Lecture 2: Exploration and Exploitation

Hado van Hasselt

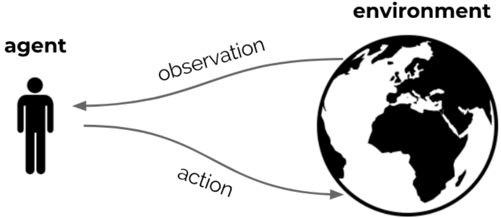
Reinforcement learning, 2021

Background

Recommended reading:
Sutton & Barto 2018, Chapter 2

Further background material:
Bandit Algorithms, Lattimore & Szepesvári, 2020
Finite-time analysis of the multiarmed bandit problem, Auer, Cesa-Bianchi, Fischer, 2002

Recap



- Reinforcement learning is the science of learning to make decisions
- Agents can learn a **policy**, **value function** and/or a **model**
- The general problem involves taking into account **time** and **consequences**
- Decisions affect the **reward**, the **agent state**, and **environment state**
- Learning is **active**: decisions impact data

This Lecture

In this lecture, we simplify the setting

- The environment is assumed to have only a **single state**
- $\implies$ actions no longer have long-term consequences in the environment
- $\implies$ actions still do impact **immediate reward**
- $\implies$ other observations can be ignored
- We discuss how to learn a policy in this setting

Blackboard: Example

Exploration vs. Exploitation

- Learning agents need to trade off two things
  - **Exploitation**: Maximise performance based on current knowledge
  - **Exploration**: Increase knowledge
- We need to gather information to make the best overall decisions
- The best long-term strategy may involve short-term sacrifices

Formalising the problem

The Multi-Armed Bandit

- A multi-armed bandit is a set of distributions $\{\mathcal{R}_a | a \in \mathcal{A}\}$
- $\mathcal{A}$ is a (known) set of actions (or “arms")
- $\mathcal{R}_a$ is a distribution on rewards, given action a
- At each step t the agent selects an action $A_t \in \mathcal{A}$
- The environment generates a reward $R_t \sim \mathcal{R}_{A_t}$
- The goal is to maximise cumulative reward $\sum_{i=1}^{t} R_i$
- We do this by learning a **policy**: a distribution on $\mathcal{A}$

Values and Regret

- The **action value** for action a is the expected reward

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

- The **optimal value** is

$$v_* \quad = \quad \max_{a \in \mathcal{A}} q(a) \quad = \quad \max_a \mathbb{E}\left[R_t \mid A_t = a\right]$$

- **Regret** of an action a is

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

- The regret for the optimal action is zero

Regret

- We want to minimise **total regret**:

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

- Maximise cumulative reward ≡ minimise total regret
- The summation spans over the full 'lifetime of learning'

Algorithms

Algorithms

- We will discuss several algorithms:
  - Greedy
  - ϵ-greedy
  - UCB
  - Thompson sampling
  - Policy gradients
- The first three all use **action value estimates** $Q_t(a) \approx q(a)$

Action values

- The **action value** for action a is the expected reward

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

- A simple estimate is the average of the sampled rewards:

$$Q_t(a) = \frac{\sum_{n=1}^{t} \mathcal{I}(A_n = a)\ R_n}{\sum_{n=1}^{t} \mathcal{I}(A_n = a)}$$

  $\mathcal{I}(\cdot)$ is the **indicator** function: $\mathcal{I}(\text{True}) = 1$ and $\mathcal{I}(\text{False}) = 0$

- The **count** for action a is

$$N_t(a) = \sum_{n=1}^{t} \mathcal{I}(A_n = a)$$

Action values

- This can also be updated incrementally:

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

$$\forall a \neq A_t : Q_t(a) = Q_{t-1}(a)$$

with

$$\alpha_t = \frac{1}{N_t(A_t)} \qquad \text{and} \qquad N_t(A_t) = N_{t-1}(A_t) + 1,$$

where $N_0(a) = 0$.

