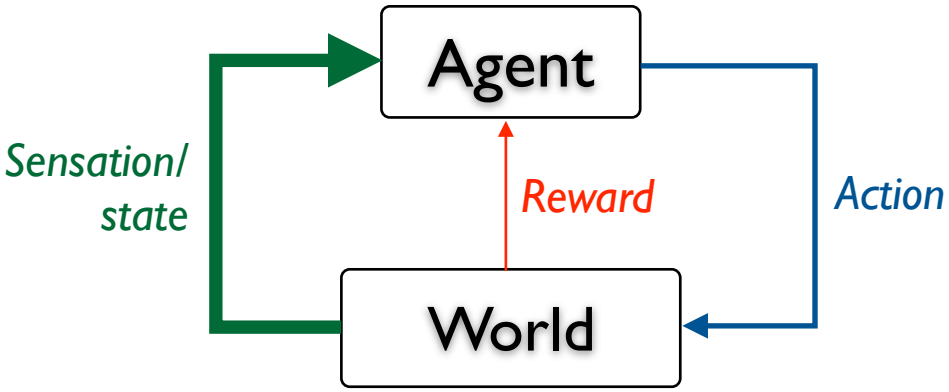
Multi-arm Bandits

The simplest reinforcement learning problem

Based on slides from "Reinforcement Learning" by Sutton and Barto, Chapter 2

- Text: http://incompleteideas.net/book/the-book-2nd.html
- Slides:
  - follow link "Teaching Aids"
  - Google Drive: 2017/Lectures/ contains videos and slides

Reinforcement Learning



- Learning that is more autonomous, requires less input from people
- Can learn for itself based on experience during normal operation

The k-armed Bandit Problem

- On each of an infinite sequence of time steps t=1, 2, 3, …,
  - you **choose** an **action** At from k possibilities
  - and then **receive** a **real-valued reward** Rt from the **Environment**
- The reward depends only on the action taken and is identically, independently distributed (i.i.d.):

Note: means true by definition

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

True Values

- **Challenges**
  - 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*

Dilemma:

- You can't do both, but you *need* to do both
- What do you do? 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:

$Q(a) \leftarrow 0$ *Estimated action-value for "a"*

$N(a) \leftarrow 0$ *Number of times action "a" is taken*

Repeat forever:

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

$R \leftarrow bandit(A)$

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

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

Bandits: What we learned so far

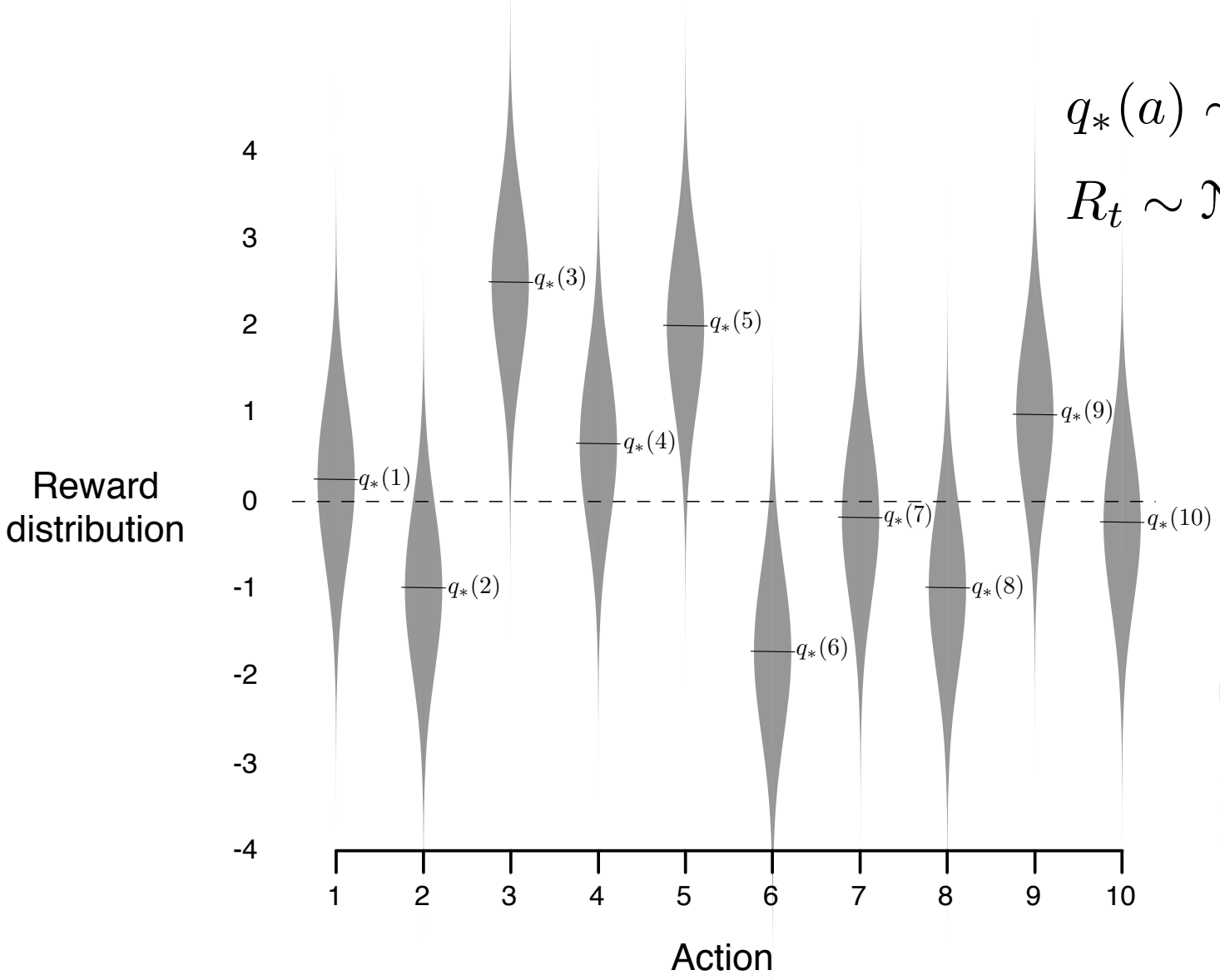
1. *Multi-armed bandits* are a simplification of the real problem
    1. they have action and reward (a goal), but no input or sequentiality
2. A fundamental *exploitation-exploration tradeoff* arises in bandits
    1. ε-*greedy action selection* is the simplest way of trading off
3. *Learning action values* is a key part of solution methods
4. *The 10-armed testbed* illustrates all

