Applied Computational Mathematics Lab

TEDDY LAZEBNIK'S COURSE · LECTURE 2

Reinforcement Learning

Lecture 2 · Exploration and exploitation: the multi-armed bandit, regret, ε-greedy, UCB, Thompson sampling and gradient bandits



Teddy Lazebnik · Applied Computational Mathematics Laboratory

BEFORE WE START

Recap, and what this lecture does

- RL is the science of learning to make decisions.
- An agent can learn a policy, a value function and/or a model.
- In general we must think about time and consequences: actions change the reward, the agent state and the environment state.
- Learning is **active**: decisions change the data we see.



IN PLAIN WORDS

Today we strip RL down to its simplest form: one state, many actions. Everything about exploration is already there.

READING

Sutton & Barto (2018), chapter 2. For more theory: Lattimore & Szepesvári, *Bandit Algorithms* (2020); Auer, Cesa-Bianchi & Fischer (2002).

BEFORE WE START

Today's map

PART 1

The bandit problem

One state, many arms; values and regret.

PART 2

Learning action values

Averages, step sizes, and the algorithms we will meet.

PART 3

Greedy and ε-greedy

Why greedy gets stuck, and the simplest fix.

PART 4

Gradient bandits

Learning the policy directly, with the log-likelihood trick.

PART 5

How well can we do?

A lower bound on regret, and optimism under uncertainty.

PART 6

UCB, Thompson sampling and beyond

Confidence bounds, beliefs, and planning to explore.



IN PLAIN WORDS

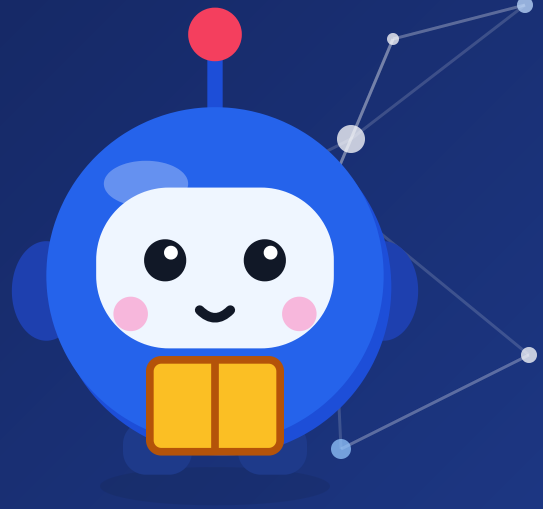
By the end you should be able to explain why exploration is needed, compute regret, and run greedy, ε-greedy, UCB and Thompson sampling by hand.

Applied Computational Mathematics Lab

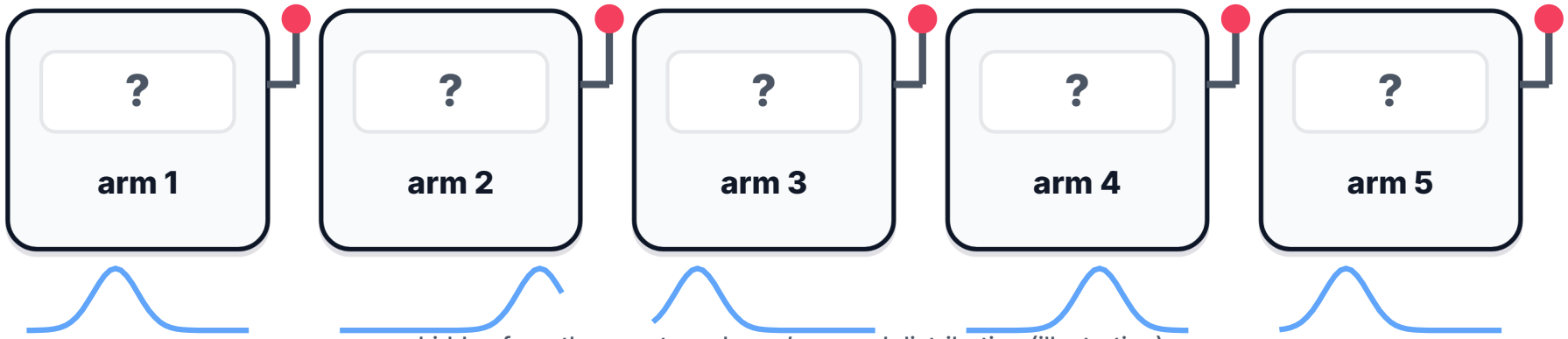
PART 1

The bandit problem

One state, many actions: the simplest setting where exploration matters.



One state, many actions



hidden from the agent: each arm's reward distribution (illustration)

- We assume the environment has a **single state**.
- So actions have **no long-term consequences**: nothing you do now changes what happens later.
- Actions still change the **immediate reward**, and other observations can be ignored.



IN PLAIN WORDS

A row of slot machines. Pulling one handle never moves you anywhere; it just pays out something. The only question is which handle to pull.

Exploration and exploitation

EXPLOIT

Use what you know

Pick the action that looks best right now, to maximise performance based on current knowledge.

EXPLORE

Learn something new

Try actions that might be better, to increase knowledge.



IN PLAIN WORDS

We must gather information to make the best decisions overall. The best long-term strategy may need **short-term sacrifices**.

RESTAURANTS

Go to your favourite place again, or try the new one down the street?

ONLINE ADS

Show the ad that worked yesterday, or test a new one?

LEARNING A FIELD

Keep reading new sources, or start writing with what you know?

ANTS

Wander at random until one finds food; then the colony follows her trail.

The multi-armed bandit

MULTI-ARMED BANDIT

A multi-armed bandit is a set of reward distributions $\{\mathcal{R}_a \mid a \in \mathcal{A}\}$, one for each action ("arm") in a known set $\mathcal{A}$.

- At each step t the agent picks an action $A_t \in \mathcal{A}$.
- The environment draws a reward $R_t \sim \mathcal{R}_{A_t}$.
- Goal: maximise the cumulative reward $\sum_{i=1}^{t} R_i$.
- We do this by learning a **policy**: a distribution over $\mathcal{A}$.

WHERE THE NAME COMES FROM

The name comes from old slot machines, nicknamed "one-armed bandits". A machine with many handles is a **multi**-armed bandit.



IN PLAIN WORDS

The bandit is a **problem**, not an algorithm. It is the simplest RL problem there is, so most exploration ideas are tested on it first.

Values and regret

- The **action value** of a is its expected reward:

$$q(a) = \mathbb{E}\left[R_t \mid A_t = a\right]$$

- The **optimal value** is the best of them:

$$v_* = \max_{a \in \mathcal{A}} q(a)$$

- The **regret** of an action is how much worse it is than the best:

$$\Delta_a = v_* - q(a)$$

TRY IT

Three arms with $q = 0.2,\ 0.5,\ 0.9$.
Then $v_* = 0.9$, and the regrets are $\Delta = 0.7,\ 0.4,\ 0$.
The best arm always has zero regret.



IN PLAIN WORDS

Regret uses the true values, which the agent does not know. It is a score for us, the designers, not a signal the agent can use.