- We will later consider other **step sizes** α
- For instance, constant α would lead to **tracking**, rather than averaging

Algorithms: greedy

The greedy policy

- One of the simplest policies is **greedy**:
  - Select action with highest value: $A_t = \operatorname*{argmax}_a Q_t(a)$
  - Equivalently: $\pi_t(a) = \mathcal{I}(A_t = \operatorname*{argmax}_a Q_t(a))$ (assuming no ties are possible)

Example:
Regret of the greedy policy

Algorithms: ϵ-greedy

ϵ-Greedy Algorithm

- Greedy can get stuck on a suboptimal action forever
  $\implies$ linear expected total regret
- The **ϵ-greedy** algorithm:
  - With probability $1 - \epsilon$ select greedy action: $a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a)$
  - With probability ϵ select a random action
  - Equivalently:

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

- ϵ-greedy continues to explore
  $\Rightarrow$ ϵ-greedy with constant ϵ has linear expected total regret

Algorithms: Policy gradients

Policy search

- Can we learn policies $\pi(a)$ directly, instead of learning values?
- For instance, define **action preferences** $H_t(a)$ and a policy

$$\pi(a) = \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}} \qquad \text{(\textbf{softmax})}$$

- The preferences are not values: they are just learnable policy parameters
- Goal: learn by optimising the preferences

Policy gradients

- Idea: update policy parameters such that expected value increases
- We can use **gradient ascent**
- In the bandit case, we want to update:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \mathbb{E}[R_t | \pi_{\theta_t}] ,$$

  where θ_t are the current policy parameters
- Can we compute this gradient?

Gradient bandits

- Log-likelihood trick (also known as REINFORCE trick, Williams 1992):

$$
\begin{aligned}
\nabla_\theta \, \mathbb{E}[R_t|\pi_\theta] &= \nabla_\theta \sum_a \pi_\theta(a) \, \overbrace{\mathbb{E}[R_t|A_t = a]}^{=\, q(a)} \\
&= \sum_a q(a) \, \nabla_\theta \pi_\theta(a) \\
&= \sum_a q(a) \, \frac{\pi_\theta(a)}{\pi_\theta(a)} \, \nabla_\theta \pi_\theta(a) \\
&= \sum_a \pi_\theta(a) \, q(a) \, \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} \\
&= \mathbb{E}\left[R_t \, \frac{\nabla_\theta \pi_\theta(A_t)}{\pi_\theta(A_t)} \right] \qquad = \mathbb{E}\left[R_t \nabla_\theta \log \pi_\theta(A_t) \right]
\end{aligned}
$$

Gradient bandits

- Log-likelihood trick (also known as REINFORCE trick, Williams 1992):

$$\nabla_\theta \mathbb{E}[R_t|\theta] = \mathbb{E}\left[R_t \nabla_\theta \log \pi_\theta(A_t)\right]$$

- We can sample this!
- So

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

  this is **stochastic gradient ascent** on the (true) value of the policy
- Can use **sampled** rewards — does not need value estimates

Gradient bandits

- For soft max:

$$
\begin{aligned}
H_{t+1}(a) &= H_t(a) + \alpha R_t \frac{\partial \log \pi_t(A_t)}{\partial H_t(a)} \\
&= H_t(a) + \alpha R_t (\mathcal{I}(a = A_t) - \pi_t(a))
\end{aligned}
$$

- $\Rightarrow$

$$
\begin{aligned}
H_{t+1}(A_t) &= H_t(A_t) + \alpha R_t (1 - \pi_t(A_t)) \\
H_{t+1}(a) &= H_t(a) - \alpha R_t \pi_t(a) && \text{if } a \neq A_t
\end{aligned}
$$

- Preferences for actions with higher rewards increase more (or decrease less), making them more likely to be selected again

Theory: what is possible?

How well can we do?

Theorem (Lai and Robbins)

Asymptotic total regret is at least logarithmic in number of steps

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

(Note: $\mathrm{KL}(\mathcal{R}_a||\mathcal{R}_{a*}) \propto \Delta_a^2$)

- Note that **regret grows at least logarithmically**
- That's still a whole lot better than linear growth! Can we get it in practice?
- Are there algorithms for which the **upper bound** is logarithmic as well?

