COMP 424 - Artificial Intelligence
Lecture 21: Bandits

Instructor: Jackie CK Cheung (jcheung@cs.mcgill.ca)

Recall: Lotteries

- A lottery is a set of outcomes, each associated with some probability

- E.g., suppose you had to choose between these two lotteries:
  - L_1: win \$1M for sure.
  - L_2: win \$5M with prob. 0.1
    win \$1M with prob. 0.89
    win \$0 with prob 0.01.

Recall: Utilities

- **Utilities map outcomes (or states) to real values.**

- **MEU principle**: Choose the action that maximizes expected utility.
  - Most widely accepted as a standard for rational behavior.

Value of Perfect Information (VPI)

- Suppose you have current evidence *E*, current best action *a**, with possible outcomes c_i. Then the expected value of *a** is:

$$EU(a^*|E) = \max_a U(a) = \max_a \sum_i U(c_i)P(c_i|E,a)$$

- Suppose you could gather further evidence about a variable *X*, should you do it?

Value of Perfect Information

- Suppose we knew $X=x$, then we would choose a_x^* such that:

$$EU(a_x^*|E, X = x) = \max_a \sum_i U(c_i)P(c_i|E, a, X = x)$$

- X is a random variable whose value is unknown, so we must compute expected gain over all possible values:

$$VPI_E(X) = \left(\sum_x P(X = x|E)EU(a_x^*|E, X = x)\right) - EU(a^*|E)$$

This is the value of knowing *X* exactly!

Properties of VPI

- **Non-negative**:

$$\forall X, E \quad \mathrm{VPI}_E(X) \geq 0$$

Note that VPI is an *expectation*. Depending on the actual value we find for X, there can actually be a loss post-hoc.

- **Non-additive**: E.g. consider obtaining X twice.

$$\mathrm{VPI}_E(X, X) \neq \mathrm{VPI}_E(X) + \mathrm{VPI}_E(X)$$

- **Order-independent**:

$$\mathrm{VPI}_E(X, Y) = \mathrm{VPI}_E(X) + \mathrm{VPI}_{E,X}(Y) = \mathrm{VPI}_E(Y) + \mathrm{VPI}_{E,Y}(X)$$

Acting under uncertainty: Other options

- Sometimes it can be advantageous to not always choose actions according to MEU, e.g. if the environment may change, or it is not fully known to the agent.
- **Random choice models**: choose the action with the highest expected utility most of the time, but keep non-zero probabilities for other actions as well.
  - Avoids being too predictable.
  - If utilities are not perfect, allows for exploration.
- **Minimizing regret**: consider the loss between the current behaviour and some "gold standard" behaviour, and try to minimize this loss.

Preference Elicitation

- Applications often require recommending something to a user or making a decision for them:
  - Online movie or book recommendation systems
  - Deciding which cancer treatment to give to a patient (has to take into account chance of survival, cost, side effects, etc.)
  - Deciding which ads to show on a dynamic web page
- For this, we need to know the utility that the user associates to different items.
- People are very bad at specifying utility values!
- **Preference elicitation** refers to finding out their preferences and translating them into utilities. Hard problem!

Outline for Today

Learning utilities from data

- Bandit problems
- Action values
- Exploration-exploitation tradeoff
- Simple exploration strategies

Bandit Problems

- A model of decision making under uncertainty; a **bandit** is the original name of a slot machine.
- **Environment is unknown**
  - We don't know what the lottery actually is!
  - Would like to learn the (expected) utility of a bandit
- ***k*-armed bandit**, a collection of *k* actions, each with a lottery associated with it

e.g., 3-armed bandit

Application #1: Internet advertising

- A large Internet company is interested in selling advertising on their website.
- It receives money when a company places an ad on the website and that ad gets clicked by a visitor to the website.
- What are the bandit's arms?

Application #1: Internet advertising

- A large Internet company is interested in selling advertising on their website.
- It receives money when a company places an ad on the website and that ad gets clicked by a visitor to the website.
- On a webpage, you can choose to display any of *n* possible ads.
  - Each ad has an unknown probability of click rate.
  - If the ad is clicked, there is a reward (utility), otherwise none.
- Q: What is the best advertisement strategy to maximize return?

