Lecture 5 - Multi-armed Bandits

Instructor: GuangBing Yang, PhD

What is about Multi-armd Bandits

- Bandit is a game about choose actions and receive reward according to the action chosen
- In *k*-armed bandit problem, there are k possible actions to choose from, e.g, 10 armed bandit has 10 actions for an agent to take, each of them corresponds to a different reward that comes from a distribution corresponding to that action.
- The goal is to identify which are best actions.
- The problem can become more complicated when the reward distributions are non-stationary.

What is about Multi-armd Bandits

- For stationary case:
  - each of actions has an expected reward, $q_*(a) = \mathbb{E}[R_t | A_t = a]$, the true value.
  - expect estimation of action value $Q_t(a)$, approximates to the true value.
  - **exploration vs. exploitation** — randomly vs. greedy to maximize rewards
  - with exploration, some rewards are lower in the short-term run, but higher in the long-term run because the exploration discovers better actions.

- Whether exploration or exploitation is a very hard problem. It is a problem of uncertainties, optimization, decision making, game theory (e.g., Nash Equilibrium), and so on.
- Action-value methods:
  - $Q_t(a) = \dfrac{\text{sum of rewards when a taken prior to t}}{\text{number of times a taken prior to t}} = \dfrac{\sum_{i=1}^{t-1} R_i \times 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$
- where $1_{A_i=a} = 1$when the action a is taken or the prediction of the action a is true, or $1_{A_i=a} = 0$ when a is not taken or the prediction of the action a is false.
- If $\sum_{i=1}^{t-1} 1_{A_i=a}$ is 0, the $Q_t(a) = 0$ by definition.

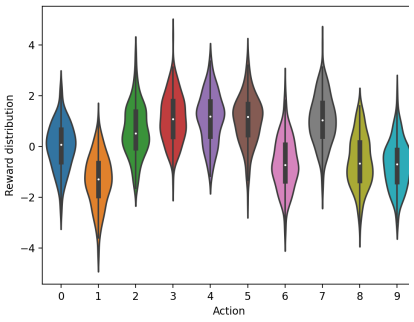
- The greedy action
  - The simplest selection rule, totally exploitation, $A_t = \text{argmax}_a Q_t(a)$, where argmax_a denotes the action a for which the expression $Q_t(a)$ outcomes takes the maximized value.
- The ε − greedy action
  - greedily selects the maximums most of the time, but every once a while, say with small probability ε,
  - instead select randomly from among all the actions with equal probability, independently of the action-value estimates.

yguangbing@gmail.com, Guang.B@chula.ac.th Feb 16 2021

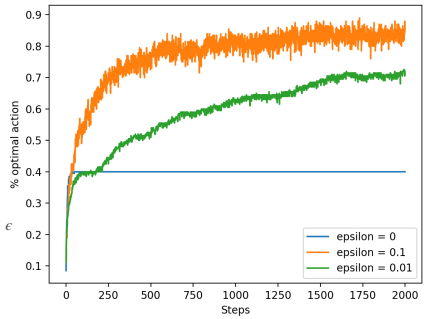
The 10-armd Testbed

- It is a test set for the performance of these algorithms:
  - composed of N=2000 randomly generated 10-bandit problems.
  - the action values $q_*(A_t)$ was randomly selected from a normal distribution $\mathbb{N}(0,1)$,
  - the reward distributions for each action are Gaussian with $\mathbb{N}(q_*(A_t),1)$.
  - the lower ε, the learning converges slower, but got higher action-values.
  - It is a straightforward problem if the distribution of the rewards is stationary.

yguangbing@gmail.com, Guang.B@chula.ac.th Feb 16 2021



An example bandit problem from the 10-armed testbed. The true value q.(a) of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean q.(a) unit variance normal distribution, as suggested by these coloured distributions.



Average performance of -greedy action-value methods on the 10-armed testbed. The epsilon= 0.1 method explored more, and usually found the optimal action earlier.

yguangbing@gmail.com, Guang.B@chula.ac.th 7 Feb 16 2021

Incremental Implementation

- It is an effective way to compute the rewards with constant memory and per-time-step computation.
- Notation:
  - R_i - the reward received after its selection of this action.
  - Q_n - the estimate of this action value after it has been selected n-1 times.
  - $Q_n = \dfrac{R_1 + R_2 + \ldots + R_{n-1}}{n-1}$,
  - an issue of this approach — calculate for every estimate and store it.
  - an incremental formulas needed to update averages with small, constant computation.

yguangbing@gmail.com, Guang.B@chula.ac.th Feb 16 2021

Incremental Implementation

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}(R_n + \sum_{i=1}^{n-1} R_i) \\
&= \frac{1}{n}(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i), \\
&= \frac{1}{n}(R_n + (n-1)Q_n) \\
&= \frac{1}{n}(R_n + nQ_n - Q_n) \\
&= Q_n + \frac{1}{n}[R_n - Q_n]
\end{aligned}
$$

- For n = 1, $Q_2 = R_1$ for arbitrary Q_1.
- In general, the update rule of the action value:

NewEstimate ← OldEstimate + StepSize[Target − OldEstimate]

[Target − OldEstimate] is the error in estimate.

