Introduction to Reinforcement Learning

Milan Straka

October 7, 2019

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

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 playing program trained solely by self-play
- IBM Watson in Jeopardy – 2011

Recent successes

- 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

Recent successes

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

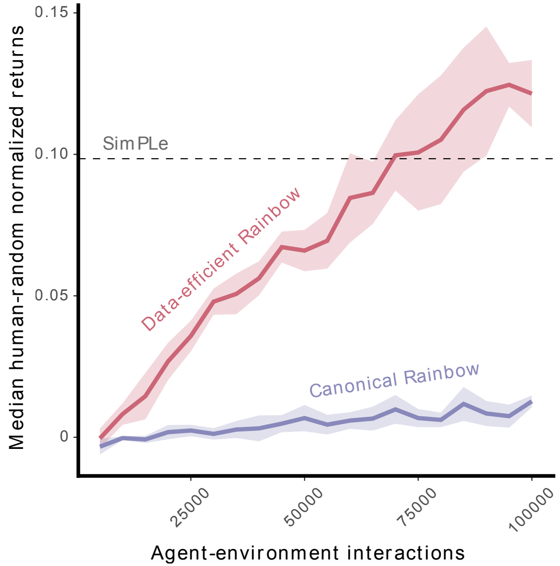


Figure 3 of the paper "When to use parametric models in reinforcement learning?" by Hado van Hasselt et al.

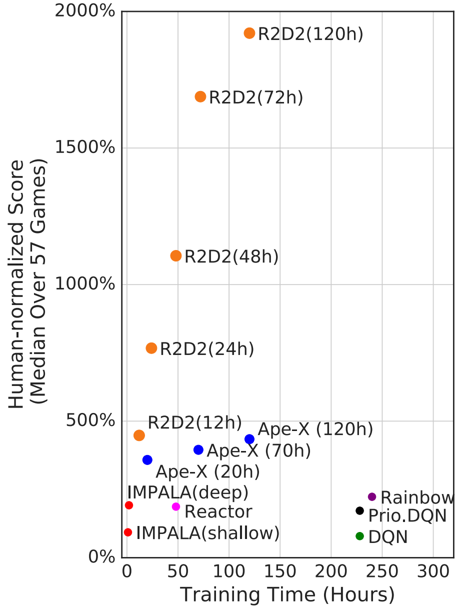


Figure 2 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Recent successes

- 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

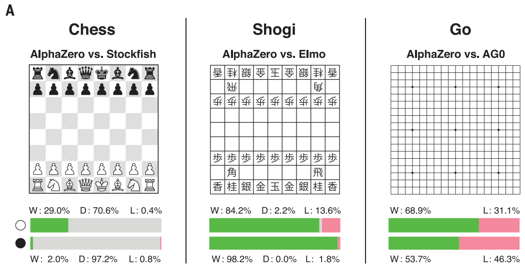


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

Recent successes

- Dota2 – Aug 2017
  - won 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
    - each 5 minutes, 4500 agent steps (15 per second)
- OpenAI Five – Aug 2018
  - won 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

Recent successes

- 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...