One Bandit Task from

The 10-armed Testbed

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

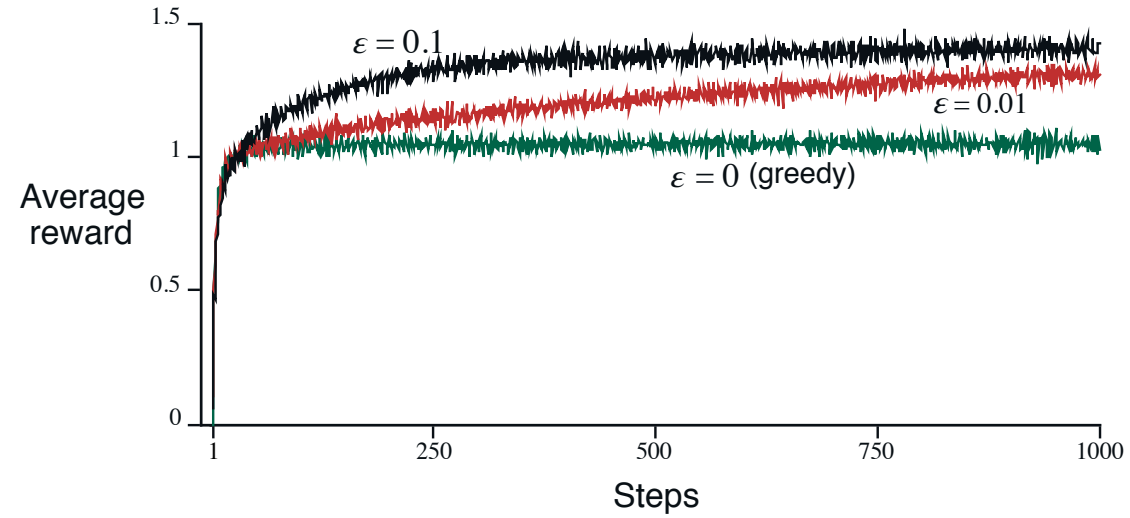
$$R_t \sim \mathcal{N}(q_*(a), 1)$$

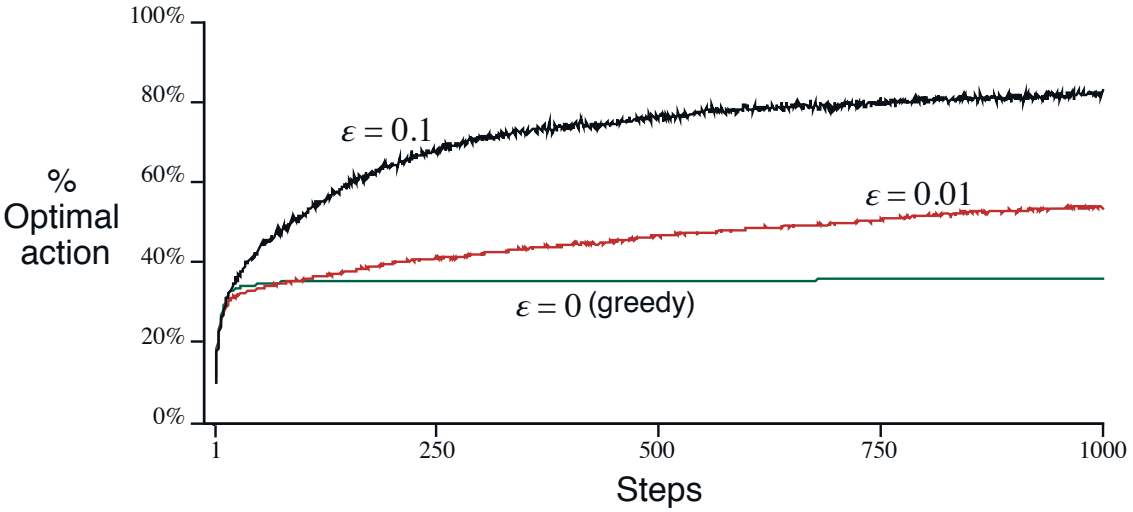


Run for 1000 steps

Repeat the whole thing 2000 times with different bandit tasks

ε-Greedy Methods on the 10-Armed Testbed





Bandits: What we learned so far

1. *Multi-armed bandits* are a simplification of the real problem
    1. they have action and reward (a goal), but no input or sequentiality
2. *The exploitation-exploration tradeoff* arises in bandits
    1. *ε-greedy action selection* is the simplest way of trading off
3. *Learning action values* is a key part of solution methods
4. *The 10-armed testbed* illustrates all