Application #2: Network server selection

- Suppose you can choose to send a job from a user to be processed on one of several servers.
- The servers have different processing speed (e.g. due to geographic location, load, etc.)
- What are the bandit's arms?

Application #2: Network server selection

- Suppose you can choose to send a job from a user to be processed on one of several servers.
- The servers have different processing speed (e.g. due to geographic location, load, etc.)
- Each server can be viewed as an action (arm).
- Over time, you want to learn what is the best action to select.
- This is used in routing, DNS server selection, cloud computing, etc.

Playing a bandit

- You can choose repeatedly among the *k* actions; each choice is called a **play**.
- After each play a_t, the machine gives a reward r_t drawn from the distribution associated with a_t.
- The value of action *a* is its expected utility: $Q^*(a) = E[r \mid a]$
  - Note that *r* is drawn from a probability distribution that depends on *a*.
- **Objective:** Choose arms in a way that maximizes the reward obtained in the long run (e.g. over 1000 rounds).

Estimating action values

- Suppose that action a has been chosen n times, and the rewards received were $r_1, r_2, \ldots, r_n$.
- Then we can estimate the value of the action as the sample average of the rewards obtained:

$$Q_n(a) = (r_1 + r_2 + \ldots + r_n) / n$$

- As we take the action more, this estimate becomes more accurate (law of large numbers) and in the limit:

$$lim_{n->\infty} Q_n(a) = Q^*(a)$$

- One can even measure how fast Q_n approaches Q^*.

Estimating action values incrementally

- Do we need to remember all rewards so far to estimate $Q_n(a)$?
- No! Just keep a running average:

$$
\begin{aligned}
Q_{n+1}(a) &= (r_1 + r_2 + \ldots + r_{n+1}) / n+1 \\
&= (r_1 + r_2 + \ldots + r_n) / (n+1) && + \; r_{n+1} / (n+1) \\
&= nQ_n(a) / (n+1) && + \; r_{n+1} / (n+1) \\
&= (nQ_n(a)+Q_n(a)-Q_n(a)) / (n+1) && + \; r_{n+1} / (n+1) \\
&= Q_n(a) && + (r_{n+1} - Q_n(a)) / (n+1)
\end{aligned}
$$

  - The first term is the old value estimate.
  - The second term is the error between the new sample and the old value estimate, weighted by a decreasing step size (=1/(n+1)).

Exploration-exploitation trade-off

Tension between the two:

- **Explore** actions, to figure out which one is best (which means some amount of random choice.)
- **Exploit** the knowledge you already have, which means picking the **greedy** action: $a_t^* = argmax_a\ Q_t(a)$

- You cannot explore all the time (because you will lose big-time).
- You cannot exploit all the time, because you may end up with sub-optimal policy.

What if the problem is non-stationary?

- Sample averaging works if problem is **stationary** (i.e. $Q^*(a)$ does not change over time).
- In some applications (e.g. advertising), $Q^*(a)$ may change over time → most recent rewards should be emphasized!
- Instead of $1/n$, use a constant step size $\alpha \in (0,1)$ for value updates:

$$Q_{n+1}(a) = Q_n(a) + \alpha(r_{n+1} - Q_n(a))$$

- This leads to a recency-weighted average estimate:

$$Q_{n+1}(a) = (1-\alpha)^n Q_0(a) + \Sigma_{i=1:n+1} \alpha(1-\alpha)^{n-i} r_i$$

- Because $\alpha \in (0,1)$, most recent reward, r_n has highest weight

Exploration in non-stationary bandits

- If the environment is stationary, you may want to reduce exploration over time.
- **If the environment is not stationary, you can never stop exploring.**

- Why?

Strategies

1. Greedy
2. Epsilon-greedy
3. Softmax action selection
4. Optimistic initialization
5. Upper confidence bound

Strategy 1: Greedy

- Always pick the arm a with the best estimate of $Q_n(a)$

- Does this favour exploration or exploitation?