Multi-armed Bandits

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

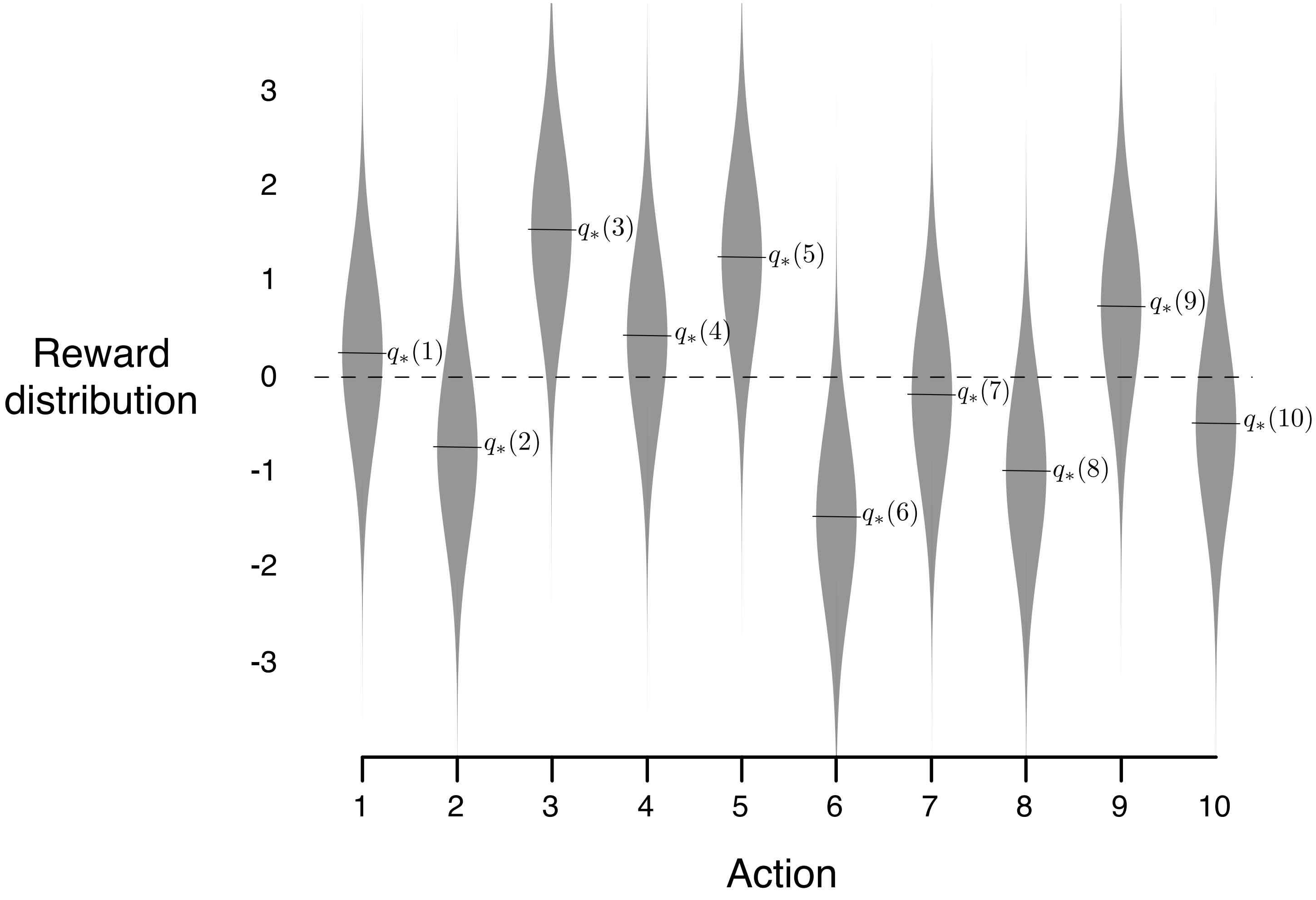


Figure 2.1 of "Reinforcement Learning: An Introduction, Second Edition".

We start by selecting action A_1, which is the index of the arm to use, and we get a reward of R_1. We then repeat the process by selecting actions A_2, A_3, ...

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

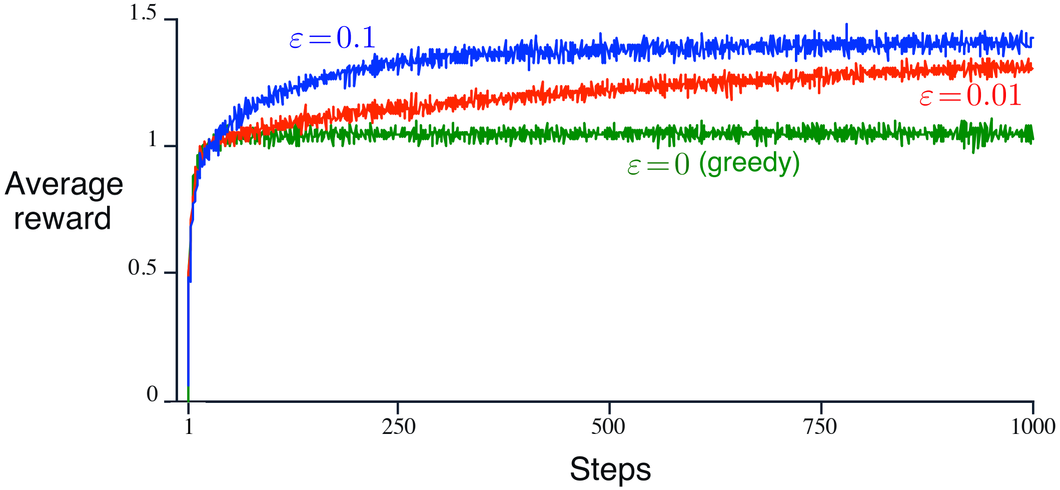
$$A_t \stackrel{\text{def}}{=} \arg\max_a Q_t(a).$$