Total regret

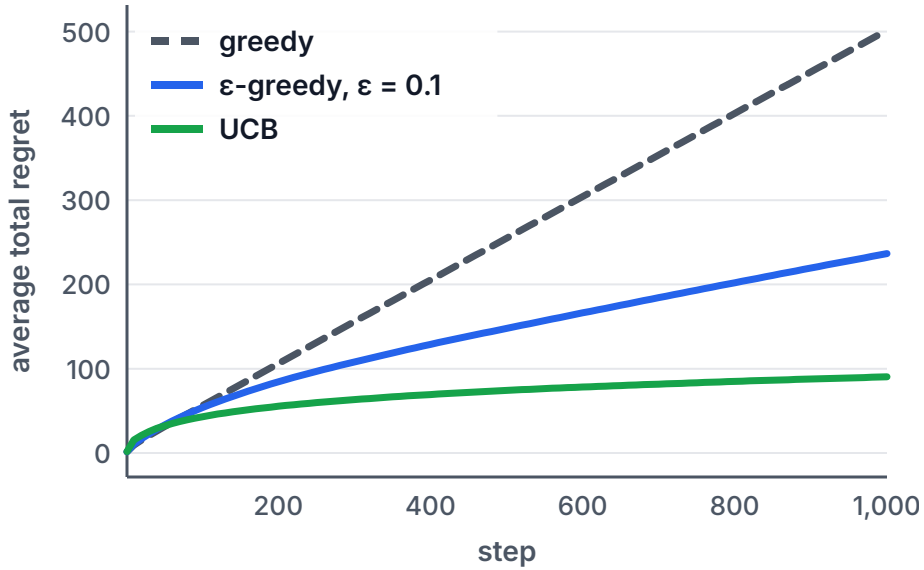
- We want to minimise the **total regret** over the whole lifetime of learning:

$$L_t = \sum_{n=1}^{t} (v_* - q(A_n)) = \sum_{n=1}^{t} \Delta_{A_n}$$

- Maximising cumulative reward is the same as minimising total regret.
- Every pull counts, including the ones spent learning.

TRY IT

With the arms above, an agent pulls arm 1 twice, arm 2 three times and arm 3 five times. Its total regret is $2(0.7) + 3(0.4) + 5(0) = 2.6$.



Our run: 10-armed testbed, average over 2000 runs. Steep while exploring, flatter once the agent knows; a good learner keeps the steep part short.

Check yourself: Part 1

CHECK YOURSELF

1. Why can a single-state problem still need exploration?

CHECK YOURSELF

2. Arms have $q = 1, 3, 4$. What are v_* and the regret of each arm?

CHECK YOURSELF

3. An agent pulls the arm with $q = 3$ ten times. What is its total regret?

CHECK YOURSELF

4. Why can't the agent compute its own regret while learning?





HINT

Answers are at the end of the deck.

Applied Computational Mathematics Lab

PART 2

Learning action values

Estimating how good each arm is, one pull at a time.



Estimating values by averaging

- The true value $q(a) = \mathbb{E}[R_t \mid A_t = a]$ is unknown.
- A simple estimate is the average of the rewards seen for a:

$$Q_t(a) = \frac{\sum_{n=1}^{t} \mathbb{I}(A_n = a)\, R_n}{\sum_{n=1}^{t} \mathbb{I}(A_n = a)}, \qquad N_t(a) = \sum_{n=1}^{t} \mathbb{I}(A_n = a)$$

- $\mathbb{I}(\cdot)$ is the indicator function: 1 if true, 0 if false. $N_t(a)$ counts the pulls of a.

STEP	ARM	REWARD	$Q(\mathrm{A})$	$Q(\mathrm{B})$
1	A	2	2	–
2	B	0	2	0
3	A	4	3	0
4	A	3	3	0
5	B	1	3	0.5

A worked example with two arms. Each estimate only changes when its own arm is pulled.



IN PLAIN WORDS

By the law of large numbers, the average settles on the true value if you pull the arm often enough.

Updating one pull at a time

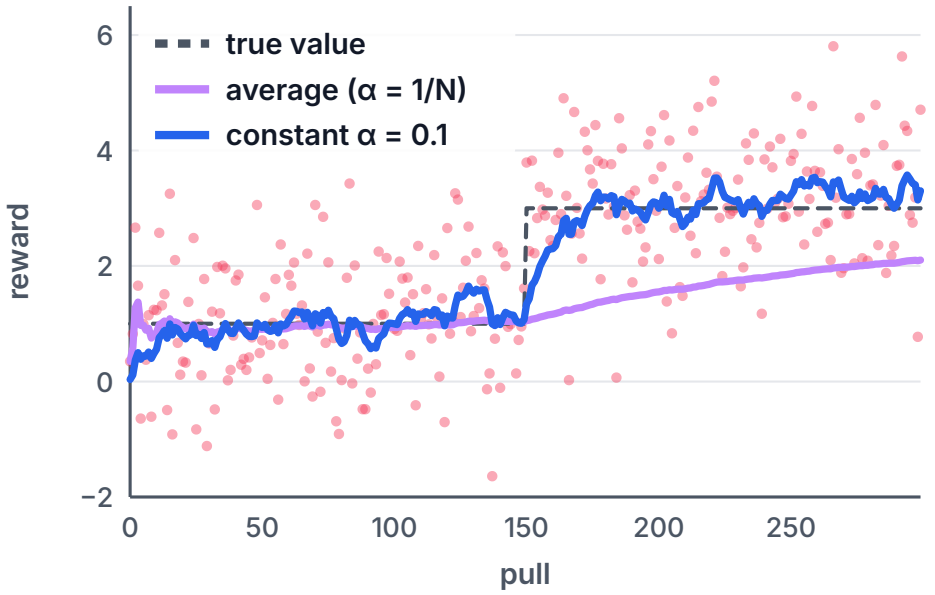
- The average can be updated incrementally, without storing past rewards:

$$Q_t(A_t) = Q_{t-1}(A_t) + \alpha_t \big(\underbrace{R_t - Q_{t-1}(A_t)}_{\text{error}}\big)$$

- with $\alpha_t = 1/N_t(A_t)$ and $N_t(A_t) = N_{t-1}(A_t) + 1$, starting from $N_0(a) = 0$. Other arms keep their estimates.
- A **constant** step size α gives **tracking** instead of averaging: recent rewards count more.

TRY IT

$Q = 3$ after 2 pulls; the third pull pays 6. Then $Q \leftarrow 3 + \frac{1}{3}(6 - 3) = 4$, the average of all three.



Our run: an arm whose true value jumps from 1 to 3. The average reacts slowly; a constant step size tracks the change.

The algorithms we will meet

USES Q

Greedy

Always pick the arm with the highest estimate.

USES Q

ε-greedy

Greedy, but sometimes pick at random.

USES Q

UCB

Pick the highest estimate plus an uncertainty bonus.

USES BELIEFS

Thompson sampling

Keep a belief per arm; act on a random draw from each.

USES A POLICY

Policy gradients

Learn action preferences directly by gradient ascent.