Exercise:

1. Come up with an example where this strategy results in not finding the optimal action.
2. Come up with the general condition where this strategy results in a suboptimal solution.

Strategy 2: ε-greedy action selection

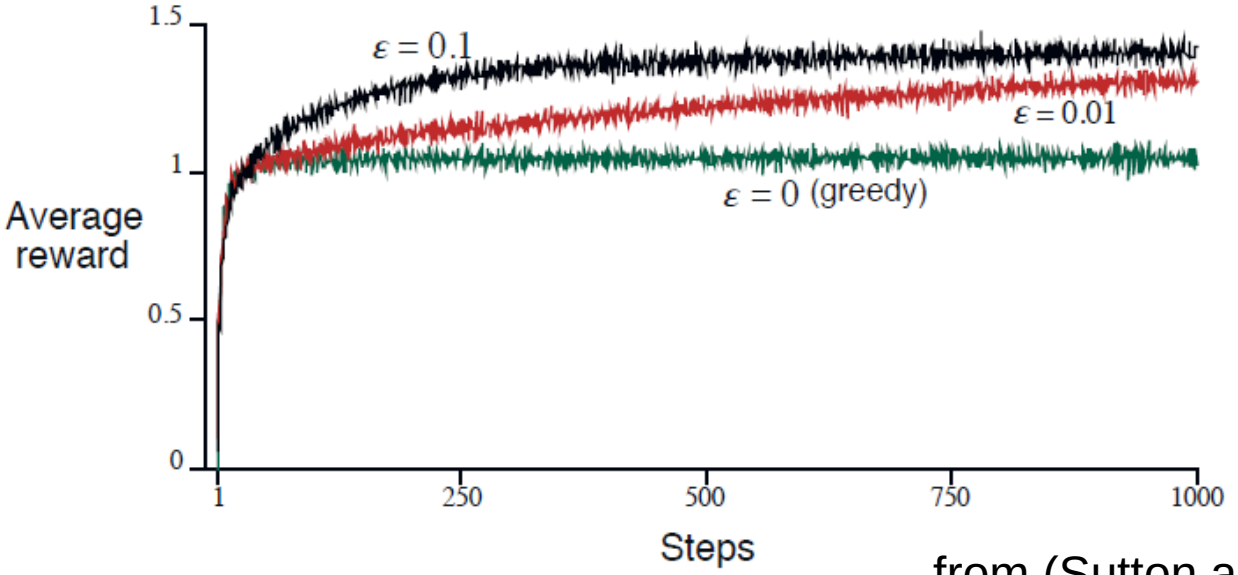
- Pick $\varepsilon \in (0,1)$ (a constant), usually small (e.g. $\varepsilon = 0.1$).
- On every play:
  - With probability ε you pull a random arm (explore).
  - With probability $1-\varepsilon$, pull the best arm according to current estimates (exploit).
- You can make ε depend on time (e.g. $1/t$, $1/\sqrt{t}$, …)

- Advantage: Very simple! Easy to understand, easy to implement.
- Disadvantage: Leads to discontinuities (Why?)

Example: 10-armed bandit problem

- The mean reward for each arm is chosen from a normal distribution with mean 0 and standard deviation 1
- Rewards are generated from a normal distribution around the true mean, with st.dev.1.
- We average 2000 different independent runs, each starts from $Q(a)=0$ and does 1000 pulls using the ε–greedy strategy.