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



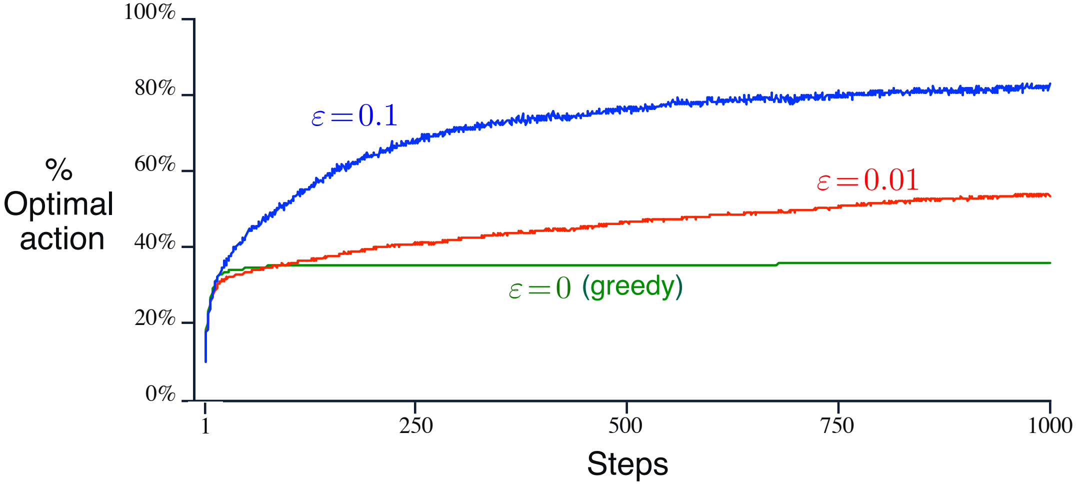


Figure 2.2 of "Reinforcement Learning: An Introduction, Second Edition".

Incremental Implementation

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

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

ε-greedy Method Algorithm

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

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

$R \leftarrow bandit(A)$

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

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

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.