Counting Regret

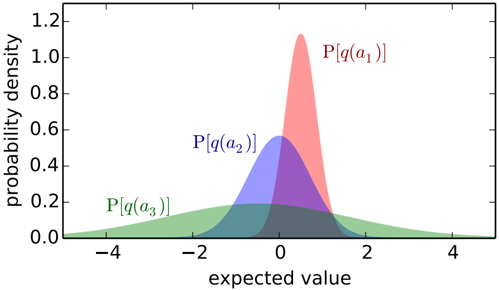
- Recall $\Delta_a = v_* - q(a)$
- Total regret depends on action regrets Δ_a and action counts

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

- A good algorithm ensures small counts for large action regrets

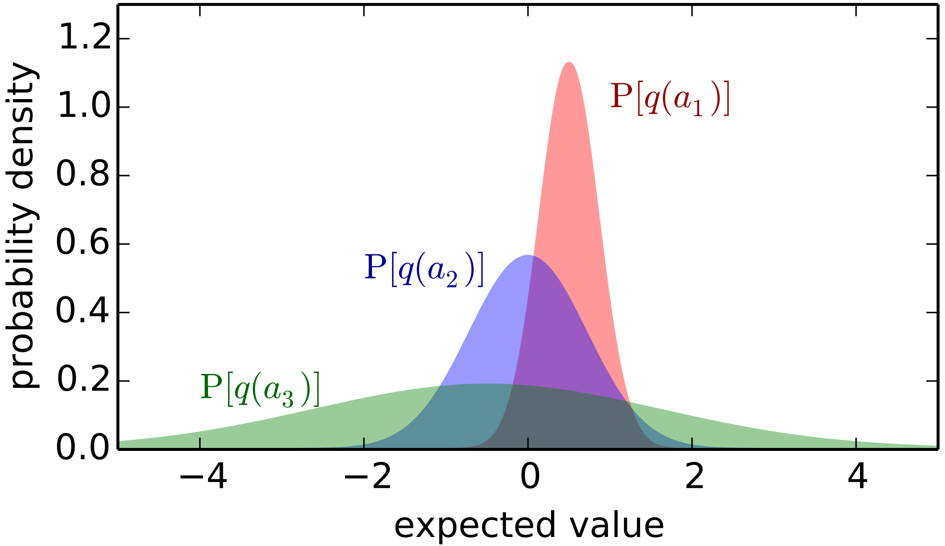
Optimism in the face of uncertainty

Optimism in the Face of Uncertainty

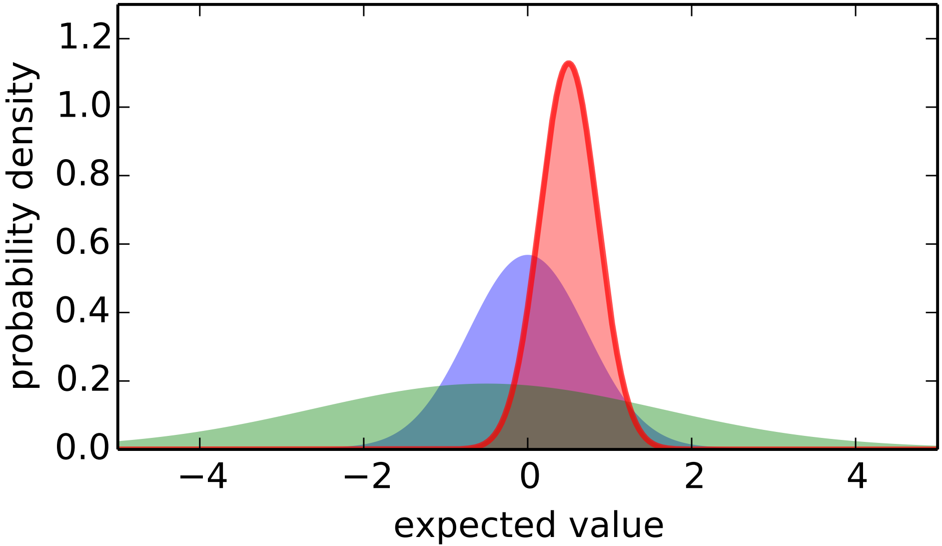


- Which action should we pick?
- More uncertainty about its value: more important to explore that action