5. Learning as averaging – a *fundamental learning rule*

Averaging ⟶ learning rule

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

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

- Question: 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

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

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

Averaging ⟶ learning rule

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

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

- Question: 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:

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

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":

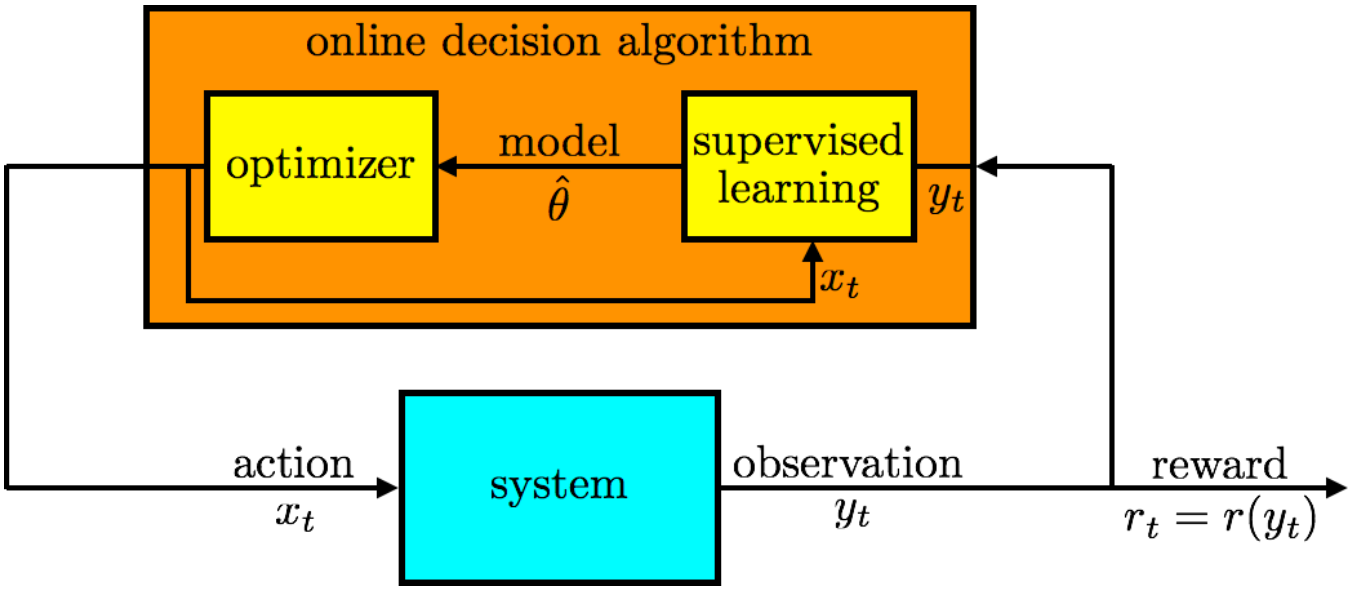
$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big]$$