One shared loop

Pull an arm, observe the reward, update. Only the choice and the update change.



IN PLAIN WORDS

The first three all use action-value estimates $Q_t(a) \approx q(a)$. They differ only in **how much** and **where** they explore.

Applied Computational Mathematics Lab

PART 3

Greedy and ε-greedy

The simplest policy, why it gets stuck, and the simplest fix.



The greedy policy

- Pick the arm with the highest current estimate:

$$A_t = \operatorname*{argmax}_a Q_t(a)$$

- Equivalently $\pi_t(a) = \mathbb{I}(a = \operatorname{argmax}_b Q_t(b))$, assuming no ties.
- It never explores on purpose.



IN PLAIN WORDS

Greedy is not stupid; it is confident too early.

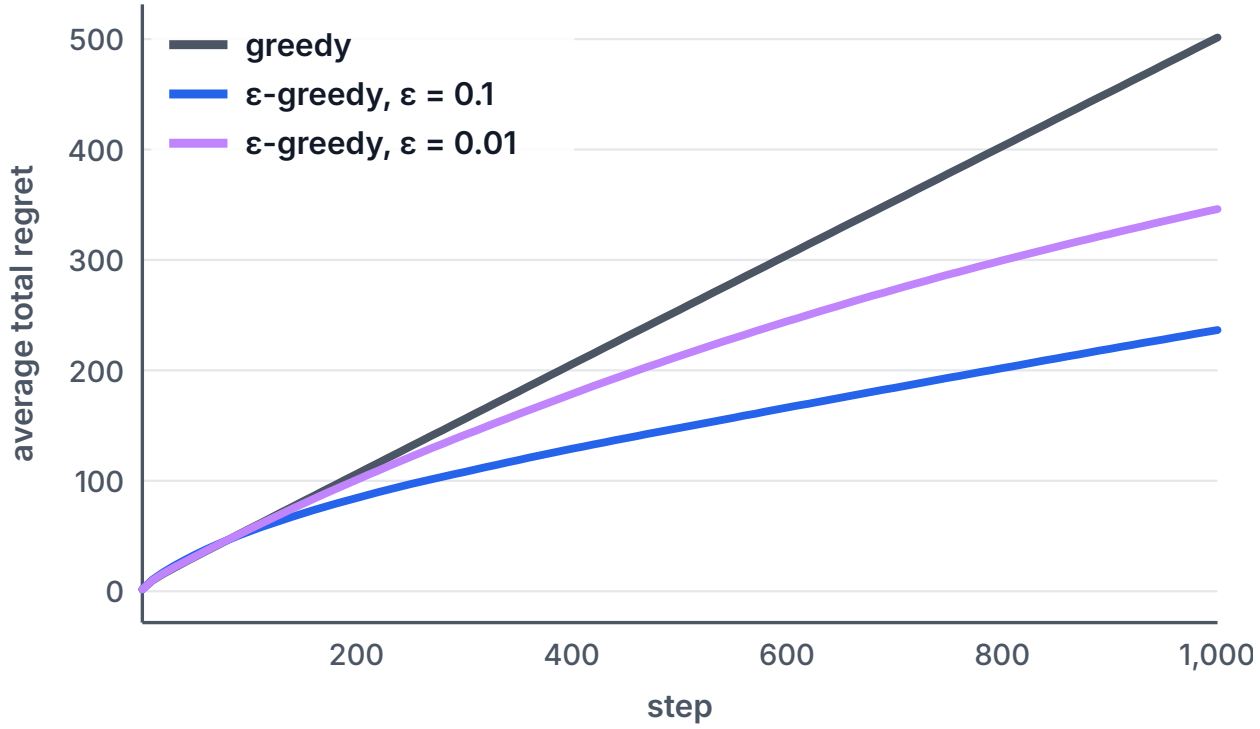
HOW GREEDY GETS STUCK

Two arms. Arm A has true value 1, arm B has true value 2. Both are pulled once to start.
Arm B is unlucky on its first pull and pays −0.5; arm A pays 1.2.
Now $Q(\mathrm{A}) = 1.2 > Q(\mathrm{B}) = -0.5$, so greedy pulls A, and keeps pulling A. B's estimate never changes, so greedy **never finds out** that B is better.

TRY IT

Every pull of A costs regret $\Delta_{\mathrm{A}} = 2 - 1 = 1$.
After 1000 pulls the total regret is about 1000: it grows in a straight line.

Greedy in practice: our run



Total regret after 1000 steps: greedy 501, ε = 0.01: 346, ε = 0.1: 237.

THE 10-ARMED TESTBED (SUTTON & BARTO, CH. 2)

Ten arms. Each arm's true value is drawn from a normal distribution with mean 0 and spread 1; each reward is the arm's value plus normal noise. 1000 steps, 2000 runs averaged.

- Greedy settles early on whatever looked best first.
- It ends on the truly best arm in only 35% of runs (our run).
- ε-greedy keeps learning and pulls ahead.

The ε-greedy algorithm

- With probability $1 - \epsilon$: pick the greedy arm.
- With probability ϵ: pick an arm uniformly at random.
- Equivalently:

$$\pi_t(a) = \begin{cases} (1-\epsilon) + \epsilon/|\mathcal{A}| & \text{if } Q_t(a) = \max_b Q_t(b) \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$$



IN PLAIN WORDS

The random pull is what saves you from getting stuck. ε is the knob: 0 is plain greedy, 1 is pure chance.

TRY IT

Ten arms and $\epsilon = 0.1$: the greedy arm is picked with probability $0.9 + 0.1/10 = 0.91$; each other arm with 0.01.

WHY IT MATTERS

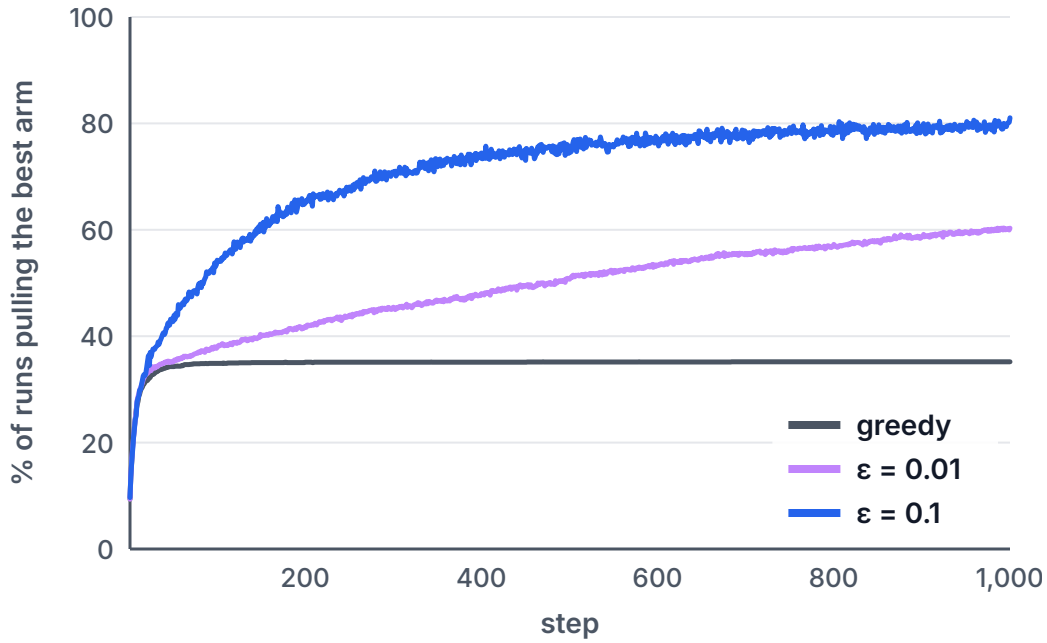
Most RL algorithms in use today have ε-greedy somewhere inside: it is simple, and it works.

