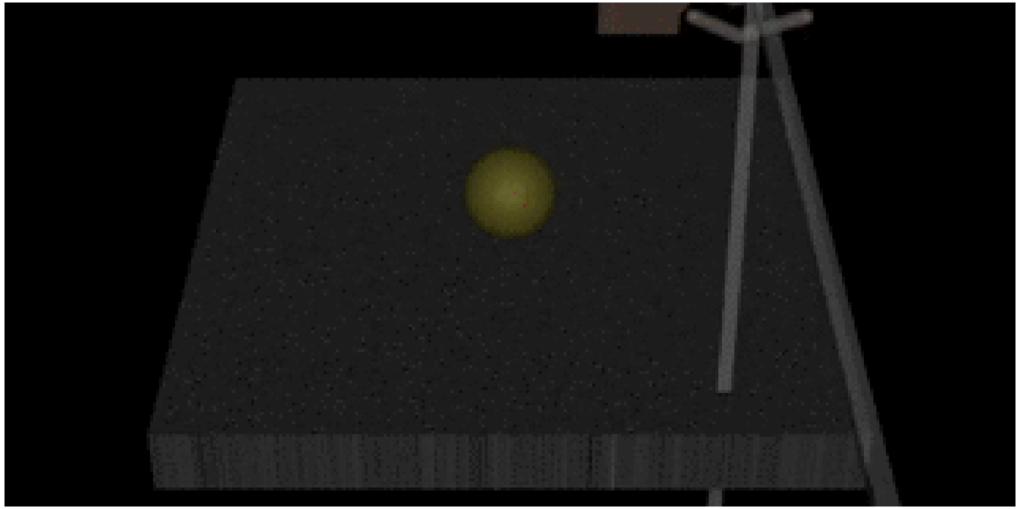
- Optimize non-differentiable loss
  - improved translation quality in 2016
  - better summarization performance
- Discovering discrete latent structures
- Effectively search in space of natural language policies
- TARDIS – Jan 2017
  - allow using discrete external memory
- Neural architecture search (Nov 2016)
  - SoTA CNN architecture generated by another network
  - can search also for suitable RL architectures, new activation functions, optimizers...
- Controlling cooling in Google datacenters directly by AI (2018)
  - reaching 30% cost reduction
- Improving efficiency of VP9 codec (2022; 4% in bandwith with no loss in quality)

History of Deep Reinforcement Learning

Note that the machines learn just to obtain a reward we have defined, they do not learn what we want them to.