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

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

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

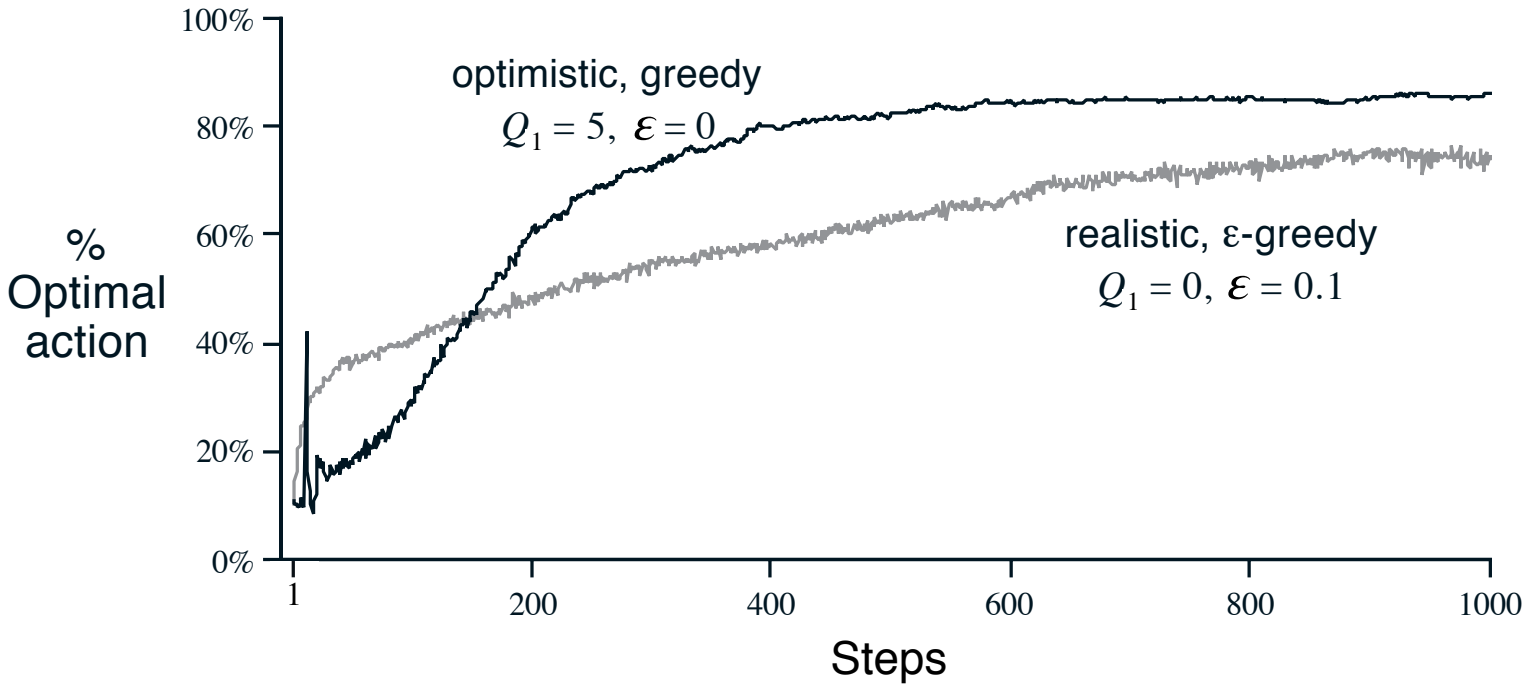
Online Decision Algorithm



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 $\alpha = 0.1$)



Other Solutions for MABs

- Slides based on https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/ figures and note, just simple

