Reinforcement Learning – COMS4061A/7071A

Multi-armed Bandits

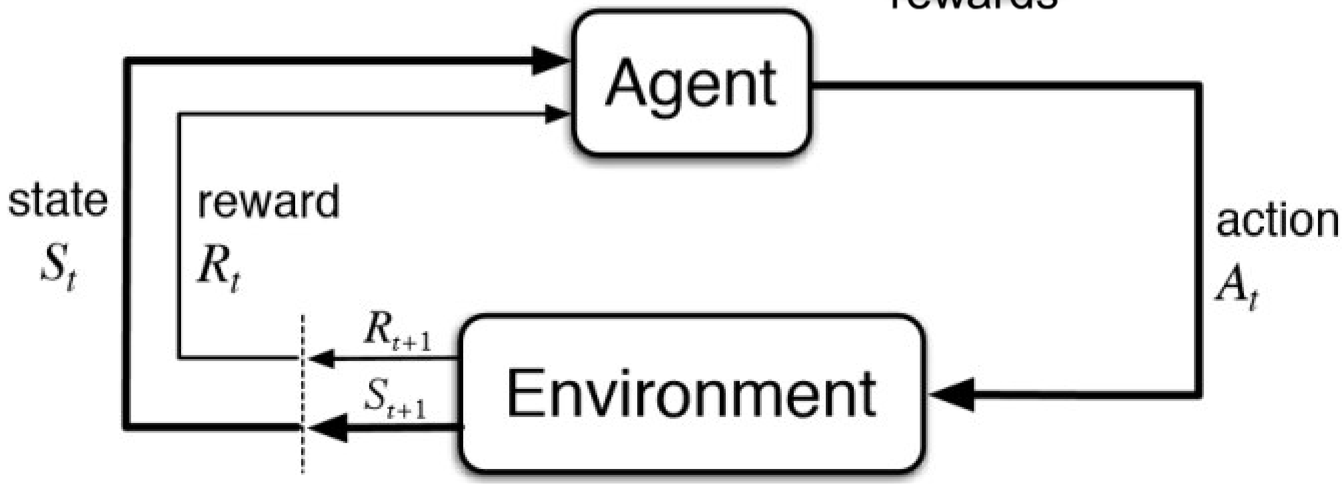
Prof. Benjamin Rosman

Benjamin.Rosman1@wits.ac.za / benjros@gmail.com

Based heavily on slides by Rich Sutton and Doina Precup

The RL problem

From last week:

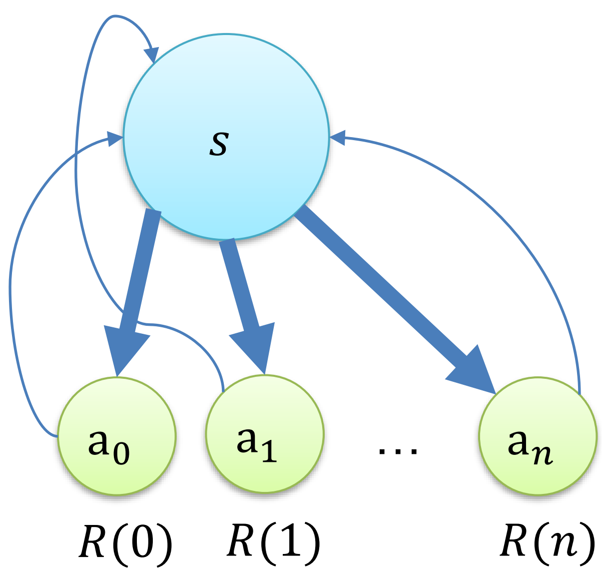


So the RL problem is to find a **policy**: optimal actions a for each state s to maximise total rewards R

The action selection problem

Let's ignore state for now

- Assume there is only one
- We need to keep choosing an action there
- This is the **bandit** problem



What do we need to know to act optimally here?