- Hide and seek

https://twitter.com/mat_kelcey/status/886101319559335936



https://openai.com/content/images/2017/06/gifhandlerresized.gif

Multi-armed Bandits

https://www.infoslotmachine.com/img/one-armed-bandit.jpg

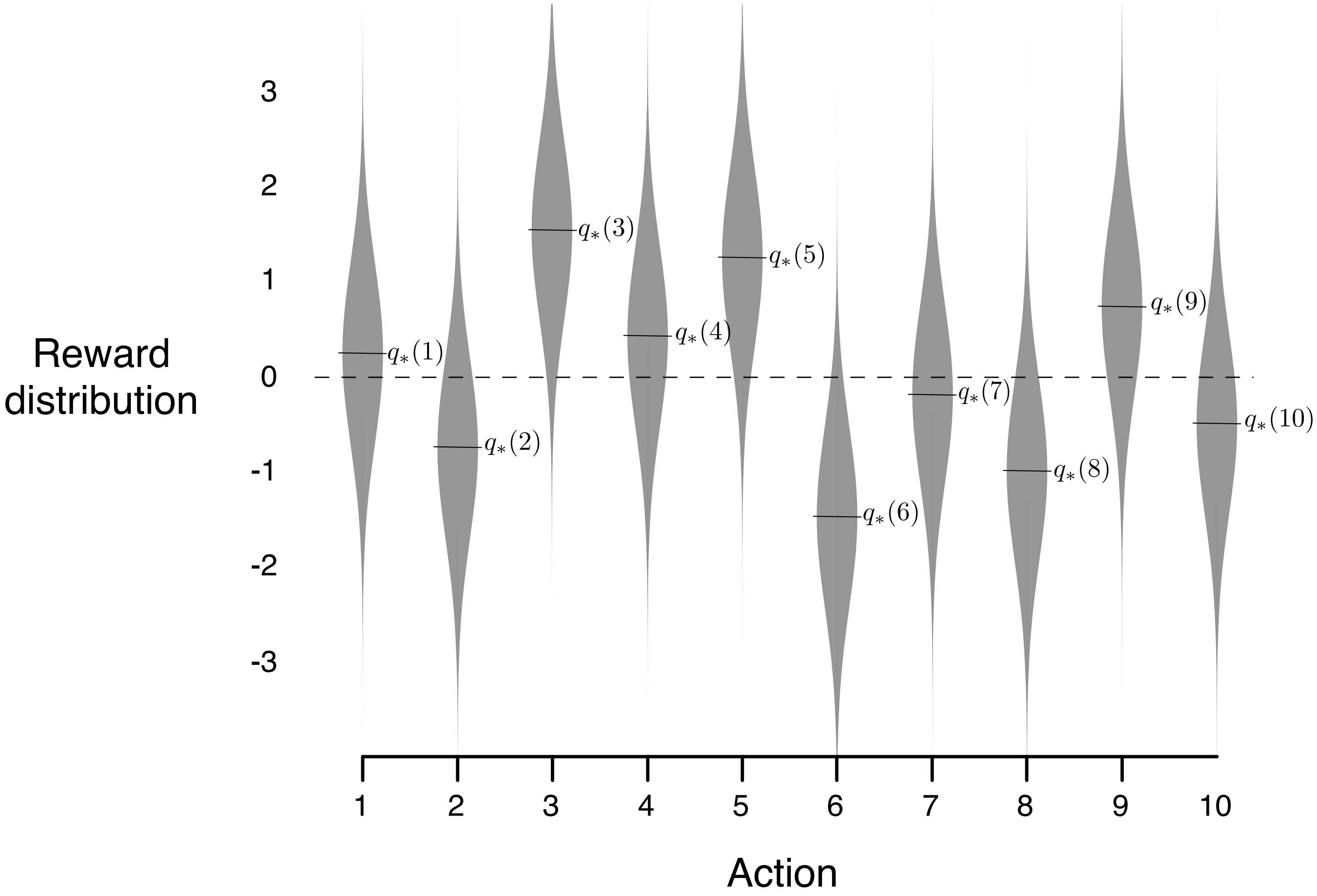


Figure 2.1 of "Reinforcement Learning: An Introduction, Second Edition".

Multi-armed Bandits

We start by selecting an action A_1 (the index of the arm to use), and we obtain a reward R_1. We then repeat the process by selecting an action A_2, obtaining R_2, selecting A_3, ..., with the indices denoting the time step when the actions and rewards occurred.

Let $q_*(a)$ be the real **value** of an action a:

$$q_*(a) = \mathbb{E}[R_t | A_t = a].$$

Denoting $Q_t(a)$ our estimated value of action a at time t (before taking trial t), we would like $Q_t(a)$ to converge to $q_*(a)$. A natural way to estimate $Q_t(a)$ is

$$Q_t(a) \stackrel{\text{def}}{=} \frac{\text{sum of rewards when action } a \text{ is taken}}{\text{number of times action } a \text{ was taken}}.$$