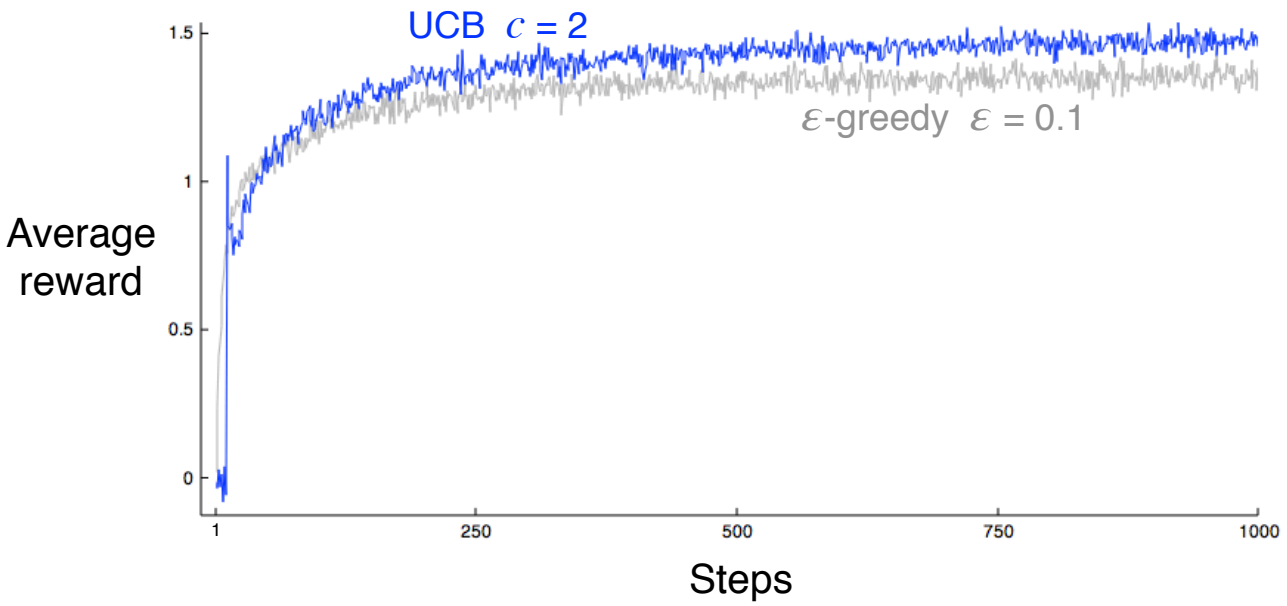
Upper Confidence Bound (UCB) action selection

- 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

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

Sqrt *term measures the uncertainty or variance.*

c>0 *controls level of exploration, determines confidence level*



log t : *increases get smaller over time, but never go away*

Nt(a) *= 0 : infinity, so a is maximized, goes down as data is collected*

Gradient-Bandit Algorithms

- Let $H_t(a)$ be a learned *preference* for taking action a relative to other actions

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

***Policy**: the probability of taking actions **a** at time **t**. "Boltzman or Softmax distribution"*

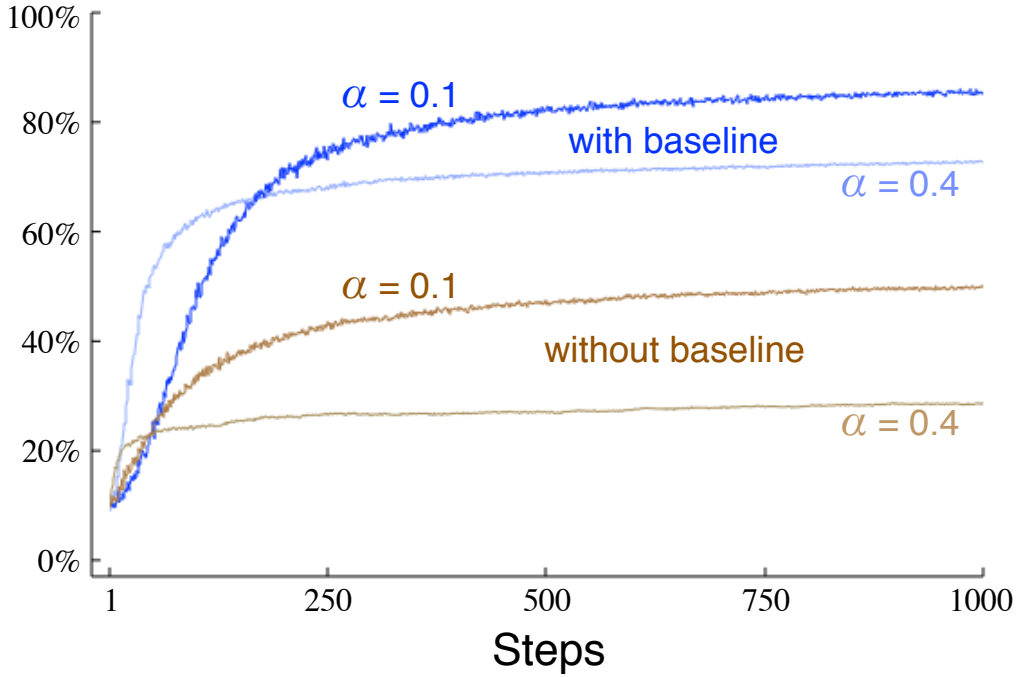
$$H_{t+1}(a) \doteq H_t(a) + \alpha\big(R_t - \bar{R}_t\big)\big(\mathbf{1}_{a=A_t} - \pi_t(a)\big), \qquad \forall a,$$