Stepsize changes each step. This might be a problem because when n is large, the stepwise becomes small, and the recent action has less affect.

We call this approach as a simple bandit algorithm shown next,

Incremental Implementation

A simple bandit algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Loop forever:
$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

A simple (average) bandit algorithm screenshot from the book "Reinforcement Learning An Introduction Second Edition by Richard S and Andrew G Barto"

Tracking a Nonstationary Problem

- The simple (average) algorithms are useful for stationary bandit problems—the reward probabilities do not change over time.
- But often, the reward probabilities change—nonstationary.
- Need to give more weight to recent rewards than to past ones.
- A common way for this is using a constant step-size parameter.
- Then, $Q_{n+1} = Q_n + \alpha[R_n - Q_n] = \ldots = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$, weighted average
- because $(1-\alpha) + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$, (prove it? why does it equal to one?)
- because $1 - \alpha < 1$, the weight given to R_i decreases exponentially as the number of intervening rewards increases.
- If $1 - \alpha = 0$, then all the weight goes on the very last reward, R_n (note: $0^0 = 1$)

Tracking a Nonstationary Problem

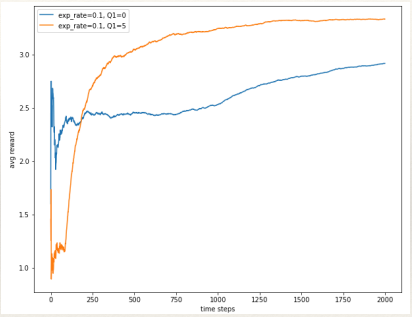
- It is convenient to vary the step-size parameter from step to step.
  - Let $\alpha_n(a)$ denote the step-size parameter.
  - the choice $\alpha_n(a) = \frac{1}{n}$ results in the sample-average method,
  - the sample-average method is guaranteed to converge to the true action values by the law of large numbers.
  - the conditions ensure the converge are: $\sum_{n=1}^{\infty} \alpha_n(a) = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$.
  - Both conditions are satisfied in the stationary case, like the simple-average, but not for non stationary case.
  - The second condition is not satisfied in non stationary cases.
  - Actually, this un-satisfied situation is desirable in a non stationary environment and is the most common in reinforcement learning because this allows the exploration.

Optimistic Initial Values

- Most methods discussed so far are dependent to the initial action-value estimates $Q_1(a)$.
- These methods are biased by their initial estimates.
  - for the sample-average, the bias disappears when all actions have been selected at least once,
  - for constant α, the bias is permanent.
  - In practice, this bias is usually not a problem, but can sometimes be very helpful.
  - It is an easy way to provide a prior knowledge of the rewards to be expected.
  - Initial action value is used as a simple way to encourage exploration.

Optimistic Initial Values

- This technique for encouraging exploration is called optimistic initial values.
- The result is that all actions are tried several times before the value estimates converge. The system does a fair amount of exploration even if greedy actions are selected all the time.



The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods use step-size=0.1

Upper Confidence-bound Action Selection

- Exploration is essential due to the uncertainty about the accuracy of the action-value estimates.
- The greedy actions look best at present.
- Others try non-greedy actions, for example, ϵ-greedy with no preference for those that are nearly greedy or particularly uncertain.
- A better way is to select among the non-greedy actions according to their potential for actually being optimal.
- This idea leads to the UCB and gradient bandit algorithms.

Upper Confidence-bound Action Selection

- The UCB selects actions according to: $A_t = \text{argmax}_a[Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}}]$,
  - where $\ln(t)$ denotes the natural logarithm of t,
  - $N_t(a)$ — the number of times that action a has been selected prior to time t,
  - c > 0 controls the degree of exploration.
  - If $N_t(a) = 0$, then a is considered to be a maximizing action.

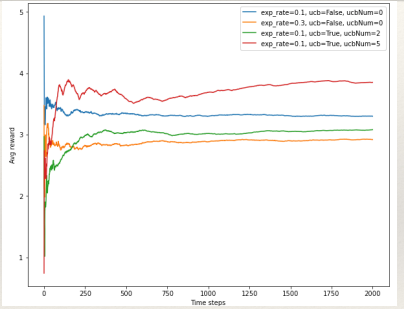
Upper Confidence-bound Action Selection

- In UCB algorithm, the square-root term is a measure of the uncertainty or variance in the estimate of a's value.
- Thus, the maximum determines its upper bound on the possible true value of action a—means it has a upper limit.
- Each time when a is selected, $N_t(a)$ increases, so the uncertainty decreases—which make sense because when more a selected, more certainty to a—as it appears in the denominator in the UCB algorithm.
- In contrast, each time an action other than a is selected, t increases but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.
- That is the idea or explanation of the UCB algorithm.