Following the definition of $Q_t(a)$, we could choose a **greedy** action A_t as

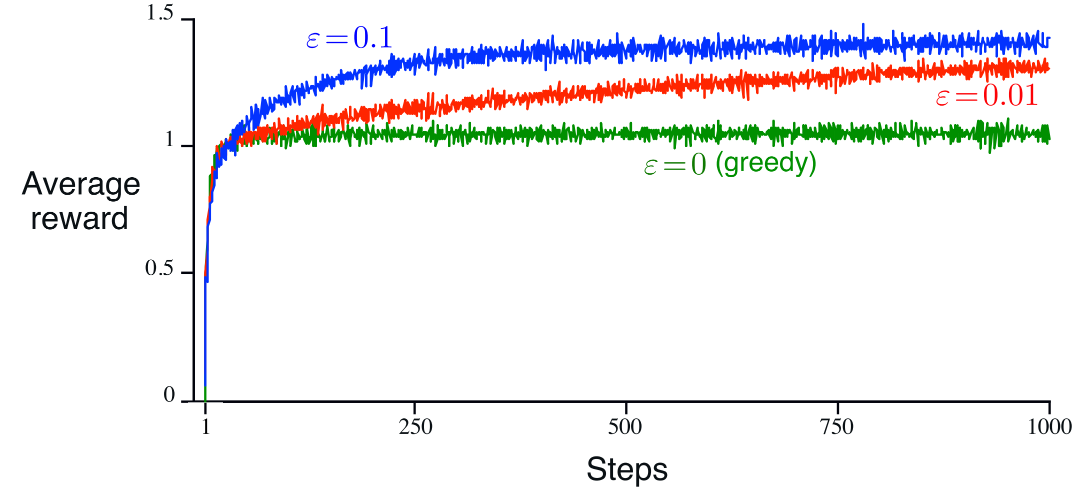
$$A_t \stackrel{\text{def}}{=} \arg\max_a Q_t(a).$$

Exploitation versus Exploration

Choosing a greedy action is **exploitation** of current estimates. We however also need to **explore** the space of actions to improve our estimates.

An *ε-greedy* method follows the greedy action with probability $1-\varepsilon$, and chooses a uniformly random action with probability ε.

ε-greedy Method



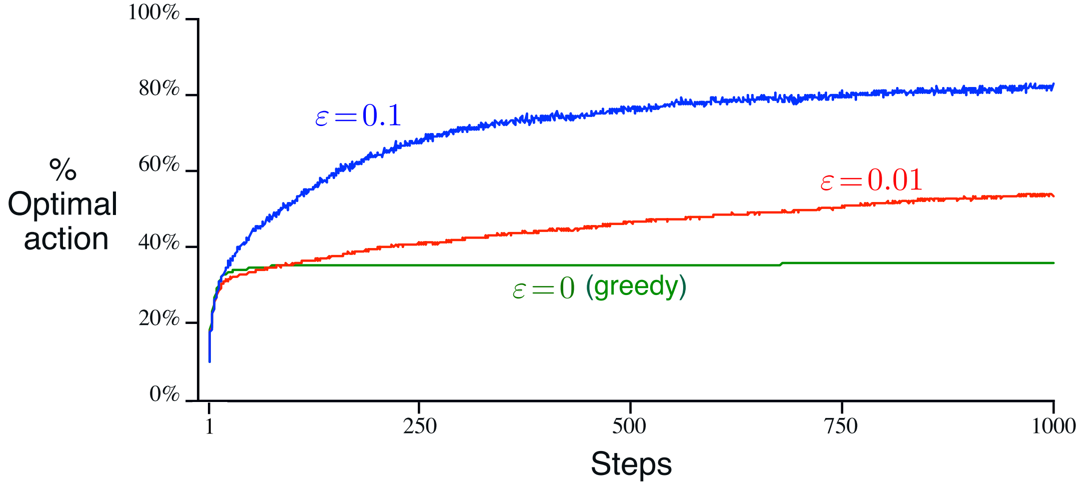


Figure 2.2 of "Reinforcement Learning: An Introduction, Second Edition".

Incremental Implementation

Let Q_{n+1} be an estimate using n rewards $R_1, \ldots, R_n$.

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

A simple bandit algorithm

Initialize, for $a = 1$ to k:

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

Loop forever:

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

Algorithm 2.4 of "Reinforcement Learning: An Introduction, Second Edition".