Baseline: mean reward

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

Probability of taking action a

- ***increases** if it beats the current baseline*
- ***decreases** if it is below the baseline*



“Beta Bernoulli” Bandit

- There are K actions
- When action k is played it produces a reward of 1 with probability θ_k and zero with $1 - \theta_k$
- Each θ_k is like the actions probability or mean reward, but these are unknown.
- We take actions and sample from the distribution to estimate these distributions
- This can be done with a Beta prior

Thompson Sampling

- History
  - Old method from 1930's but not 'discovered' by AI until recently, last 20 years or so
- Main idea
  - Epsilon-greedy is fine but it wastes a lot of time randomly sampling suboptimal actions
  - Samples actions in proportion to belief they are optimal. Shifts away from unlikely optimal actions and focusses on good leads that are still uncertain
- Benefits
  - Easy to use
  - Instantaneously self-correcting
  - Has asymptotic convergence results
- Similarity and differences to UCB

Wasteful Sampling

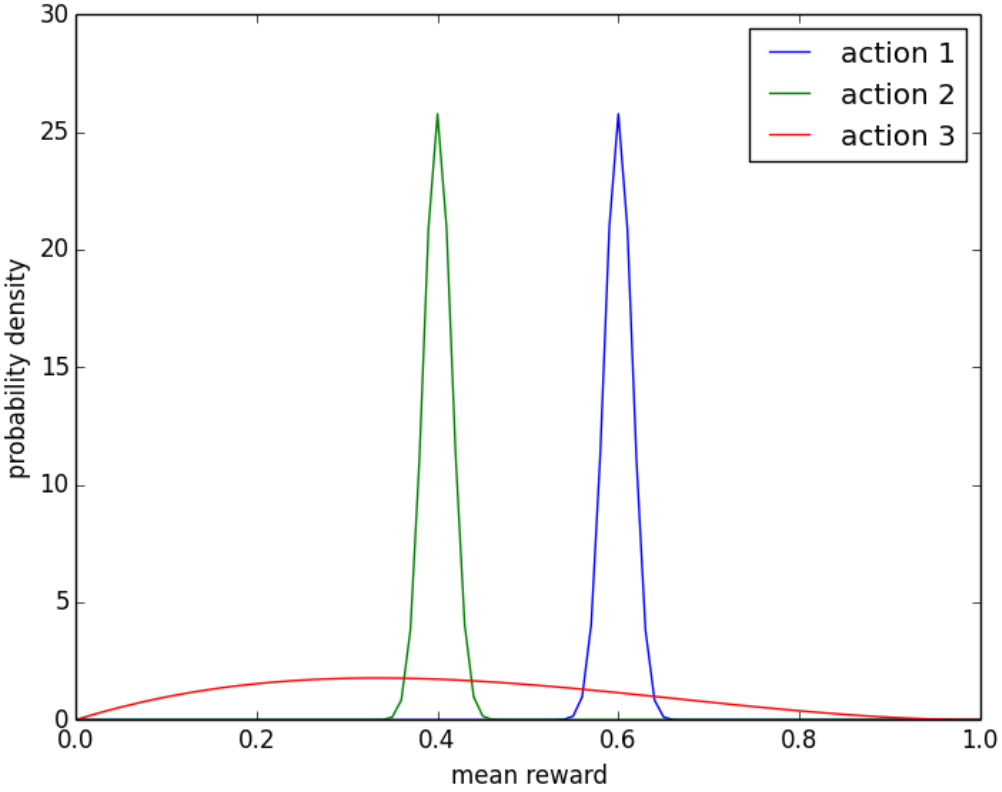


Figure 2.2: Probability density functions over mean rewards.