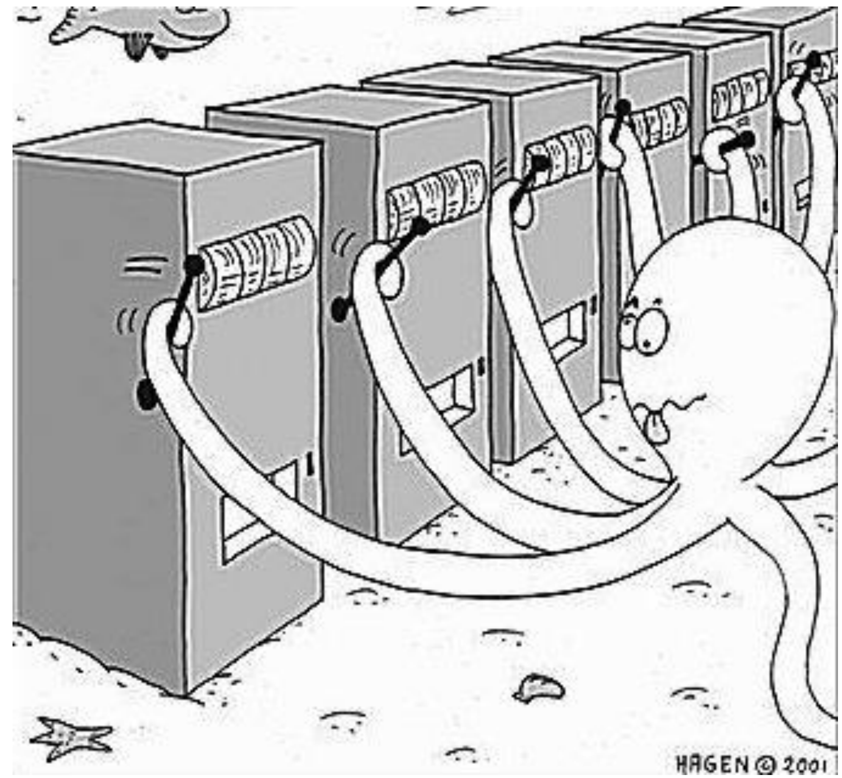
Practically: a restaurant

If you keep visiting a restaurant, how do you make sure you order the best dish?

Multi-armed bandits (MABs)

From a fictional casino:

- There are k different one-armed bandit machines
- Each has a different payoff distribution
- Which one(s) should you play to maximise total payoffs?



Multi-armed bandits (MABs)

?

2

-5

?

-1

2	-1
-5	3
-7	0
0	4
1	-2
-4	-2

Example

- **Action 1**: reward is always 8
- **Action 2**: 88% chance of 0, 12% chance of 100
- **Action 3**: randomly between -10 and 35, equiprobably
- **Action 4**: $\frac{1}{3}$ chance of 0, $\frac{1}{3}$ chance of 20, $\frac{1}{3}$ chance from {8, 9, …, 18}

What to choose if playing repeatedly?

(Hint: what is the expected reward/value of each one?)

Example

- **Action 1**: reward is always 8
  - $q_*(1) = 8$
- **Action 2**: 88% chance of 0, 12% chance of 100
  - $q_*(2) = 0.88 * 0 + 0.12 * 100 = 12$
- **Action 3**: randomly between -10 and 35, equiprobably
  - $q_*(3) = \frac{35-10}{2} = 12.5$
- **Action 4**: $\frac{1}{3}$ chance of 0, $\frac{1}{3}$ chance of 20, $\frac{1}{3}$ chance from {8, 9, …, 18}
  - $q_*(4) = \frac{1}{3} * 0 + \frac{1}{3} * 20 + \frac{1}{3} * 13 = 11$

What to choose if playing repeatedly?

(Hint: what is the expected reward/value of each one?)

The k-armed Bandit Problem

- On each of an infinite sequence of time steps $t = 1,2,\ldots,$ 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 and independently distributed (i.i.d.):

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

- These true values and distributions are *unknown*

- But: maximise your total reward

- So: try actions that **both learn their values (explore), and prefer those that seem best (exploit)**

The Exploration/Exploitation Dilemma

- Suppose you have estimates

$$Q_t(a) \approx q_*(a), \qquad \forall a$$

action-value estimates

- Define the **greedy action** at time t as:

$$A_t^* = \arg\max_a Q_t(a)$$

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

- You need to do both!
- Should you ever stop exploring? Maybe explore less over time?

- But where did the estimates come from?