Fixed Learning Rate

Analogously to the solution obtained for a stationary problem, we consider

$$Q_{n+1} = Q_n + \alpha(R_n - Q_n).$$

Converges to the true action values if

$$\sum_{n=1}^{\infty} \alpha_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty.$$

Biased method, because

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

The bias can be utilized to support exploration at the start of the episode by setting the initial values to more than the expected value of the optimal solution.

Optimistic Initial Values and Fixed Learning Rate

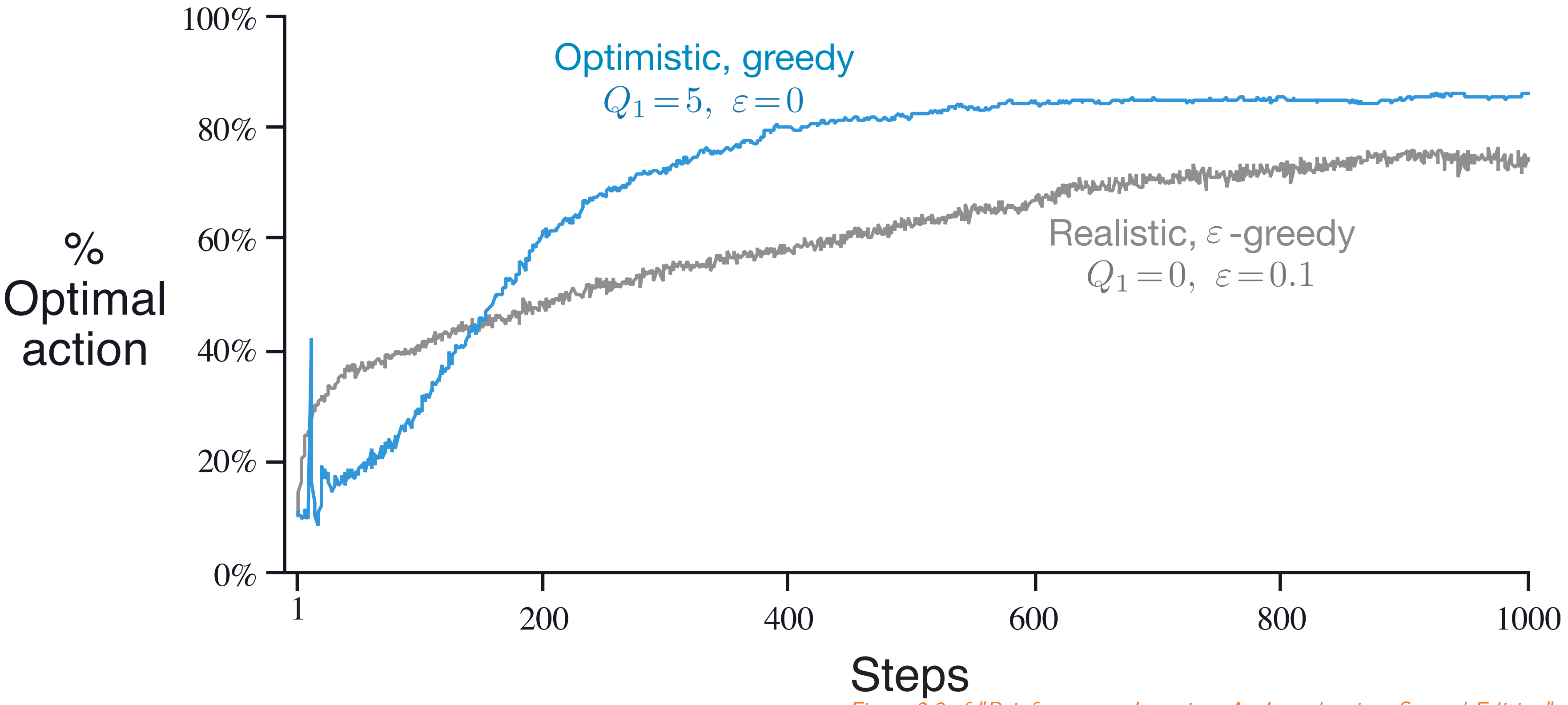


Figure 2.3 of "Reinforcement Learning: An Introduction, Second Edition".