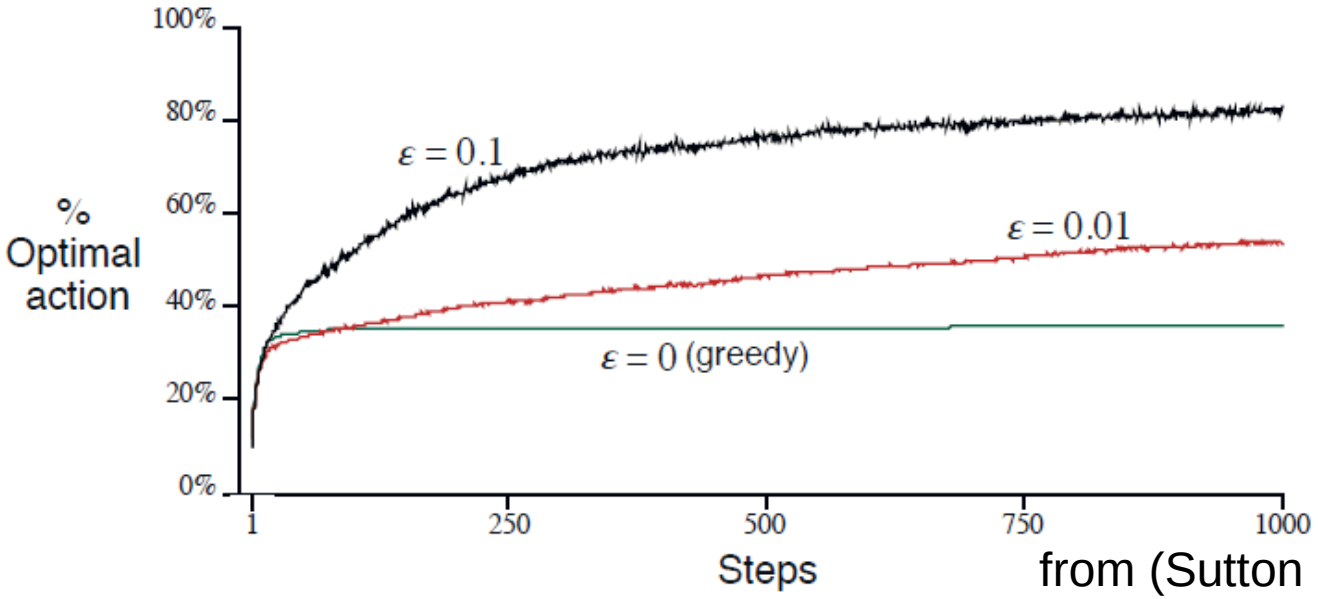
Example: 10-armed bandit



from (Sutton and Barto, 2016)

- If $\varepsilon=0$, convergence to a sub-optimal strategy.
- If ε is too low, convergence is very slow.
- If ε is too high (not pictured here) rewards received during learning may be too low, and have high variance.

Example: 10-armed bandit (cont'd)



from (Sutton and Barto, 2016)

- If $\varepsilon=0$, convergence to a sub-optimal strategy.
- If ε is too low, convergence is very slow.
- If ε is too high (not pictured here) rewards received during learning may be too low, and have high variance.

Strategy 3: Softmax action selection

- **Key idea**: make the action probabilities a function of the current action values, $Q_t(a)$.

- At time t, we choose action a with probability proportional to:

$$Pr(a_t) = e^{Qt(a)/\tau}$$

  - Normalize probabilities so they sum to 1 over the actions.
  - τ is the temperature parameter.
  - Similar to simulated annealing.

How does τ influence the exploration strategy?