Action-Value Methods

- Basic idea: learn action value estimates (and nothing else)
- For example: estimate as sample averages:

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

Indicator function

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

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

Number of times action a was 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 one)
- Simplest way to balance exploration and exploitation
- Can also decay ϵ over time

ϵ-Greedy Action Selection

Example algorithm:

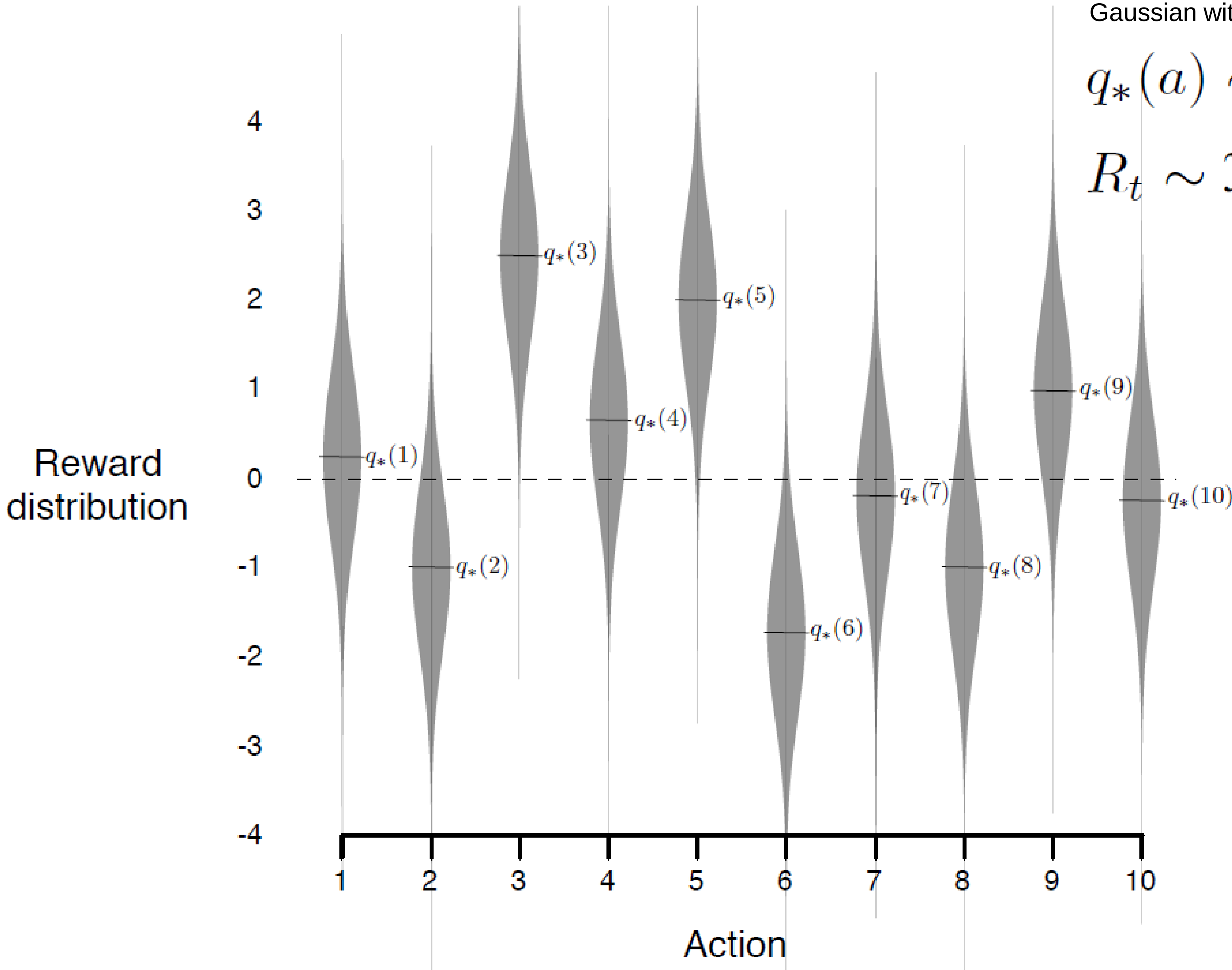
- Repeat forever:
  - $A \leftarrow \begin{cases} \arg\max_a Q(a) & with\ probability\ 1-\epsilon \\ random\ action & with\ probability\ \epsilon \end{cases}$ exploit / explore
  - $R \leftarrow bandit(A)$
  - $Update\ Q(A)\ using\ R$