IN PRACTICE

Start with a large ε and shrink it over time: explore early, exploit later. The right schedule depends on the problem.

Is ε-greedy enough?

- With a **constant** ε the agent keeps exploring forever, so it keeps paying regret: total regret still grows **linearly**, just more gently than greedy.
- A larger ε learns faster at first, but wastes more pulls later.
- If ε shrinks at the right speed, regret can grow much more slowly, but the right speed depends on gaps we do not know.



Our run, 10-armed testbed, 2000 runs. ε = 0.1 finds the best arm fastest but can never pull it more than 91% of the time.



IN PLAIN WORDS

We would like exploration that fades by itself as we learn. That is what UCB and Thompson sampling will give us.

Check yourself: Parts 2 and 3

CHECK YOURSELF

5. Rewards for one arm are 4, 2, 6. Compute Q after each pull using $\alpha = 1/N$.

CHECK YOURSELF

6. When would you prefer a constant step size α to averaging?

CHECK YOURSELF

7. Five arms and $\epsilon = 0.2$. What is the probability of picking the greedy arm?

CHECK YOURSELF

8. Why does greedy have linear total regret?



Applied Computational Mathematics Lab

PART 4

Gradient bandits

Learning the policy directly, without value estimates.



Learning a policy directly

- Instead of learning values, can we learn $\pi(a)$ itself?
- Give each arm a **preference** $H_t(a)$ and turn preferences into probabilities with a softmax:

$$\pi(a) = \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}}$$

- Preferences are **not values**: they are just policy parameters to learn.
- Goal: adjust the preferences so the expected reward goes up.

TRY IT

Preferences $H = (1, 0, 0)$ for three arms. Then $e^1 = 2.72$ and $e^0 = 1$, so $\pi = (2.72, 1, 1)/4.72 = (0.58, 0.21, 0.21)$.



IN PLAIN WORDS

Rewards live on any scale; probabilities are always between 0 and 1. The softmax is the same one used in classification networks.

Climbing the gradient

- Update the policy parameters θ so that the expected reward increases, by gradient **ascent**:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \mathbb{E}\left[R_t \mid \pi_{\theta_t}\right]$$

- Plus, not minus: we maximise reward, so we walk uphill.
- But can we compute this gradient? The expectation depends on $q(a)$, which we don't know.

WHY THE UPHILL DIRECTION HELPS

Two arms at 50/50, with $q = 1$ and $q = 3$. The expected reward is $0.5 \cdot 1 + 0.5 \cdot 3 = 2$. Move 10% of the probability to the second arm: $0.4 \cdot 1 + 0.6 \cdot 3 = 2.2$. The gradient points toward the better arm.



IN PLAIN WORDS

We want to know which way is uphill without knowing the map. The next slide shows we can sample it, one pull at a time.

The log-likelihood trick

$$\nabla_\theta \mathbb{E}[R_t \mid \pi_\theta] = \nabla_\theta \sum_a \pi_\theta(a)\, q(a) = \sum_a q(a)\, \nabla_\theta \pi_\theta(a)$$

$$= \sum_a \pi_\theta(a)\, q(a)\, \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} = \mathbb{E}\left[R_t\, \nabla_\theta \log \pi_\theta(A_t)\right]$$

- The last line is an expectation of something we can **observe**: the reward times the gradient of the log-probability of the arm we pulled.
- So we can sample it, and take a stochastic gradient step:

$$\theta \leftarrow \theta + \alpha\, R_t\, \nabla_\theta \log \pi_\theta(A_t)$$

WHY THE DIVISION

Why divide by $\pi_\theta(a)$? We only see the arms we happen to pull. An arm pulled with probability 0.1 shows up about once in 10 steps, so each time it does, its term is scaled up by $1/0.1 = 10$. On average, every arm is then counted in full.



IN PLAIN WORDS

Also called the REINFORCE trick (Williams, 1992). It needs no value estimates, only sampled rewards.

THE CATCH

The price is noise: each step uses one reward, so the steps jump around. That is stochastic gradient ascent on the true value of the policy.

Gradient bandits with a softmax

- For the softmax policy the update becomes:

$$H_{t+1}(a) = H_t(a) + \alpha\, R_t\, (\mathbb{I}(a = A_t) - \pi_t(a))$$

- The pulled arm goes up: $H(A_t) \mathrel{+}= \alpha R_t(1 - \pi_t(A_t))$.
- Every other arm goes down: $H(a) \mathrel{-}= \alpha R_t\, \pi_t(a)$.
- Arms with higher rewards rise more (or fall less), so they get picked more often.

