Lecture 13: Bandit problems. Markov Processes

- Bandit problems
- Action values (and now to compute them)
- Exploration-exploitation trade-off
- Simple exploration strategies
  - ϵ-greedy
  - Softmax (Boltzmann) exploration
  - Optimism in the face of uncertainty
- Markov decision processes

Recall: Lotteries and utilities

- Last time we defined a lottery as a set of *lottery* as a set of outcomes and a probability distribution over them
- If an agent has a "consistent" set of preferences over outcomes, each outcome can be associated with a *utility (reward, payoff)* (a real number)
- The utility of a lottery $L = (C, P)$ is the expected value of the utility of its outcomes:

$$U(L) = \sum_{c_i \in C} P(c_i) U(c_i)$$

- From now on, we will always talk about utilities instead of consequences
- The goal of a rational agent will be to *maximize its expected utility, over the long term*

Bandit problems

- Invented in early 1950s by Robbins to model decision making under uncertainty when the environment is unknown
- Named after the original name of slot machines

- A *k-armed bandit* is a collection of k *actions (arms)*, each having a lottery associated with it
- Unlike in the settings discussed before, the lotteries are *unknown ahead of time*
- The best action must be determined by interacting with the environment

Application: Internet advertising

- You are a large Internet company who sells advertising on the web site of their search engine
- You get paid by a company placing an ad with you if the ad gets clicked
- On any web page, you can choose one of n possible ads to display
- Each ad can be viewed as an action, with an unknown probability of success
- If the action succeeds, there is a reward, otherwise no reward
- What is the best strategy to advertise?
- Note that this setup requires *no knowledge* of the user, the ad content, the web page content... which is great if you need to make a decision very fast

Application: Network server selection

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

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:

$$Q^*(a) = E\{r|a\}$$

(Note that r is drawn from a probability distribution that depends on a)

- The objective is to play in a way that maximizes the reward obtained in the long run (e.g. over 1000 plays)

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:

$$Q_n(a) = \frac{1}{n}(r_1 + r_2 + \ldots r_n)$$

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

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

One can even express 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:

$$\begin{aligned} Q_{n+1}(a) &= \frac{1}{n+1}(r_1 + r_2 + \cdots + r_{n+1}) \\ &= \frac{1}{n+1} r_{n+1} + \frac{1}{n+1}\frac{n}{n}(r_1 + r_2 + \cdots + r_n) \\ &= \frac{1}{n+1} r_{n+1} + \frac{n}{n+1} Q_n(a) \\ &= Q_n(a) + \frac{1}{n+1}(r_{n+1} - Q_n(a)) \end{aligned}$$

- 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 *step size*
- We will see this pattern of update a lot in learning algorithms

What if the problem is non-stationary?

- Using the sample average works if the problem we want to learn is *stationary* ($Q^*(a)$ does not change)
- In some applications (e.g. advertising) $Q^*(a)$ may change over time, so we want the most recent rewards to be emphasized
- Instead of $1/n$ use a *constant step size* $\alpha \in (0, 1)$ in the value updates:

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

- This leads to a *recency-weighted average*:

$$Q_n(a) = (1-\alpha)^n Q_0(a) + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} r_i$$

Because $\alpha \in (0, 1)$, the most recent reward, r_n, has the highest weight, and the weights decrease exponentially for older rewards.

How to choose actions?

- Suppose you played a 2-armed bandit 3 times (2 for the left arm, 1 for the right arm). With the left arm, you won once and lost once. With the right arm you lost. What should you do next?
- Suppose you played a 2-armed bandit 30 times (20 for the left arm, 10 for the right arm). With the left arm, you won 10 times and lost 10 times. With the right arm you won 7 times and lost 3 times. What should you do next?
- Suppose you played a 2-armed bandit 3000 times (2000 for the left arm, 1000 for the right arm). With the left arm, you won 1000 times and lost 1000 times. With the right arm you won 700 times and lost 300 times. What should you do next?

Exploration-exploitation trade-off

- On one hand, you need to *explore* actions, to figure out which one is best (which means some amount of random choice)
- On the other hand, you want to *exploit* the knowledge you already have, which means picking the *greedy action*:

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

- You cannot explore all the time (because you may lose big-time)
- You cannot exploit all the time, because you may end up with a sub-optimal action
- If the environment is stationary, you may want to reduce exploration over time
- If the environment is not stationary, you can never stop exploring

Exploration-exploitation trade-off

- Simple randomized strategies:
  - ϵ-greedy
  - Softmax (Boltzman) exploration
- Deterministic strategy: optimism in the face of uncertainty
  - Optimistic initialization
  - Confidence intervals
  - UCB1
  - ...
- A lot of other work!
  - Gittins indices
  - Aciton elimination
  - ...