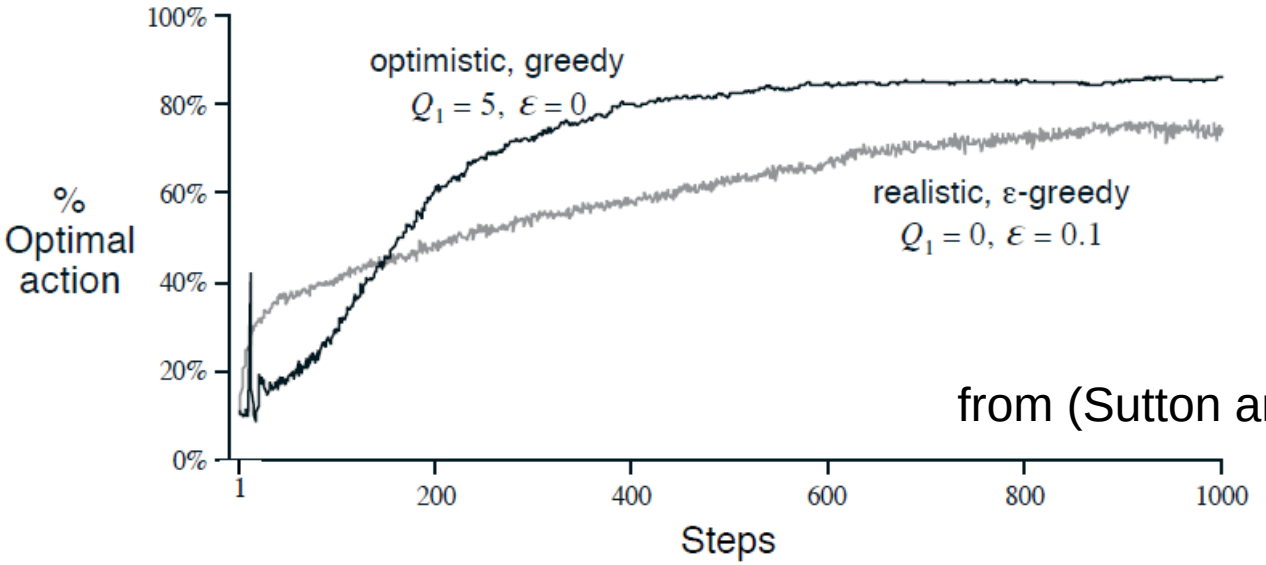
Strategy 4: Optimistic initialization

- If you do not know anything about an action, assume it's great! (***optimism in the face of uncertainty***)
  - Very powerful idea – recall A* and admissible heuristics.

- Implementation:
  - Initialize action values higher than they could possibly be, $Q_0(a)=\infty$, or a large positive constant.
  - Choose actions according to deterministic strategy: always pick the action with the best current estimate (break ties randomly).

- Whatever you do, you will be "disappointed", but…
  - If the action is not so disappointing, you will try it often.
  - If an action is very disappointing, you will learn quickly to avoid it.

Illustration: Optimistic initialization



from (Sutton and Barto, 2016)

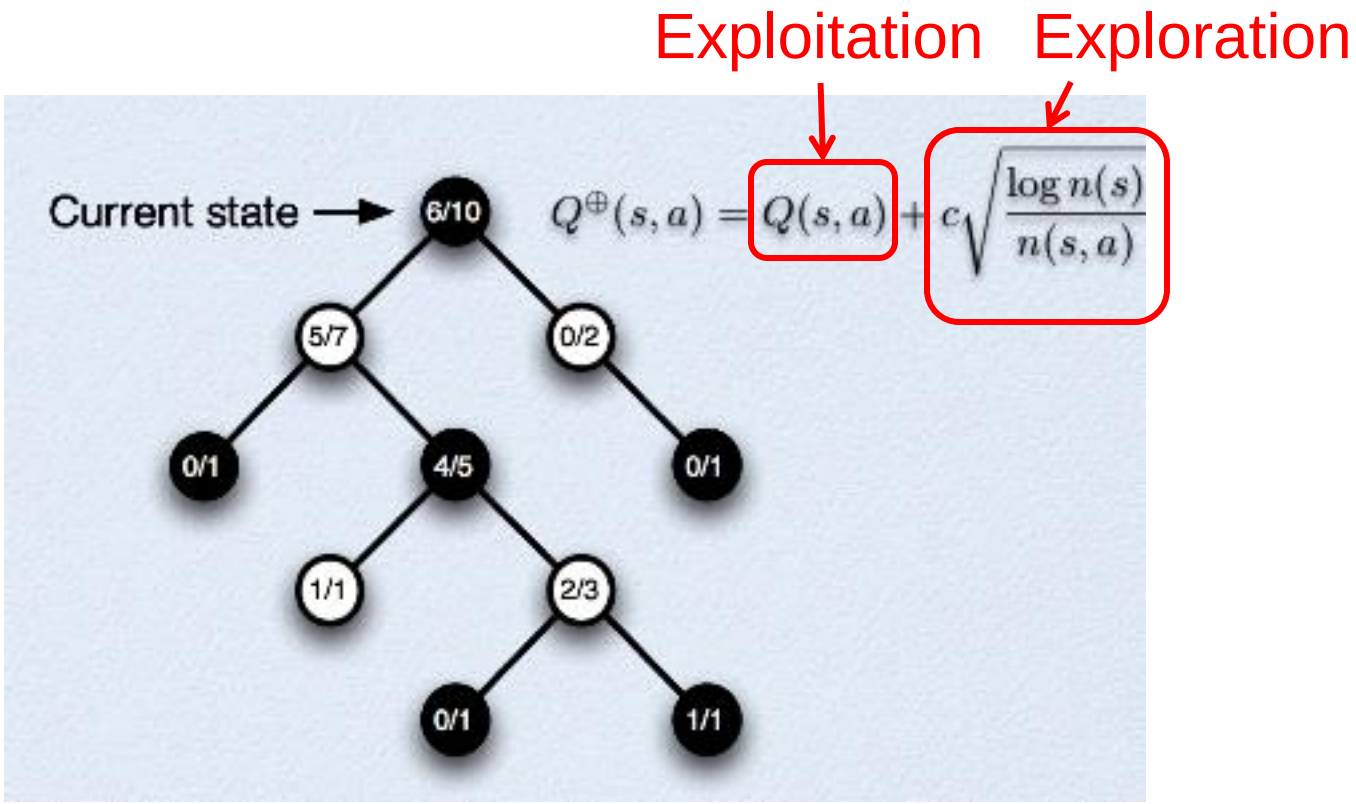
- Leads to more rapid exploration than ε–greedy strategy, which is bad in the short run but good in the long run.
- Once the optimal strategy is found, it stays there (since there is no randomness).