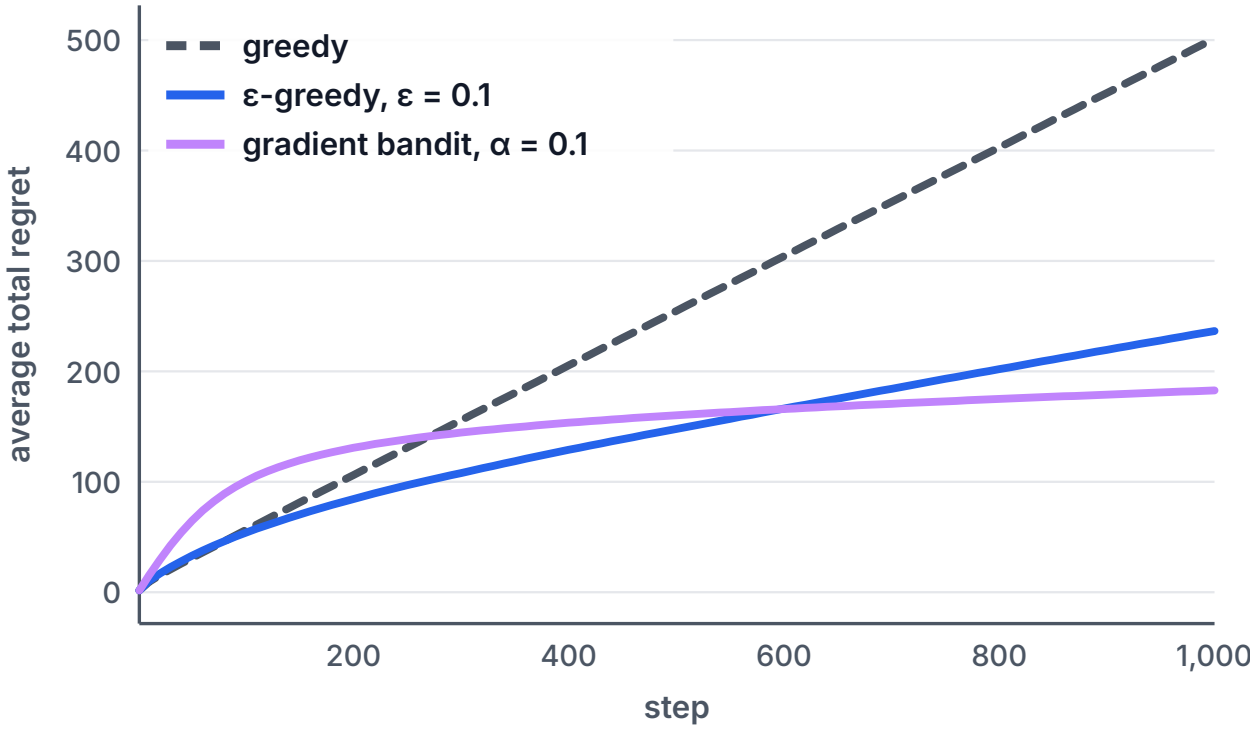
TRY IT

Three arms, all $H = 0$, so $\pi = 1/3$ each. You pull arm 2 and get $R = 1$, with $\alpha = 0.1$.
Arm 2: $0 + 0.1(1)(1 - 1/3) = +0.067$.
Arms 1 and 3: $0 - 0.1(1)(1/3) = -0.033$.

A USEFUL EXTRA

In practice we subtract a **baseline**, the average reward so far, from R_t. It doesn't change the expected direction, but it makes the steps much less noisy (Sutton & Barto, section 2.8). Our runs use it.

Gradient bandits: our run



Our run: 10-armed testbed, 2000 runs, with a baseline. Total regret after 1000 steps: gradient bandit 183, ε-greedy 237, greedy 501.

- The gradient bandit explores through its own probabilities: early on they are spread out, later they concentrate.
- Its regret curve bends: the policy becomes nearly deterministic once one arm clearly wins.



IN PLAIN WORDS

Three ways to explore so far: not at all (greedy), at random (ε-greedy), or by following the gradient of your own policy.

Applied Computational Mathematics Lab

PART 5

How well can we do?

A lower bound on regret, and the idea of optimism in the face of uncertainty.



A lower bound on regret

THEOREM (LAI & ROBBINS, 1985)

In the long run, the total regret of any algorithm grows at least logarithmically in the number of steps:

$$\lim_{t\to\infty} L_t \geq \log t \sum_{a|\Delta_a>0} \frac{\Delta_a}{\mathrm{KL}(\mathcal{R}_a \,\|\, \mathcal{R}_{a^*})}$$

- KL measures how different an arm's reward distribution is from the best arm's; it grows like Δ_a^2.
- Arms that are similar to the best are hard to tell apart, and cost more regret.



IN PLAIN WORDS

Nobody can be perfect: some regret is the unavoidable price of learning. But logarithmic is far better than linear.

t	LINEAR t	$\log t$
10	10	2.3
1,000	1,000	6.9
1,000,000	1,000,000	13.8

Why logarithmic growth is so much better.



IN PLAIN WORDS

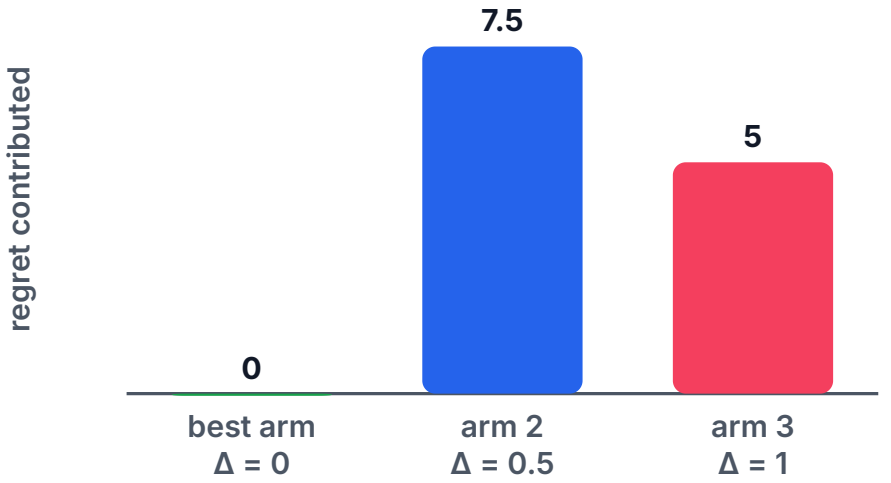
Two questions follow: can we reach this in practice, and are there algorithms whose regret is also bounded **above** by a logarithm?

Counting regret

- Recall $\Delta_a = v_* - q(a)$. Total regret can be counted per arm:

$$L_t = \sum_{n=1}^{t} \Delta_{A_n} = \sum_{a \in \mathcal{A}} N_t(a)\, \Delta_a$$

- Each arm contributes its gap times how often it was pulled.
- A good algorithm keeps the counts **small** for arms with **large** gaps.



TRY IT

Counts after 100 pulls: 80, 15 and 5. Regret $= 80(0) + 15(0.5) + 5(1) = 12.5$. The bad arm was pulled rarely, so it cost little.

Optimism in the face of uncertainty



Illustration: the agent's beliefs about three arms.

- Arm 1 looks best on average, and we are fairly sure about it.
- Arm 3 looks worse on average, but we know very little about it: it might be the best.
- Which arm should we pick?



IN PLAIN WORDS

The more uncertain we are about an arm, the more important it is to **explore** it. Don't pick what looks great; pick what **could** be great.

WHY OPTIMISM IS SAFE

If we try arm 3 and it is bad, its belief narrows and moves left, and we stop picking it. If it is good, we found something. Either way we learn.

Applied Computational Mathematics Lab

PART 6

UCB, Thompson sampling and beyond

Confidence bounds, beliefs, and planning to explore.

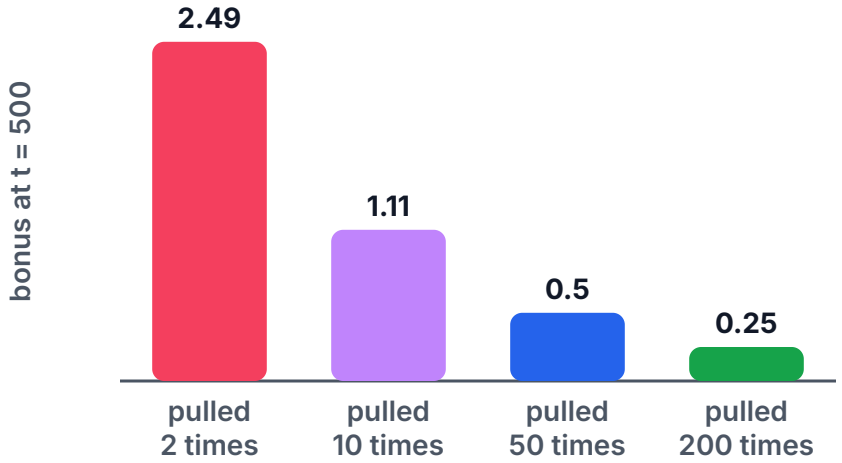


Upper confidence bounds

- For each arm, estimate an upper confidence $U_t(a)$, so that $q(a) \leq Q_t(a) + U_t(a)$ with high probability.
- Pick the arm with the highest upper bound:

$$A_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; Q_t(a) + U_t(a)$$

- The bonus should depend on the count $N_t(a)$: small count, large bonus; large count, small bonus.
- So an arm is chosen if it looks good, **or** if we are unsure about it, or both.