The 10-armed testbed

Choose random means from a Gaussian for each bandit, and then draw samples from a Gaussian with that mean

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

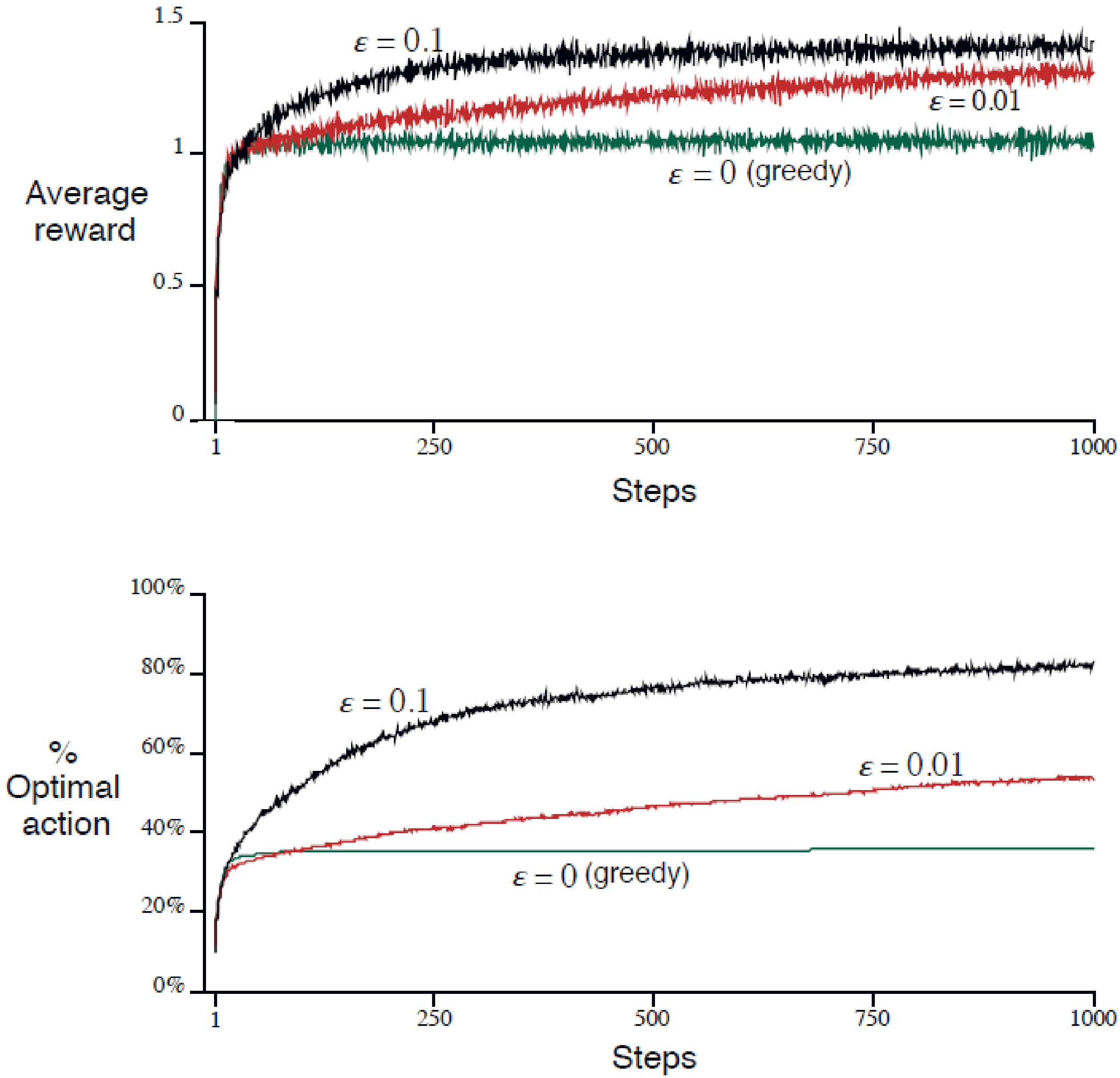
$$R_t \sim \mathcal{N}(q_*(a), 1)$$



Run for 1000 steps

Repeat the whole thing 2000 times with different bandit tasks

ϵ-Greedy methods on the 10-armed testbed



Metrics

Plot as functions of time (number of steps)
Sometimes averaged over a sliding window