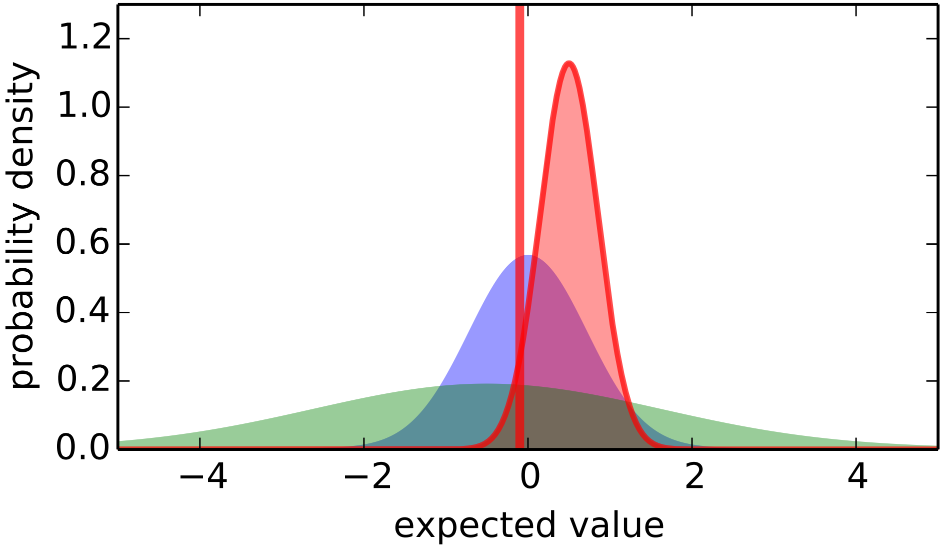
Optimism in the Face of Uncertainty



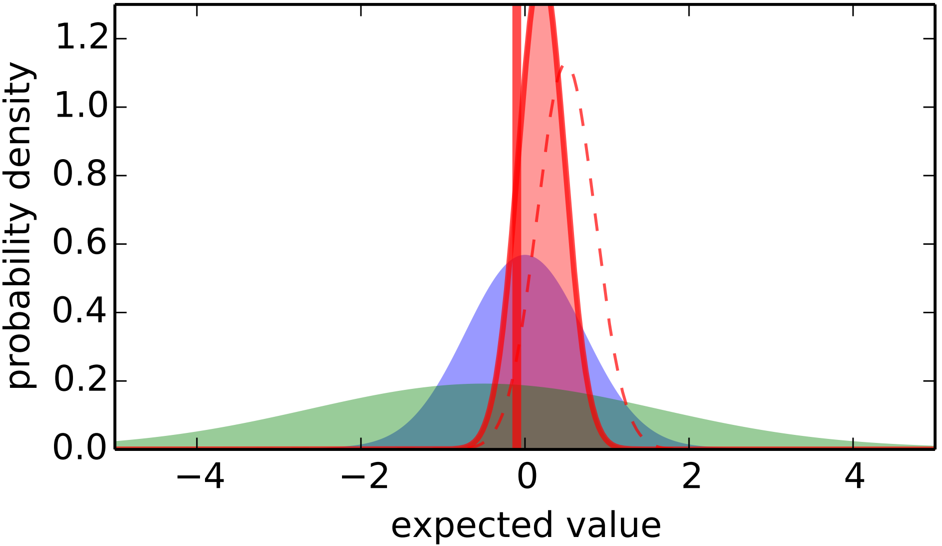
Optimism in the Face of Uncertainty



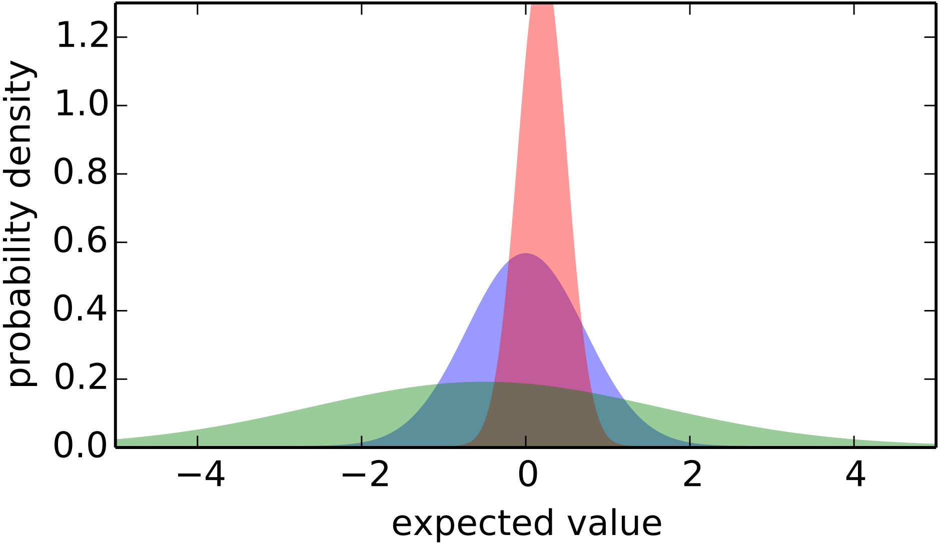
Optimism in the Face of Uncertainty



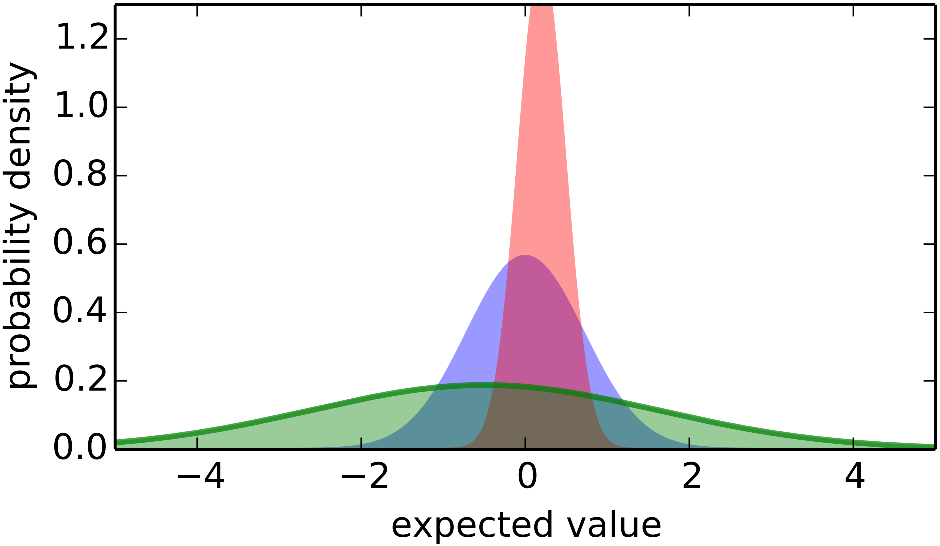
Optimism in the Face of Uncertainty



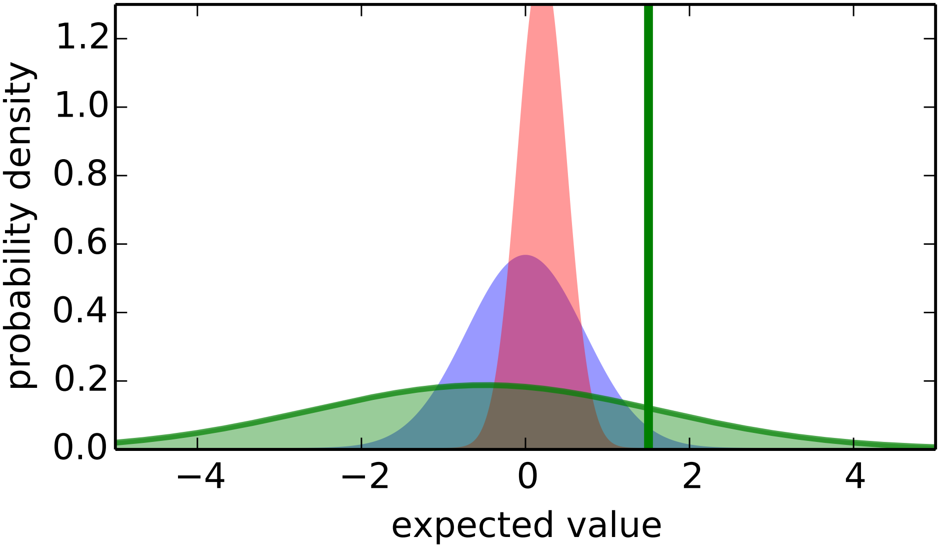
Optimism in the Face of Uncertainty