The UCB bonus $\sqrt{2}\sqrt{\log t / N_t(a)}$ at step 500 for different counts.



IN PLAIN WORDS

Notice what changed: we no longer maximise the estimate, but an upper bound on it.

Hoeffding's inequality

THEOREM (HOEFFDING)

Let $X_1, \ldots, X_n$ be independent samples in $[0, 1]$ with mean μ, and $\bar{X}_n$ their average. Then for any $u > 0$

$$p\left(\bar{X}_n + u \leq \mu\right) \leq e^{-2nu^2}$$

- For a bandit with rewards in $[0, 1]$:

$$p\left(Q_t(a) + U_t(a) \leq q(a)\right) \leq e^{-2N_t(a)\,U_t(a)^2}$$

- By symmetry, the same bound holds for the estimate being too **high**.



IN PLAIN WORDS

The chance that the average plus a bonus still falls below the truth shrinks **exponentially**, both in the number of samples and in the bonus squared.

TRY IT

A coin with unknown win rate, flipped $n = 100$ times. The chance that the true rate is more than $u = 0.1$ above your average is at most $e^{-2 \cdot 100 \cdot 0.01} = e^{-2} \approx 0.14$.

Calculating the bound

- Pick a small probability p that the truth exceeds the bound, and solve for $U_t(a)$:

$$e^{-2N_t(a)U_t(a)^2} = p \quad \Longrightarrow \quad U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

- Then the chance of being wrong is at most p.
- Shrink p as we observe more, for example $p = 1/t$:

$$U_t(a) = \sqrt{\frac{\log t}{2N_t(a)}}$$

- This keeps exploring forever, but less and less.

TRY IT

At $t = 1000$, how many pulls until the bonus is 0.1? Solve $\sqrt{\log 1000/(2N)} = 0.1$: $N = 6.91/0.02 \approx 345$ pulls.
For a bonus of 0.01: about 34,500 pulls.



IN PLAIN WORDS

Time ticks on by itself, and you control the counts. Precision is expensive: ten times more precision costs a hundred times more pulls.

The UCB algorithm

$$A_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}}$$

- c is a hyper-parameter that sets how much to explore.
- If Δ_a is large, then $N_t(a)$ stays small, because $Q_t(a)$ is likely to be small.
- So for every arm, either the gap is small or the count is small. One can prove $\Delta_a N_t(a) \leq O(\log t)$.

THEOREM (AUER, CESA-BIANCHI & FISCHER, 2002)

UCB with $c = \sqrt{2}$ has logarithmic expected total regret:

$$L_t \leq 8 \sum_{a \mid \Delta_a > 0} \frac{\log t}{\Delta_a} + O\Big(\sum_a \Delta_a\Big)$$

ARM	Q	N	BONUS	UCB
A	0.60	50	0.45	1.05
B	0.70	100	0.32	1.02
C	0.40	5	1.42	**1.82**

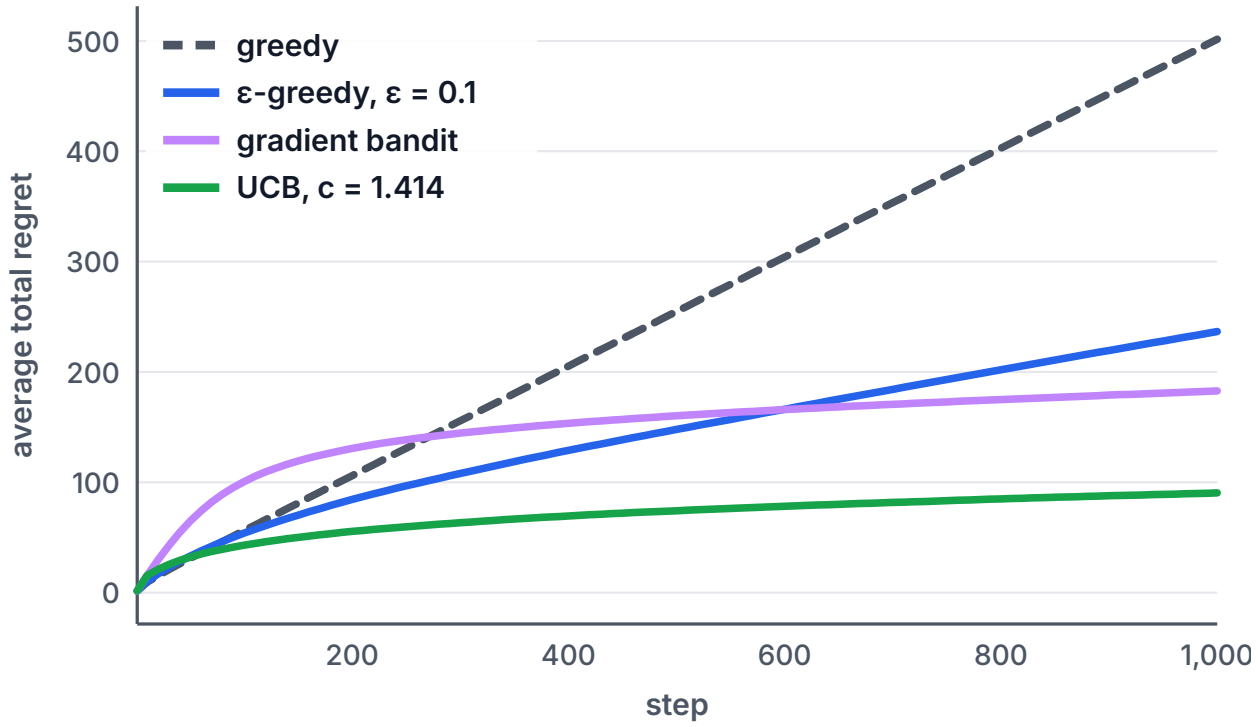
At $t = 155$ with $c = \sqrt{2}$: arm C has the lowest estimate but has barely been tried, so UCB picks it.



IN PLAIN WORDS

UCB explores a lot at first, then less and less, without anyone setting a schedule.

UCB against ε-greedy: our run



Our run, 10-armed testbed, 2000 runs. Total regret after 1000 steps: UCB 90, gradient bandit 183, ε-greedy 237, greedy 501.