TS Algorithm

Choose the action **a** according to the probability that it maximizes the expected reward

Requires

1. a likelihood function $P(r \mid \theta, a, x)$
2. a set of parameters θ of the distribution of
3. a prior distribution $P(\theta)$ on these parameters
4. past observations $D = \{(x, a, r)\}$
5. a posterior distribution $P(\theta \mid D) \propto P(D \mid \theta) P(\theta)$

Greedy vs Thompson Sampling

Algorithm 1 BernGreedy(K, α, β)

for $t = 1, 2, \ldots$ **do**
  #estimate model:
  for $k = 1, \ldots, K$ **do**
    $\hat{\theta}_k \leftarrow \alpha_k / (\alpha_k + \beta_k)$
  end for

  #select and apply action:
  $x_t \leftarrow \text{argmax}_k \, \hat{\theta}_k$
  Apply x_t and observe r_t

  #update distribution:
  $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t} + r_t, \beta_{x_t} + 1 - r_t)$
end for

Algorithm 2 BernTS(K, α, β)

for $t = 1, 2, \ldots$ **do**
  #sample model:
  for $k = 1, \ldots, K$ **do**
    Sample $\hat{\theta}_k \sim \text{beta}(\alpha_k, \beta_k)$
  end for

  #select and apply action:
  $x_t \leftarrow \text{argmax}_k \, \hat{\theta}_k$
  Apply x_t and observe r_t

  #update distribution:
  $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t} + r_t, \beta_{x_t} + 1 - r_t)$
end for

Regret

$$\text{Regret}(T) = \sum_{t=1}^{T} \left(\max_{1 \leq k \leq K} \theta_k - \theta_{x_t} \right)$$

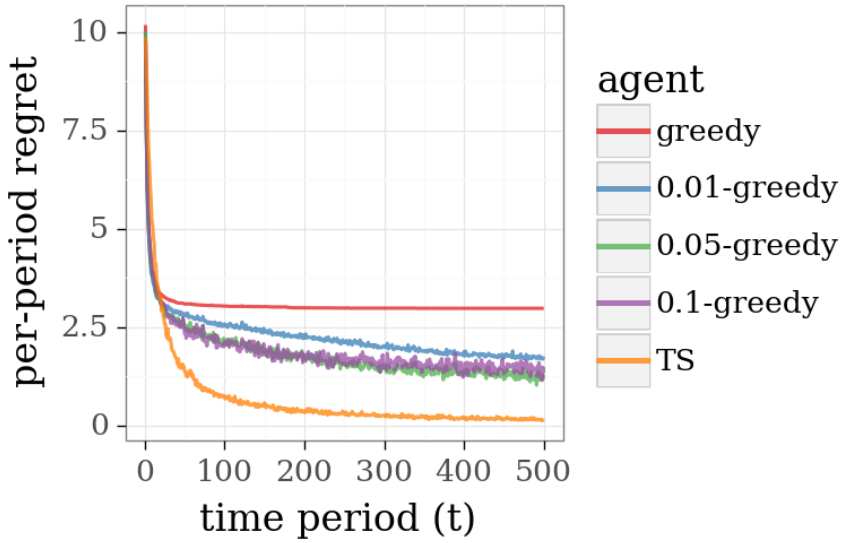
- Expected reward generated by action x under parameter theta

$$\mu(x, \theta) = \mathbb{E}[r\left(g(x, \theta, w_t)\right) \mid \theta]$$

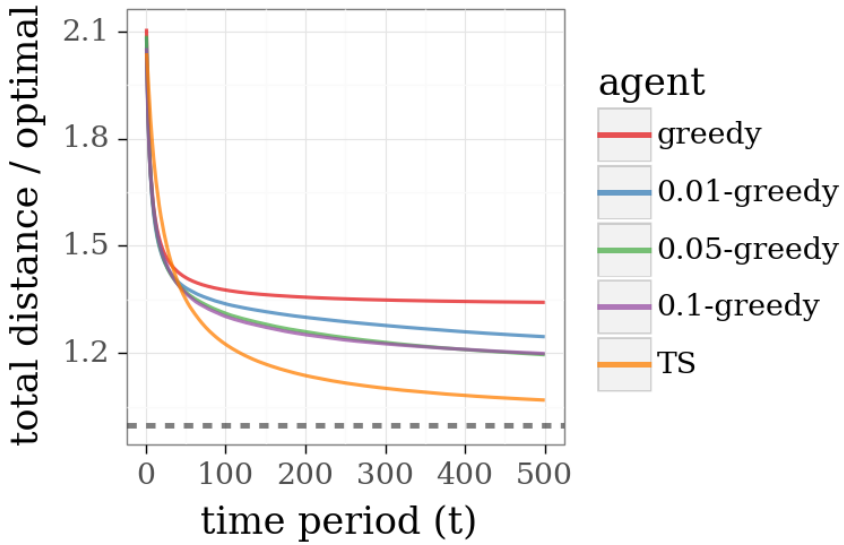
- Expected cumulative regret

$$\mathbb{E}\left[\text{Regret}(T)\right] = \mathbb{E}\left[\sum_{t=1}^{T} \left(\mu(x^*, \theta) - \mu(x_t, \theta)\right)\right]$$