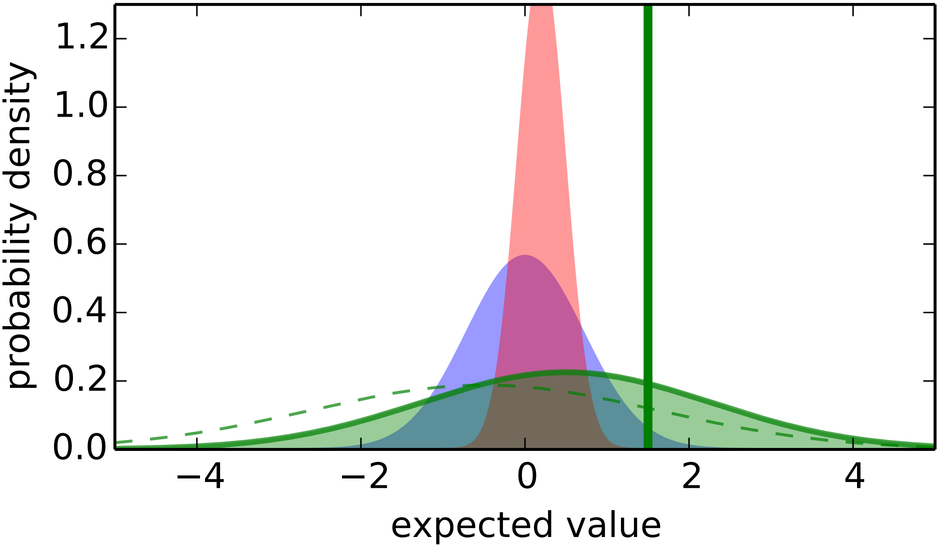
Optimism in the Face of Uncertainty



Optimism in the Face of Uncertainty



Optimism in the Face of Uncertainty



Algorithms: UCB

Upper Confidence Bounds

- Estimate an upper confidence $U_t(a)$ for each action value, such that $q(a) \leq Q_t(a) + U_t(a)$ with high probability
- Select action maximizing **upper confidence bound** (UCB)

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

- The uncertainty should depend on the number of times $N_t(a)$ action a has been selected
  - Small $N_t(a) \Rightarrow$ large $U_t(a)$ (estimated value is uncertain)
  - Large $N_t(a) \Rightarrow$ small $U_t(a)$ (estimated value is accurate)
- Then a is only selected if either...
  - ...$Q_t(a)$ is large (=good action), or
  - ...$U_t(a)$ is large (=high uncertainty) (or both)
- Can we **derive** an optimal bound?