- ε-greedy: a straight line.
- UCB: the curve bends and flattens, as the theory promises.
- Same arms, same runs: only the algorithm changed.



IN PLAIN WORDS

Should you always add UCB to your agent? You can't know in advance. In large problems, counts are hard to define; try it with and without.

Keeping a belief

- A Bayesian agent keeps a **distribution** over each arm's value, $p(q(a) \mid \theta_t)$: how much it believes each possible value.
- For example, θ_t could hold the mean and variance of a Gaussian belief per arm.
- We can start from **prior** knowledge θ_0, and use the **posterior** to guide exploration.



IN PLAIN WORDS

One number throws away how sure you are. A belief keeps it.

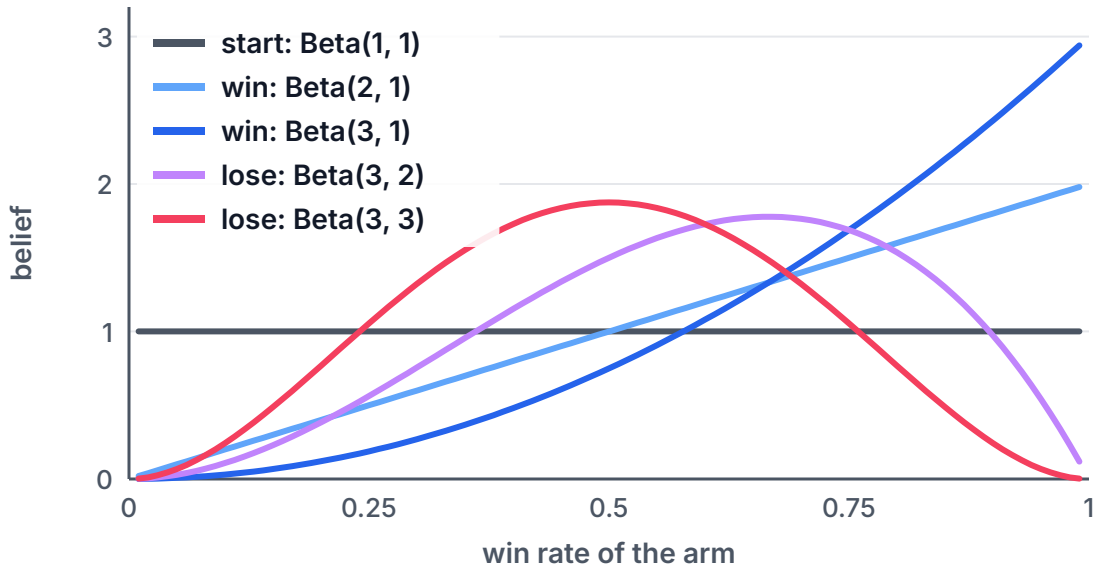
AN OPTIMISTIC CHOICE FROM BELIEFS

Bayesian UCB: use the spread of the belief as the bonus. With belief mean $\mu_t(a)$ and standard deviation $\sigma_t(a)$, pick the arm maximising $\mu_t(a) + c\,\sigma_t(a)$.

TRY IT

Arm A: mean 0.6, spread 0.05. Arm B: mean 0.5, spread 0.2. With $c = 2$: A scores 0.70, B scores 0.90. Pick B: it might be much better.

Win-or-lose arms and the Beta belief



Our computed curves for the sequence win, win, lose, lose.

- Each arm pays 1 (win) or 0 (lose). The unknown win rate is between 0 and 1.
- Start flat: every win rate equally likely, Beta(1, 1).
- The belief stays a Beta: add 1 to the first number for a win, to the second for a loss.



IN PLAIN WORDS

After two wins and two losses the counts look balanced, as at the start, but the belief is now a hump around 0.5. It remembers how much it has seen.

CERTAINTY

Same average, different certainty: 2 wins in 4 pulls gives a wide hump; 200 wins in 400 a narrow spike.

Probability matching

- Another idea: pick each arm with the probability that it is the **best** one, given everything seen so far:

$$\pi_t(a) = p\left(q(a) = \max_{a'} q(a') \,\middle|\, \mathcal{H}_{t-1}\right)$$

- This is also optimistic: an arm gets a high probability if its estimate is high **or** if it is very uncertain.
- Computing these probabilities exactly is hard, but they are easy to **sample**.

EXAMPLE

Arm A wins 60 of 100 pulls; arm B wins 3 of 4. A looks safer, but B could easily be better. Probability matching gives B a real chance instead of ignoring it.



IN PLAIN WORDS

An arm gets pulled about as often as the agent believes it is the best one.

Thompson sampling

THOMPSON SAMPLING (THOMPSON, 1933)

1. For every arm, draw one sample $Q_t(a) \sim p_t(q(a))$ from its belief.
2. Pull the arm with the highest sample: $A_t = \operatorname{argmax}_a Q_t(a)$.
3. Observe the reward and update that arm's belief.

- This is sample-based probability matching: $\pi_t(a) = \mathbb{E}[\mathbb{I}(Q_t(a) = \max_{a'} Q_t(a'))]$.
- For win-or-lose bandits it reaches the Lai & Robbins lower bound: it is **optimal**.
- No bonus to tune, no ε to schedule: the uncertainty does the exploring.

TRY IT

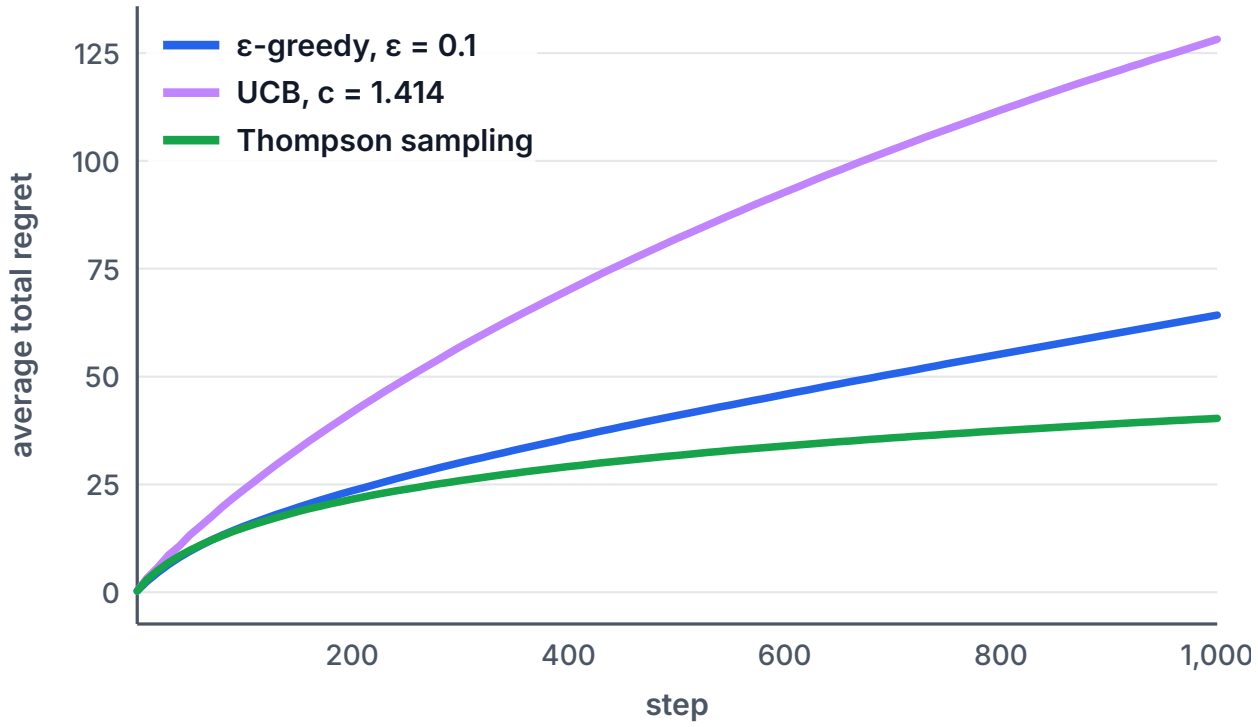
Beliefs: A = Beta(61, 41), B = Beta(4, 2), C = Beta(2, 5). One round draws, say, A: 0.62, B: 0.71, C: 0.35. Pull B. If it loses, B becomes Beta(4, 3) and is less likely to win the next draw.