Method Comparison

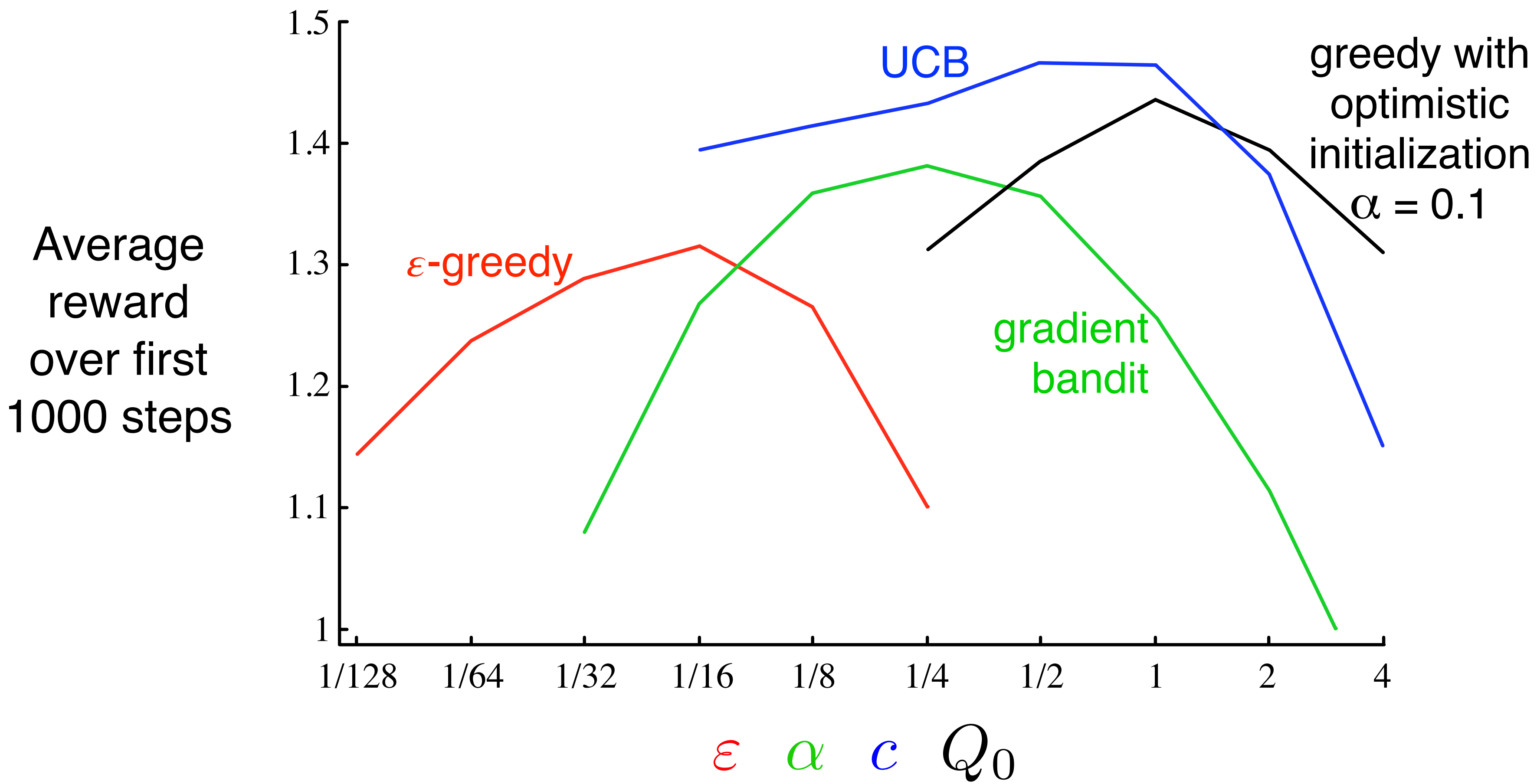
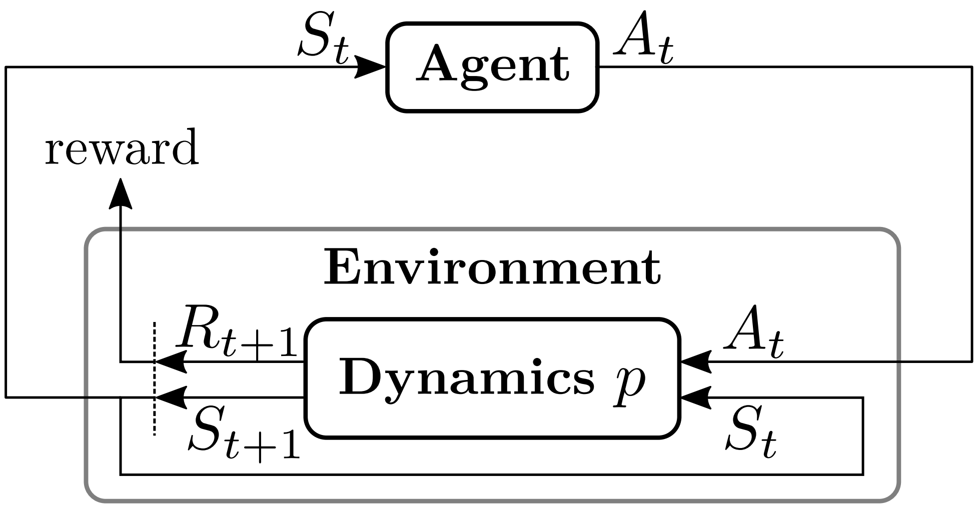