- **Reward**
  - The reward R_t obtained at each time step
- **Regret**
  - Loss incurred by not taking the best action in hindsight at each step: $R_{best} - R_t$
  - Maximise reward = minimise regret
- **% optimal action**
  - The fraction of times the optimal action was chosen in the last k steps

Always plot **averages over multiple runs**

Turning averaging into a learning rule

- Focus on estimating the value of one action
- Estimate after $n-1$ rewards:

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

- How to update this incrementally?
- Could store a running sum and a count, or:

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

- Standard form for learning/update rules:

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

Derivation of incremental update

- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$
- $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i$

$$= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right)$$

$$= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right)$$

$$= \frac{1}{n}(R_n + (n-1)Q_n)$$

$$= \frac{1}{n}(R_n + nQ_n - Q_n)$$

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

Non-stationary problems

Non-stationary problem:

- The true action values change slowly over time

Why might sample averages not be a good idea?

Better is exponential, recency-weighted average:

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

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

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

There is bias due to Q_1 that becomes smaller over time

A note on α

- Standard conditions for convergence of stochastic approximations
- To assure convergence with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty$$

And

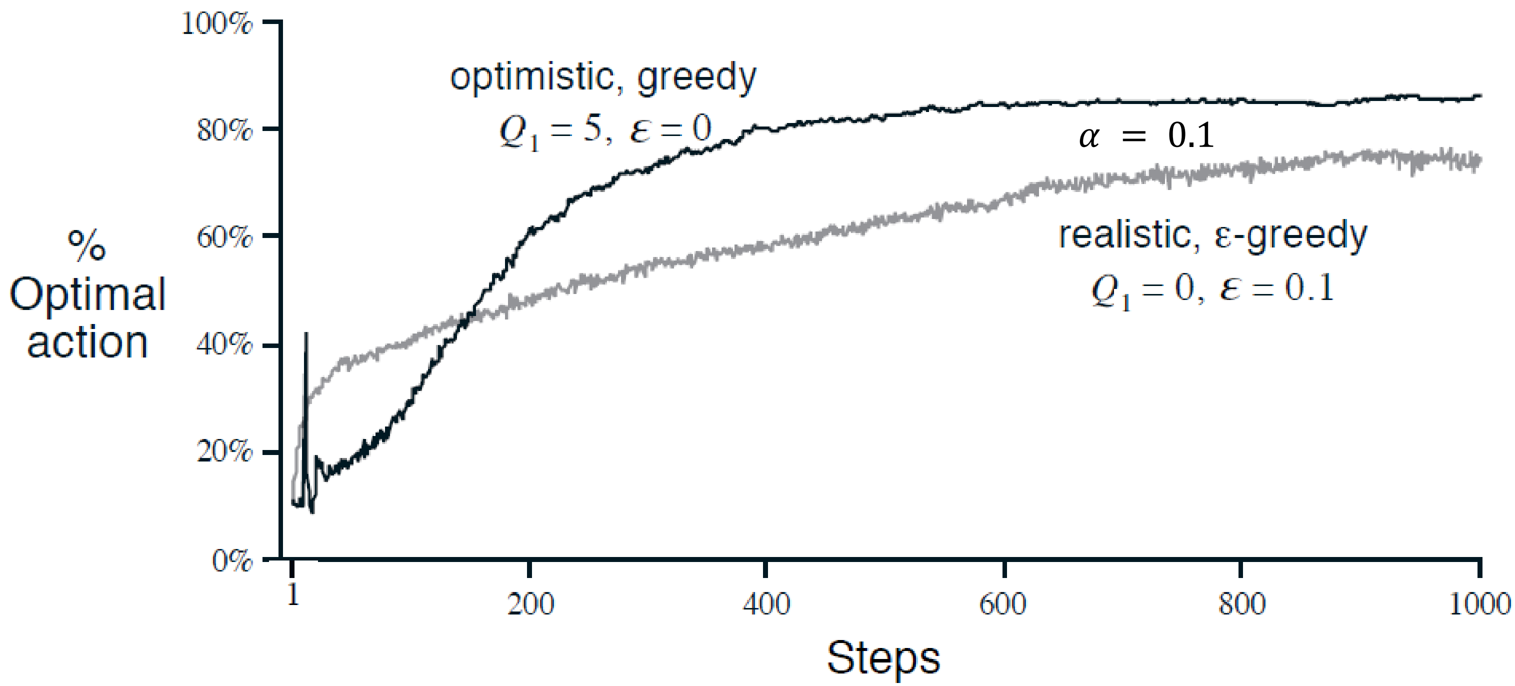
$$\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