Theory: the optimality of UCB

Hoeffding's Inequality

Theorem (Hoeffding's Inequality)

Let $X_1, \ldots, X_n$ be i.i.d. random variables in [0,1] with true mean $\mu = \mathbb{E}[X]$, and let $\overline{X}_t = \frac{1}{n}\sum_{i=1}^n X_i$ be the sample mean. Then

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

- We can apply Hoeffding's Inequality to bandits with bounded rewards
- If $R_t \in [0, 1]$, then

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

- By symmetry, we can also flip it around

$$p\left(Q_t(a) - U_t(a) \ge q(a)\right) \le e^{-2N_t(a)U_t(a)^2}$$

Calculating Upper Confidence Bounds

- We can pick a maximal desired probability p that the true value exceeds an upper bound and solve for this bound $U_t(a)$

$$
\begin{aligned}
&& e^{-2N_t(a)U_t(a)^2} &= p \\
\implies && U_t(a) &= \sqrt{\frac{-\log p}{2N_t(a)}}
\end{aligned}
$$

  We then know the probability that this happens is smaller than p

- Idea: reduce p as we observe more rewards, e.g., $p = 1/t$

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

- This ensures that we always keep exploring, but not too much

UCB

- UCB:

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

- Intuition:
  - If Δ_a is large, then $N_t(a)$ is small, because $Q_t(a)$ is likely to be small
  - So either Δ_a is small or $N_t(a)$ is small
  - In fact, we can prove $\Delta_a N_t(a) \leq O(\log t)$, for all a

UCB

- UCB:

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

where c is a hyper-parameter

Theorem (Auer et al., 2002)

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

$$L_t \leq 8 \sum_{a|\Delta_a > 0} \frac{\log t}{\Delta_a} + O(\sum_a \Delta_a), \qquad \forall t\,.$$

Blackboard:
UCB derivation

Bayesian approaches

Bayesian Bandits

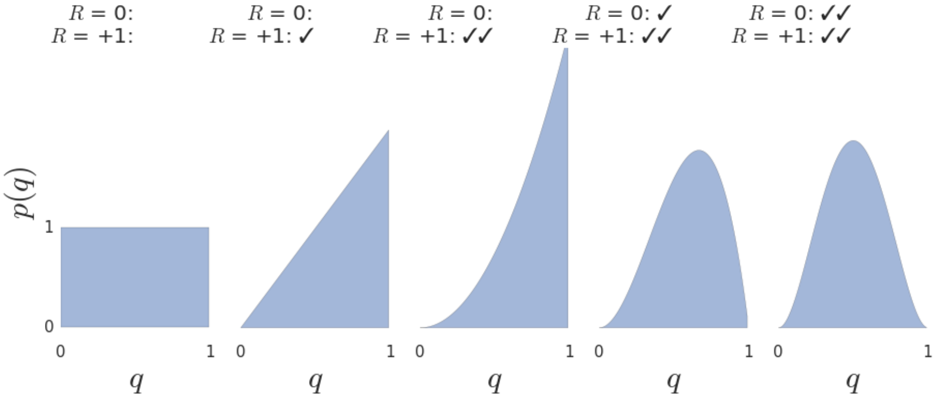
- We could adopt **Bayesian** approach and model distributions over values $p(q(a) \mid \theta_t)$
- This is interpreted as our **belief** that, e.g., $q(a) = x$ for all $x \in \mathbb{R}$
- E.g., θ_t could contain the means and variances of Gaussian belief distributions
- Allows us to inject rich prior knowledge θ_0
- We can then use posterior belief to guide exploration