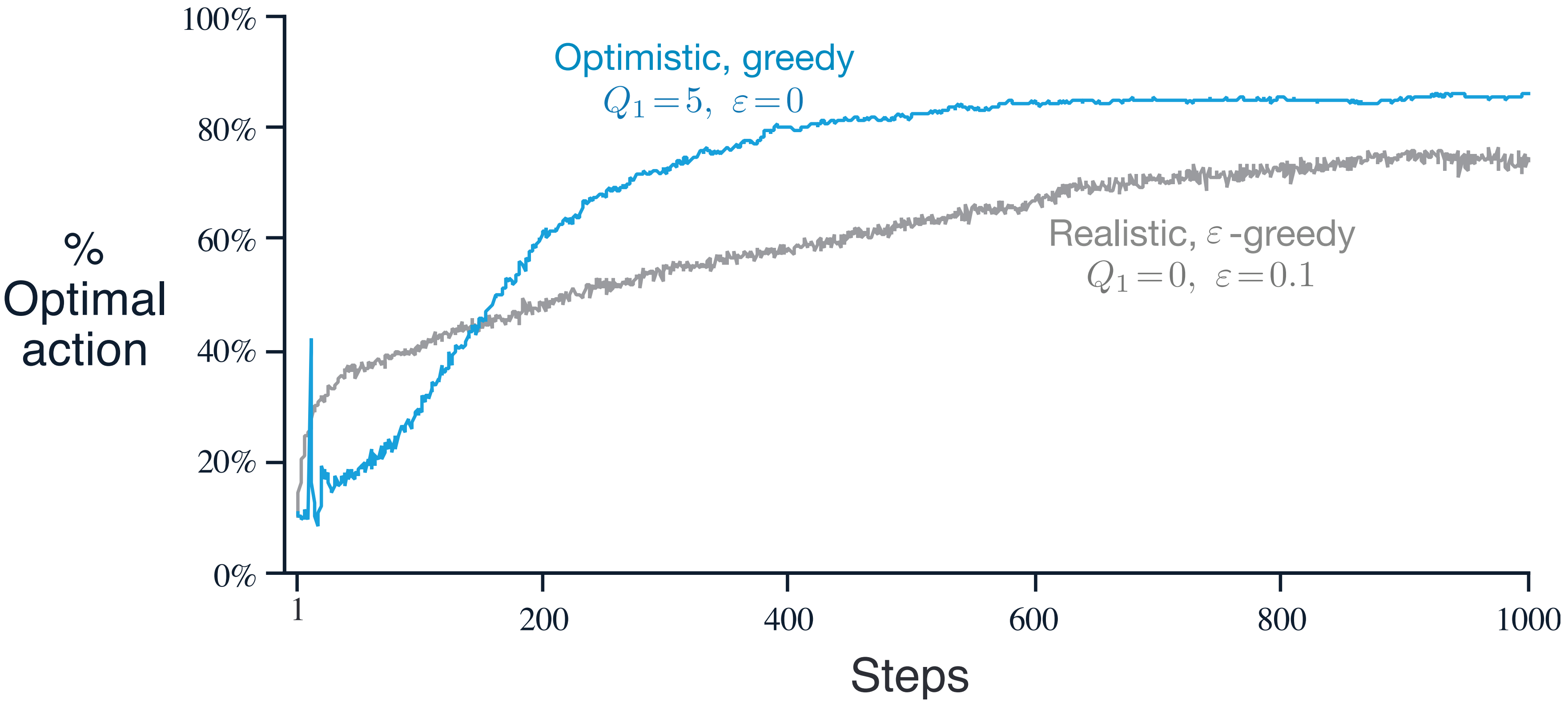


Figure 2.3 of "Reinforcement Learning: An Introduction, Second Edition".

Upper Confidence Bound

Using same epsilon for all actions in ε-greedy method seems inefficient. One possible improvement is to select action according to upper confidence bound (instead of choosing a random action with probability ε):

$$A_t \stackrel{\text{def}}{=} \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right].$$

The updates are then performed as before (e.g., using averaging, or fixed learning rate α).

Note that if $N_t(a) = 0$, the right expression is assumed to have a value of ∞.

Motivation Behind Upper Confidence Bound

Actions with little average reward are probably selected too often.

Instead of simple ε-greedy approach, we might try selecting an action as little as possible, but still enough to converge.

Assuming random variables X_i bounded by $[0,1]$ and $\bar{X} = \sum_{i=1}^N X_i$, (Chernoff-)Hoeffding's inequality states that

$$P(\bar{X} - \mathbb{E}[\bar{X}] \geq \delta) \leq e^{-2n\delta^2}.$$

Our goal is to choose δ such that for every action,

$$P(Q_t(a) - q_*(a) \geq \delta) \leq \left(\frac{1}{t}\right)^\alpha.$$

We can achieve the required inequality (with $\alpha = 2$) by setting

$$\delta \geq \sqrt{(\ln t)/N_t(a)}.$$

Asymptotical Optimality of UCB

We define *regret* as a difference of maximum of what we could get (i.e., repeatedly using action with maximum expectation) and what a strategy yields, i.e.,

$$\mathit{regret}_N \stackrel{\text{def}}{=} N \max_a q_*(a) - \sum_{i=1}^N \mathbb{E}[R_i].$$

It can be shown that regret of UCB is asymptotically optimal, see Lai and Robbins (1985), Asymptotically Efficient Adaptive Allocation Rules.