Optimistic initial values

What does it mean to be optimistic? Better than anything that could happen

- All these methods depend on $Q_1(a)$
  - These are biased
  - So far, we have used $Q_1(a) = 0$

- We can instead initialise the action values optimistically
  - Why would this work?

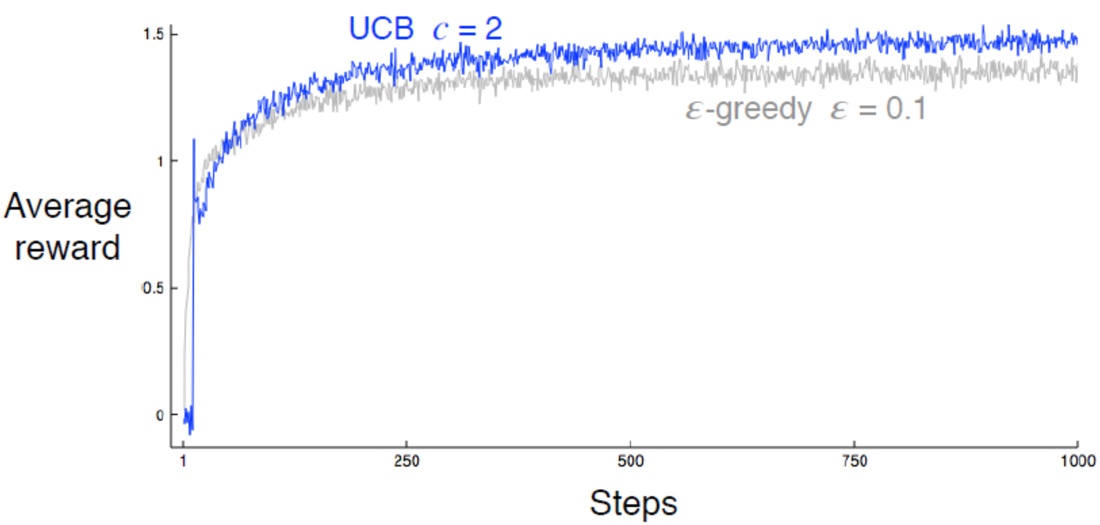


Upper Confidence Bound (UCB) action selection

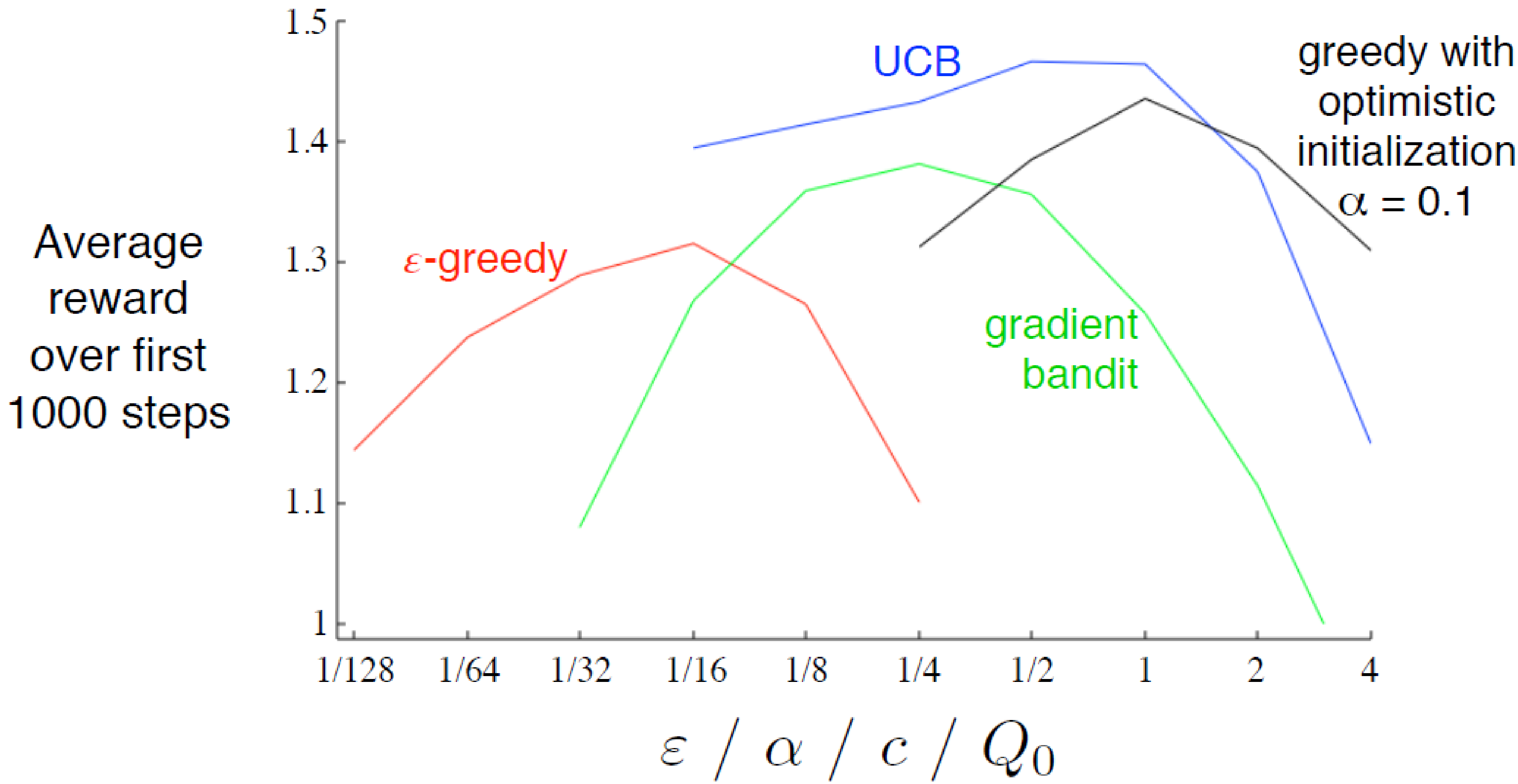
- A different strategy for reducing exploration over time
- Estimate an upper bound on the true action values
- Select the action with the largest (estimated) upper bound

$$A_t = \underset{a}{\arg\max} \left[\underbrace{Q_t(a)}_{\text{“exploit”}} + c \underbrace{\sqrt{\frac{\log t}{N_t(a)}}}_{\text{“explore”}} \right]$$

Trade-off parameter between the two terms



Summary comparison of algorithms



Contextual bandits

- Bandits often used in website settings:
  - Choose adverts, next video to play, …
- Actions: content options
- Rewards: what was clicked on
- Introduce idea of context:
  - Some extra information about e.g. the user
  - Have a different bandit problem for each context vector (user)
  - Or: cluster users
  - Or: do something fancier…

Sits somewhere between bandits and full RL

Exercise

In groups of UP TO FOUR:

1. Implement a MAB:
   - Let each arm give rewards from a Gaussian of variance 1, and means drawn from a Gaussian of mean 0, variance 3 when they are created.
   - You should be able to "pull" an arm (select an action) and receive a random reward.
2. Implement the ϵ-greedy, greedy with optimistic initialisation, and UCB algorithms.
3. Run the three algorithms with different parameter settings on a 10-arm bandit.

By next week's lecture, submit on Moodle:

1. A plot of reward over time (averaged over 100 runs each) on the same axes, for ϵ-greedy with $\epsilon = 0.1$, greedy with $Q_1 = 5$, and UCB with $c = 2$
2. A summary comparison plot of rewards over first 1000 steps for the three algorithms with different values of the hyperparameters
3. Your code