Bayesian Bandits: Example

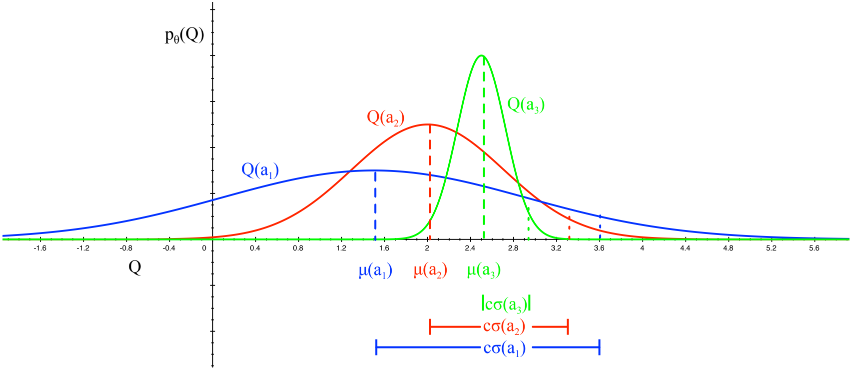
- Consider bandits with **Bernoulli** reward distribution: rewards are 0 or +1
- For each action, the prior could be a **uniform distribution** on $[0, 1]$
- This means we think each value in $[0, 1]$ is equally likely
- The posterior is a Beta distribution $\text{Beta}(x_a, y_a)$ with initial parameters $x_a = 1$ and $y_a = 1$ for each action a
- Updating the posterior:
  - $x_{A_t} \leftarrow x_{A_t} + 1$ when $R_t = 0$
  - $y_{A_t} \leftarrow y_{A_t} + 1$ when $R_t = 1$

Bayesian Bandits: Example

Suppose: $R_1 = +1$, $R_2 = +1$, $R_3 = 0$, $R_4 = 0$



Bayesian Bandits with Upper Confidence Bounds



- We can estimate upper confidences from the posterior
  - e.g., $U_t(a) = c\sigma_t(a)$ where $\sigma(a)$ is std dev of $p_t(q(a))$
- Then, pick an action that maximises $Q_t(a) + c\sigma(a)$

Algorithms: Thompson sampling

Probability Matching

- A different option is to use **probability matching**:
  Select action a according to the probability (belief) that a is optimal

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

- Probability matching is optimistic in the face of uncertainty:
  Actions have higher probability when either the estimated value is high, or the uncertainty is high
- Can be difficult to compute $\pi(a)$ analytically from posterior (but can be done numerically)

Thompson Sampling

- Thompson sampling (Thompson 1933):
  - Sample $Q_t(a) \sim p_t(q(a)), \forall a$
  - Select action maximising sample, $A_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q_t(a)$
- **Thompson sampling** is sample-based probability matching

$$\pi_t(a) = \mathbb{E}\left[\mathcal{I}(Q_t(a) = \max_{a'} Q_t(a'))\right]$$
$$= p\left(q(a) = \max_{a'} q(a')\right)$$

- For Bernoulli bandits, Thompson sampling achieves Lai and Robbins lower bound on regret, and therefore is **optimal**

Planning to explore

Information State Space

- We have viewed bandits as **one-step** decision-making problems
- Can also view as **sequential** decision-making problems
- Each step the agent updates state S_t to summarise the past
- Each action A_t causes a transition to a new **information state** S_{t+1} (by adding information), with probability $p(S_{t+1} \mid A_t, S_t)$
- We now have a Markov decision problem
- The state is fully internal to the agent
- State transitions are random due to rewards & actions
- Even in bandits actions affect the future after all, via learning

Example: Bernoulli Bandits

- Consider a Bernoulli bandit, such that

$$p(R_t = 1 \mid A_t = a) = \mu_a$$
$$p(R_t = 0 \mid A_t = a) = 1 - \mu_a$$

- E.g., win or lose a game with probability μ_a
- Want to find which arm has the highest μ_a
- The information state is $I = (\boldsymbol{\alpha}, \boldsymbol{\beta})$
  - α_a counts the pulls of arm a where reward was 0
  - β_a counts the pulls of arm a where reward was 1

Solving Information State Space Bandits

- We formulated the bandit as an infinite MDP over information states
- This can be solved by reinforcement learning
- E.g., learn a Bayesian reward distribution, plan into the future
- This is known as **Bayes-adaptive** RL:
  optimally trades off exploration with respect to the prior distribution
- Can be extended to full RL, by also learning a transition model
- Can be unwieldy... unclear how to scale effectively

Example

End of lecture