yguangbing@gmail.com, Guang.B@chula.ac.th Feb 16 2021

Upper Confidence-bound Action Selection

- Results with UCB on the 10-armed testbed are shown in following Figure.
- UCB generally performs better than ε -greedy action selection, except in the first k steps, when it selects randomly among the as-yet-untried actions.



yguangbing@gmail.com, Guang.B@chula.ac.th Feb 16 2021

Gradient Bandit Algorithms

- It is an algorithm that considers learning a numerical preference for each action a, which we denote $H_t(a)$.
  - larger preference, more frequently the action is selected.
  - but the preference has no interpretation in terms of rewards, which means the larger preference does not imply the larger reward.
  - To keep this constraint, use softmax probability to represent the preference of the action.
  - $Pr(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{K} e^{H_t(b)}} = \pi_t(a)$, where $\pi_t(a)$, denotes for the probability of taking action a at time t.
  - Initially all action preferences are the same (e.g., H(a) = 0, for all a) so that all actions have an equal probability of being selected.

yguangbing@gmail.com, Guang.B@chula.ac.th Feb 16 2021

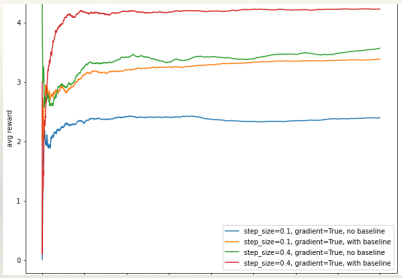
Gradient Bandit Algorithms

- The learning algorithm for this setting is based on the idea of stochastic gradient ascent.
- On each step, after selecting action A and receiving the reward R, the action preferences are updated by:
  - $H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t))$,
  - $H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), \ \forall a \neq A_t$.
  - where $\alpha > 0$ is the step-size parameter,
  - $\bar{R}_t \in \mathbb{R}$, is the average of all the rewards up through and including time t, which can be computed incrementally as described in Incremental approach.
  - $\bar{R}_t$ is the baseline, if reward is higher than the baseline, the probability takes A_t in the future is increased.
  - If it is less than the baseline, it decreases the probability to select the A_t.
  - That is the whole idea of the gradient bandit algorithm.

yguangbing@gmail.com, Guang.B@chula.ac.th Feb 16 2021

Gradient Bandit Algorithms

- Figure at the right side shows results with the gradient bandit algorithm on a variant of the 10- armed testbed in which the true expected rewards were selected according to a normal distribution with a mean of +4 instead of zero (and with unit variance as before).
- This shifting up of all the rewards has absolutely no effect on the gradient bandit algorithm because of the reward baseline term, which instantaneously adapts to the new level. But if the baseline were omitted (that is, if $\bar{R}_t$ was taken to be constant zero), then performance would be significantly degraded, as shown in the figure.



Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the q*(a) are chosen +4 rather than near zero.

Summary

- There are several simple ways of balancing exploration and exploitation.
- The ϵ-greedy methods choose randomly a small fraction of the time with non-greedy approach for exploration.
- UCB methods choose deterministically but achieve exploration by subtly favouring at each step the action that have received fewer samples.
- The Gradient bandit algorithms estimate action preference rather than action values directly, and select the more preferred actions based on probabilities of the preference using a softmax distribution.
- The initial value approach estimates optimistically causes even greedy methods to explore significantly.
- It is not easy to say which method is the best, but UCB performs best overall speaking.

Summary

- Despite these methods discussed in this lecture are simple, they can fairly be considered the state of the art.
- Due to the simplicity of these methods, the approach of balancing the exploration and exploitation is far too simple to be fully satisfactory solutions for practical applications.
- In the Bayesian setting it is even conceivable to compute the optimal balance between exploration and exploitation.
- One can compute for any possible action the probability of each possible immediate reward and the resultant posterior distributions over action values.

Assignment 2

- Assignment 2 worth 15%, and is about multi-armed bandit algorithms.
- same as the assignment 1, copy and download my Colab to your Google drive (Important note: Don't modify my Colab notebook, otherwise other classmates will see your work.)
- Working on your copy of the Colab notebook. Don't forget to add your name and student id in it.
- After finishing it, share it with me (only me, do not share your work with others.)
- All programming exercises MUST be running correctly in Colab without any errors and exceptions. If your code cannot run at all, and I cannot see any kind of outputs, you receive no grade points for that part.
- Before you submit your Colab notebook, make sure to leave the outputs (results) of the functions in the notebook. I ONLY review the outputs of your functions or the final results.
- The assignment due at Mar 16, 2021. It is an individual assignment.
- Just remind you to beware of academic integrity and responsible behaviour.

Questions and Lab