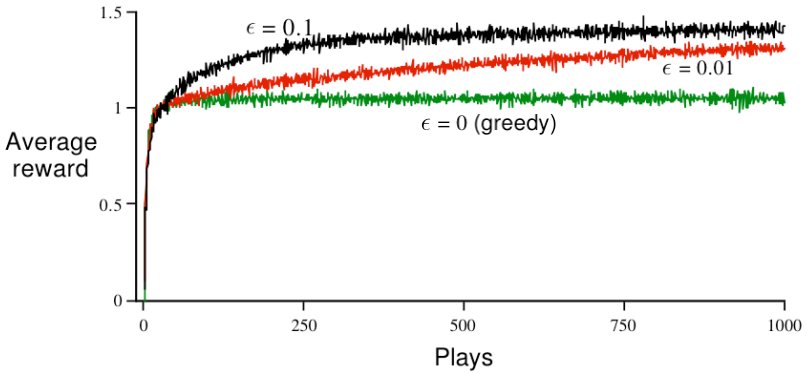
ϵ-greedy action selection

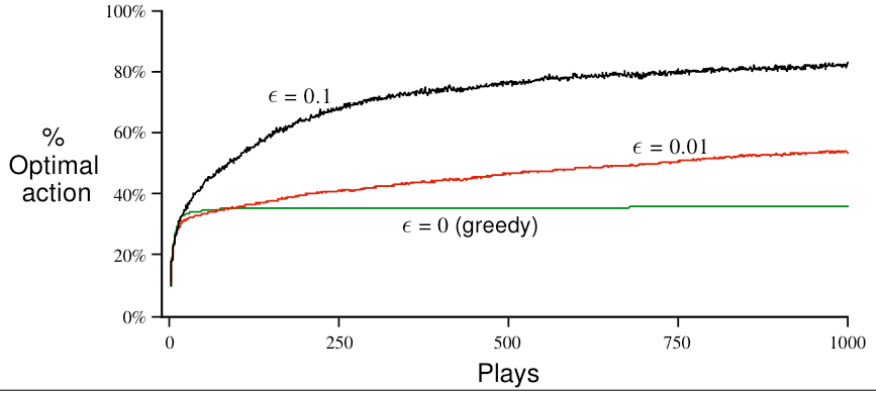
- Pick $\epsilon \in (0, 1)$ (a constant) - usually small (e.g. 0.1)
- On every play, with probability ϵ you pull a random arm
- With probability $1-\epsilon$, pull the best arm according to current estimates (greedy action)
- You can make ϵ depend on time (e..g $1/t$, $1/\sqrt{t}$, ...)
- Advantage: Very simple! Easy to understand, easy to implement
- Disadvantage: leads to *discontinuities* (why?)

Illustration: 10-armed bandit problem

- The mean reward for each arm is chosen form 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: start from scratch, do 1000 pulls
- What should the influence of ϵ be?

Illustration: 10-armed bandit problem





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

Softmax action selection

- Key idea: make the action probabilities a *function of the current action values*
- Like in simulated annealing, we use the Boltzman distribution:
- At time t we choose action a with probability proportional to:

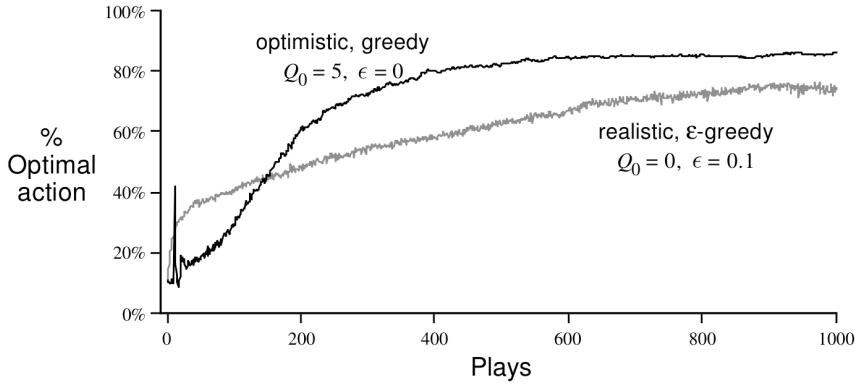
$$e^{Q_t(a)/\tau}$$

- Normalize probabilities so they sum to 1 over the actions
- τ is a temperature parameter
- How does τ influence the exploration strategy?

Optimism in the face of uncertainty

- If you do not know anything about an action, assume it's great!
- Very powerful idea - recall A^* and admissible heuristics
- Simple implementation: just initialize all action values higher than they could possibly be
- Choose actions according to a *deterministic strategy*: always pick the action with the best current estimate
- Whatever you do, you will be "disappointed", which leads to trying out all actions
- This is a *deterministic strategy*: always pick the action with the best current estimate

Illustration: optimistic initialization



- Leads to more rapid exploration that ϵ-greedy, 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)

More sophisticated 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:

  $$\sigma = \sqrt{E[(X - \mu)^2]}$$

  This can be estimated from samples
- Idea: add one standard deviation to the mean, choose action values wrt to this bound