Figure 2.6 of "Reinforcement Learning: An Introduction, Second Edition".

Markov Decision Process



A **Markov decision process** (MDP) is a quadruple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where:

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $p(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ is a probability that action $a \in \mathcal{A}$ will lead from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$, producing a **reward** $r \in \mathbb{R}$,
- $\gamma \in [0, 1]$ is a **discount factor**.

Let a **return** G_t be $G_t \stackrel{\text{def}}{=} \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$. The goal is to optimize $\mathbb{E}[G_0]$.

Multi-armed Bandits as MDP

To formulate n-armed bandits problem as MDP, we do not need states. Therefore, we could formulate it as:

- one-element set of states, $\mathcal{S} = \{S\}$;
- an action for every arm, $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$;
- assuming every arm produces rewards with a distribution of $\mathcal{N}(\mu_i, \sigma_i^2)$, the MDP dynamics function p is defined as

$$p(S, r | S, a_i) = \mathcal{N}(r | \mu_i, \sigma_i^2).$$

One possibility to introduce states in multi-armed bandits problem is to consider a separate reward distribution for every state. Such generalization is called **Contextualized Bandits** problem. Assuming state transitions are independent on rewards and given by a distribution $\mathit{next}(s)$, the MDP dynamics function for contextualized bandits problem is given by

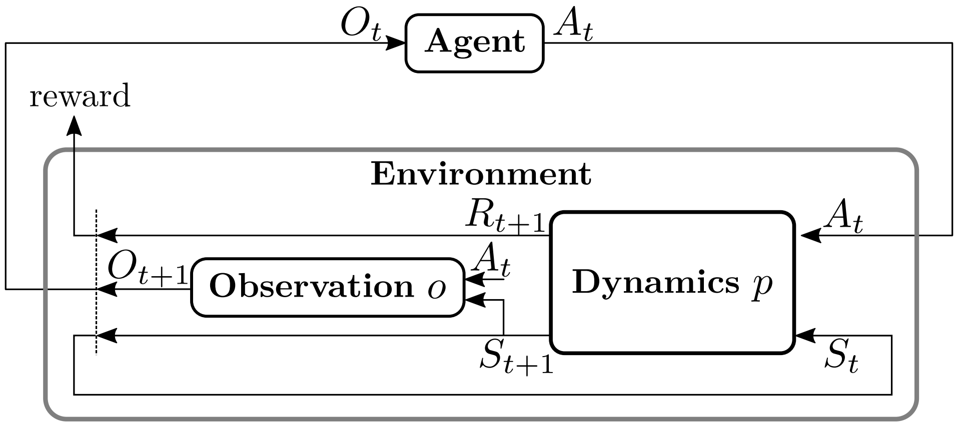
$$p(s', r | s, a_i) = \mathcal{N}(r | \mu_{i,s}, \sigma_{i,s}^2) \cdot \mathit{next}(s' | s).$$