Greedy vs Thompson Sampling



(a) regret



(b) cumulative travel time vs. optimal

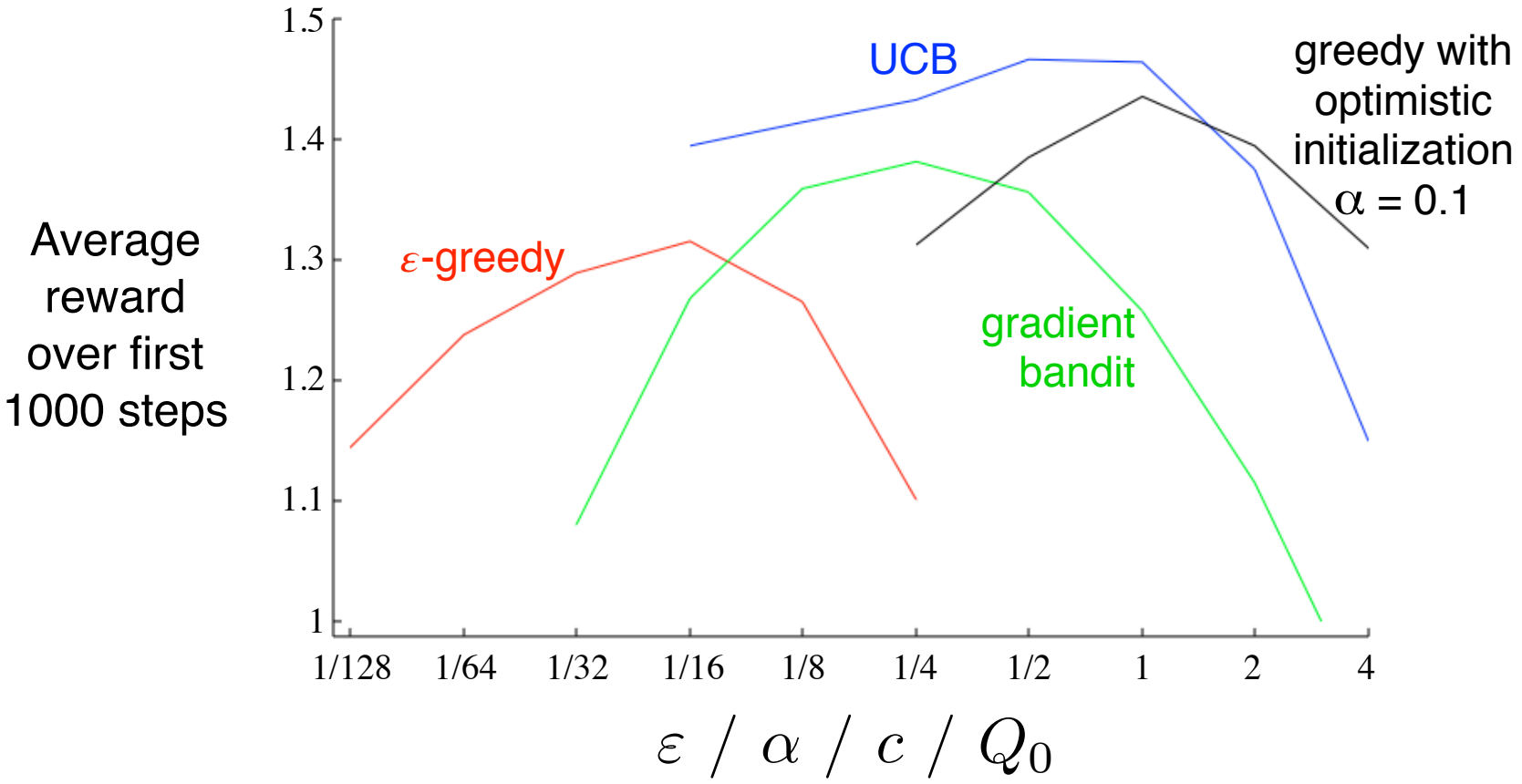
TS and UCB

- Both Thompson Sampling and Upper Confidence Bound algorithms share common motivation
- Explore the actions that our current estimates predict are optimal and be "optimistic" about actions if you have too little information.
- Both UCB and TS optimize Bayesian regret and their bounds can be converted between each other

Limitation of Thompson Sampling

- See TS tutorial section 8.2

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
- 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