Upper Confidence Bound Results

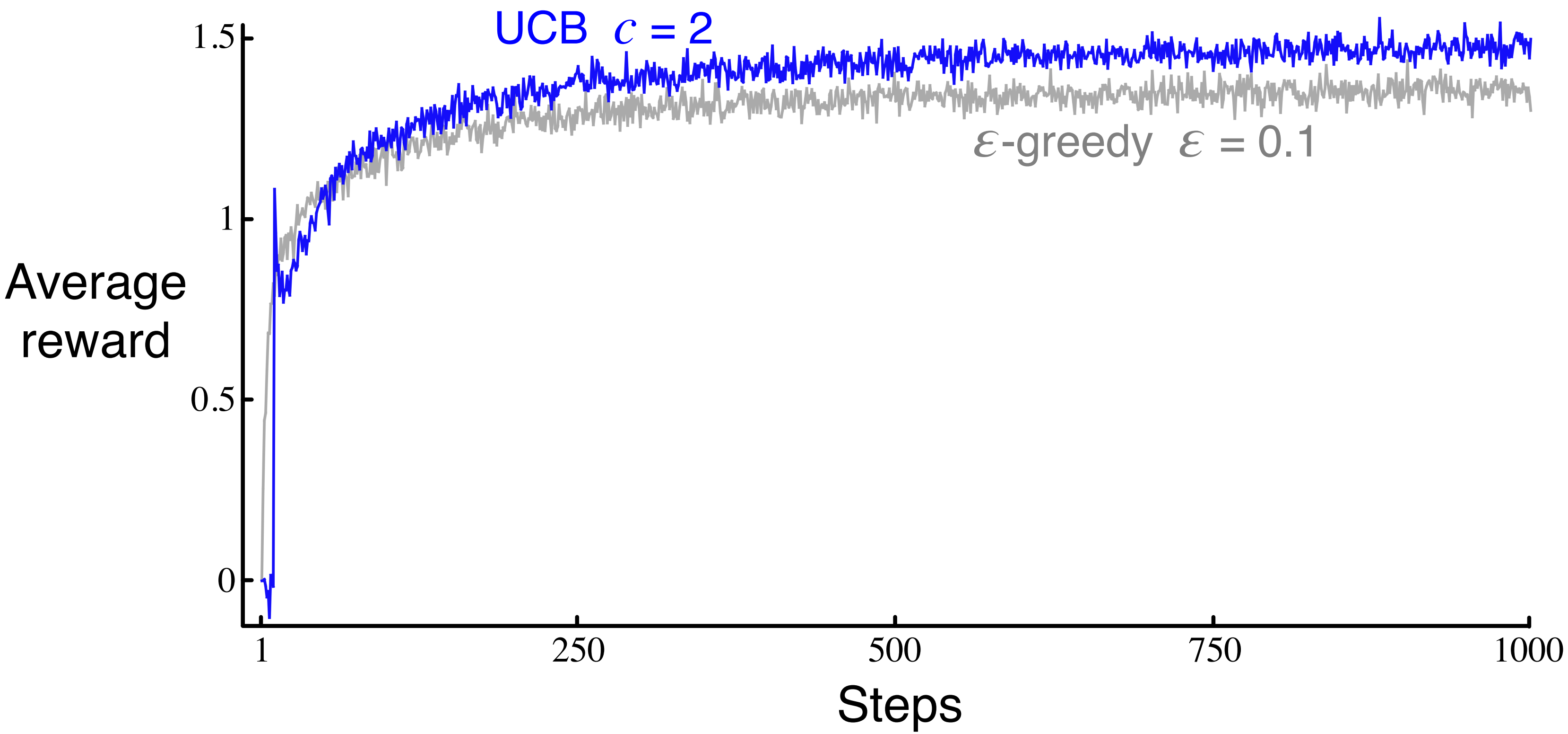


Figure 2.4 of "Reinforcement Learning: An Introduction, Second Edition".