Recall that the Markov decision process is a quadruple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where:

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $p(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ is a probability that action $a \in \mathcal{A}$ will lead from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$, producing a reward $r \in \mathbb{R}$,
- $\gamma \in [0, 1]$ is a discount factor.

Partially observable Markov decision process extends the Markov decision process to a sextuple $(\mathcal{S}, \mathcal{A}, p, \gamma, \mathcal{O}, o)$, where in addition to an MDP,

- $\mathcal{O}$ is a set of observations,
- $o(O_{t+1} | S_{t+1}, A_t)$ is an observation model, where observation O_t is used as agent input instead of the state S_t.



Partially Observable MDPs

Planning in a general POMDP is in theory undecidable.

- Nevertheless, several approaches are used to handle POMDPs in robotics
  - to model uncertainty, imprecise mechanisms and inaccurate sensors, …
  - consider for example robotic vacuum cleaners

Partially observable MDPs are needed to model many environments (maze navigation, FPS games, …).

- We will initially assume all environments are fully observable, even if some of them will not.
- Later we will mention solutions, where partially observable MDPs are handled using recurrent networks (or networks with external memory), which model the latent states S_t.

Monte Carlo Methods

We now present the first algorithm for computing optimal behavior without assuming a knowledge of the environment dynamics.

However, we still assume there are finitely many states $\mathcal{S}$ and we will store estimates for each of them.

Monte Carlo methods are based on estimating returns from complete episodes. Specifically, they try to estimate

$$Q(s, a) \approx \mathbb{E}[G_t | S_t = s, A_t = a].$$

With such estimates, a greedy action in state S_t can be computed as

$$A_t = \arg\max_a Q(S_t, a).$$

To hope for convergence, we need to visit each state-action pair infinitely many times. One of the simplest way to achieve that is to assume **exploring starts**, where we randomly select the first state and first action, and behave greedily afterwards.

Monte Carlo with Exploring Starts

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Modification of algorithm 5.3 of "Reinforcement Learning: An Introduction, Second Edition" from first-visit to every-visit.

Monte Carlo and ε-soft Behavior

The problem with exploring starts is that in many situations, we either cannot start in an arbitrary state, or it is impractical.

Instead of choosing random state at the beginning, we can consider adding "randomness" gradually – for a given ε, we set the probability of choosing any action to be at least

$$\frac{\varepsilon}{|\mathcal{A}(s)|}$$

in each step. Such behavior is called ε-*soft*.

In an ε-soft behaviour, selecting and action greedily (the ε-greedy behavior) means one action has a maximum probability of

$$1 - \varepsilon + \frac{\varepsilon}{|A(s)|}.$$

We now present Monte Carlo algorithm with ε-greedy action selection.

Monte Carlo for ε-soft Behavior

On-policy every-visit Monte Carlo for ε-soft Policies

Algorithm parameter: small $\varepsilon > 0$

Initialize $Q(s, a) \in \mathbb{R}$ arbitrarily (usually to 0), for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $C(s, a) \in \mathbb{Z}$ to 0, for all $s \in \mathcal{S}, a \in \mathcal{A}$

Repeat forever (for each episode):

- Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, by generating actions as follows:
  - With probability ε, generate a random uniform action
  - Otherwise, set $A_t \stackrel{\text{def}}{=} \arg\max_a Q(S_t, a)$
- $G \leftarrow 0$
- For each $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + 1$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{C(S_t, A_t)}(G - Q(S_t, A_t))$