A better idea: Confidence intervals

- Suppose you have a random variable X with mean $E[X]=\mu$.
- The standard deviation of X measures how much X is spread around its mean:
  - This can be est $\sigma = \sqrt{E[(X-\mu)^2]}$.
- **Idea for exploration**: Add 1 standard deviation to estimate $Q_t(a)$.

  Choose actions with respect to this **upper-confidence bound**.

Remember: UCT (Upper Confidence Trees)



Used for search and game trees!

Strategy 5: Upper-confidence bound (UCB)

- In UCB the confidence bounds are generated over time rather than generated through simulations.

- Very similar formula. Pick *a* greedily using: $Q(a) + \sqrt{\frac{2 \log n}{n(a)}}$

  where *n* is the total number of actions picked so far and *n(a)* is the number of times action *a* was picked.

- Several tweaks of the upper-bound term have been proposed, lots of theoretical analysis for this type of method has been done.

Which algorithm to use?

- **Good news**: All algorithms converge in the limit to the correct values (given appropriate parameter changes if need be), assuming the environment is stationary.

- UCB has provably the fastest convergence when considering regret: $\sum_{t=1}^{\infty} Q(a^*) - Q(a_t)$

  This sum is *O(log(t))* for UCB, and a matching lower bound exists.

- However, when considering a finite training period after which the greedy policy is used forever, simpler strategies often perform better in practice.

Contextual bandits

- The usual bandit problem has no notion of "state", we just observe some interactions and payoffs.
  - In general, more information is available, e.g. placing ads on a webpage, you can observe the current words displayed.
- Contextual bandits have some state information, summarized in a vector of measurements *s.*
- The state is chosen by the environment. The agent chooses the action. The action's value depends on the state:

$$Q(\mathbf{s}, a) = \mathbf{w}_a^T \mathbf{s}$$

- Exploration methods remain similar (greedy, Boltzmann, UCB).
- It is necessary to learn the parameters *w* in this problem.

Next Two Lectures

- Guest lectures on **deep reinforcement learning**
  - Given by Vincent François-Lavet (McGill/Mila)
  - Responsible for state-of-the-art game playing agents, like Atari, AlphaGo, Alpha
  - Material is testable on the final exam. Don't miss it!