Gradient Bandit Algorithms

Let $H_t(a)$ be a numerical *preference* for an action a at time t.

We could choose actions according to softmax distribution:

$$\pi(A_t = a) \stackrel{\text{def}}{=} \operatorname{softmax}(a) = \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}}.$$

In other words, we model a *distribution* of the most rewarding action.

Usually, all $H_1(a)$ are set to zero, which corresponds to random uniform initial policy.

Using SGD and MLE loss, we can (and later we will) derive the following algorithm:

$$H_{t+1}(a) \leftarrow H_t(a) + \alpha R_t([a = A_t] - \pi(a)).$$

The $[a = A_t]$ is a one-hot vector with the element corresponding to action A_t set to 1.

Gradient Bandit Algorithms

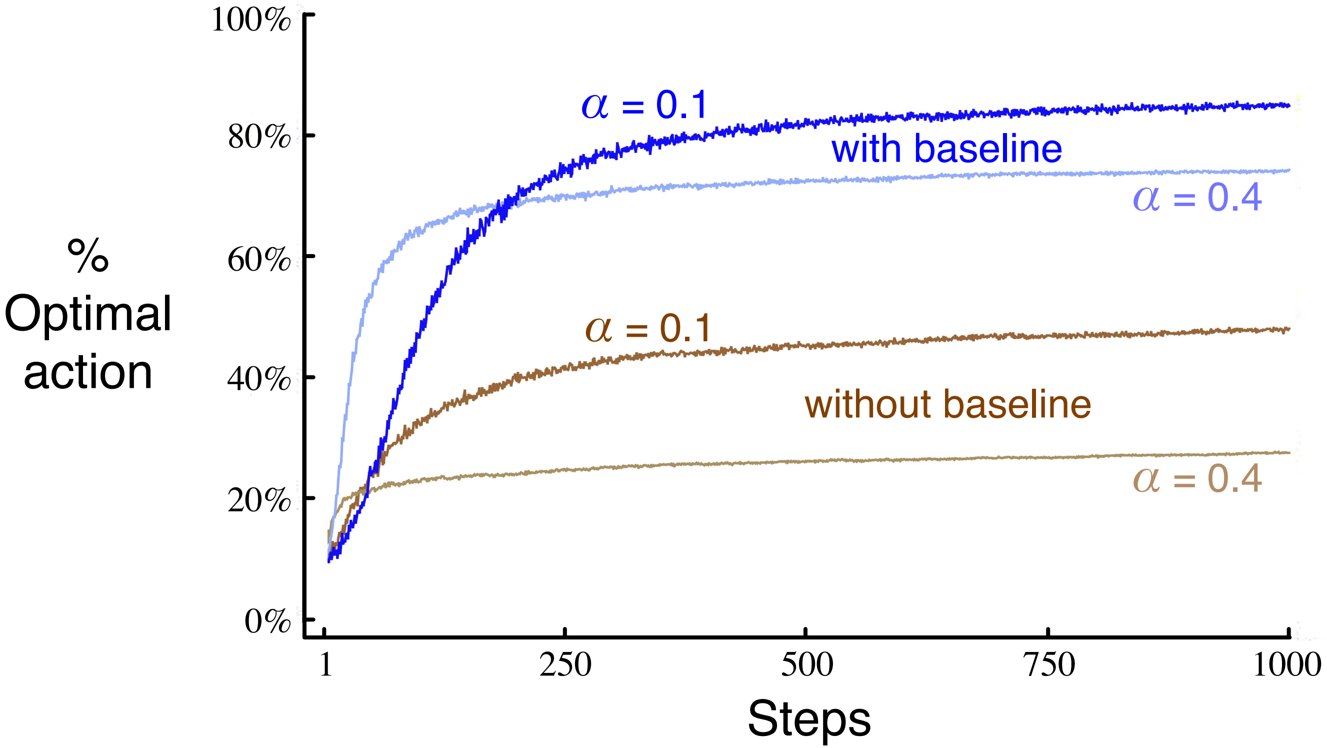


Figure 2.5: Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the $q_*(a)$ are chosen to be near $+4$ rather than near zero.