IN PLAIN WORDS

Unsure arms draw wild samples, so they get tried. Arms known to be bad almost never win a draw.

Thompson sampling: our run



Our run: 10 win-or-lose arms, win rates drawn between 0.1 and 0.9, 2000 runs. Total regret after 1000 steps: Thompson 40, ε-greedy 64, UCB 128.

- Thompson sampling bends lowest here.
- UCB with $c = \sqrt{2}$ is cautious on this problem: its bonus is large compared with win rates between 0 and 1, so it explores for longer.
- Rankings depend on the problem and the settings; the shape is the lesson.

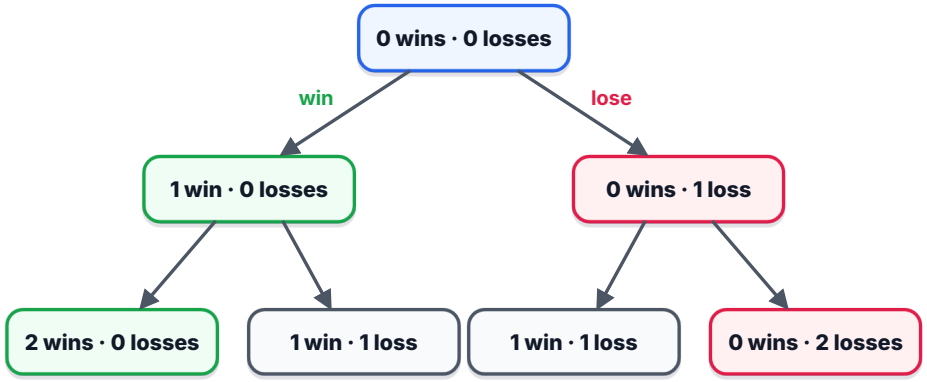


IN PLAIN WORDS

Explore where you are unsure, exploit where you are sure.

The information state

- So far we treated bandits as one-step decisions. We can also see them as **sequential** problems.
- Each step the agent updates a state S_t that summarises what it knows.
- Each action moves it to a new **information state** S_{t+1}, with probability $p(S_{t+1} \mid A_t, S_t)$.
- Now we have a Markov decision problem. The state lives inside the agent, and transitions are random because rewards are.



One win-or-lose arm: the information state is its counts of wins and losses. Every pull moves the agent down the tree.



IN PLAIN WORDS

Even in a bandit, actions affect the future after all, through learning.

Bayes-adaptive RL

- For a win-or-lose bandit with win rates μ_a, the information state is the counts of wins and losses for every arm.
- The bandit becomes an (infinite) MDP over information states, which we can try to solve with RL: plan into the future using the belief.
- This is called **Bayes-adaptive RL**. It trades off exploration optimally with respect to the prior.
- It extends to full RL by also learning a transition model, but it can be unwieldy and is hard to scale.



IN PLAIN WORDS

Every pull is worth what it pays plus what it teaches. Planning to explore puts a price on the teaching.

WHY IT IS HARD

With 10 arms and 100 pulls each, the number of possible count combinations is astronomically large, which is why this is rarely solved exactly.

WHY IT MATTERS

Even when not used directly, it tells us what exploration is **worth**, and inspires practical methods.

WRAP-UP

Summary

ALGORITHM	EXPLORES BY	NEEDS	TOTAL REGRET
Greedy	never	value estimates	linear
ε-greedy (constant ε)	random pulls	value estimates and ε	linear, gentler
Gradient bandit	its own softmax probabilities	preferences and a step size	often good in practice
UCB	a bonus for uncertainty	counts and c	logarithmic (proved)
Thompson sampling	random draws from beliefs	a belief per arm	logarithmic; optimal for win-or-lose arms
Bayes-adaptive	planning over information states	a prior and a lot of computation	optimal for the prior



IN PLAIN WORDS

Bandits are the simplest RL problem: one state, many actions. Exploration is needed because we learn only about what we try.



IN PLAIN WORDS

Lower bound: logarithmic regret (Lai & Robbins). UCB and Thompson sampling reach it; constant-ε methods do not.

Answers to check yourself

- **1.** Because the agent learns only about the arms it pulls; without trying arms it cannot know which is best.
- **2.** $v_* = 4$; regrets 3, 1 and 0.
- **3.** $10 \times 1 = 10$.
- **4.** It needs the true values $q(a)$, which the agent does not know.
- **5.** 4, then $4 + \frac{1}{2}(2 - 4) = 3$, then $3 + \frac{1}{3}(6 - 3) = 4$.
- **6.** When the arm's value changes over time (non-stationary): a constant step size keeps tracking it.
- **7.** $0.8 + 0.2/5 = 0.84$.
- **8.** It can lock onto a worse arm forever and then pays the same regret on every step.

CREDITS

Sources and credits

- The order of topics follows *Lecture 2: Exploration and Exploitation* of the DeepMind × UCL Reinforcement Learning Lecture Series (2021), by Hado van Hasselt. This deck is not an official DeepMind or UCL material.
- The text, figures, examples and simulations were written, drawn and run for ACML. Numbers labelled “our run” come from our own code; examples taken from books or papers are credited where they appear.
- Sutton & Barto (2018), *Reinforcement Learning: An Introduction*, chapter 2 (the 10-armed testbed).
- Lai & Robbins (1985); Auer, Cesa-Bianchi & Fischer (2002); Thompson (1933); Williams (1992); Lattimore & Szepesvári (2020).
- Companion videos: **ACML Explains**, season 1, episodes 6–9. More from the lab: acml.teddylazebnik.com



Applied Computational Mathematics Lab

END OF LECTURE 2

Next: many states and Markov decision processes

Lecture 3 brings back states and consequences: Markov decision processes, Bellman equations and dynamic programming.