Figure 2.5 of "Reinforcement Learning: An Introduction, Second Edition".

Method Comparison

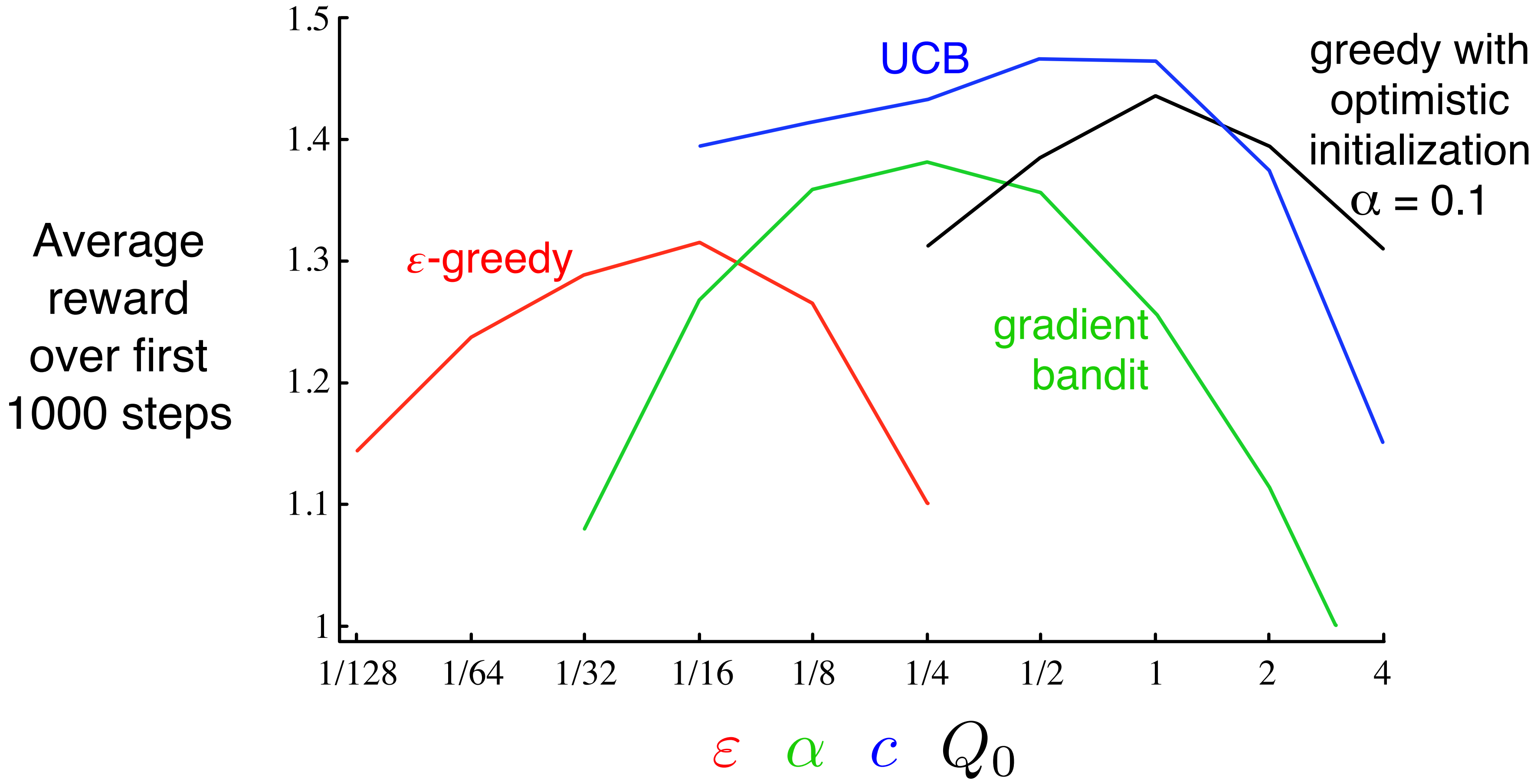


Figure 2.6 of "Reinforcement Learning: An Introduction, Second Edition".