Sequential and reinforcement learning: Stochastic Optimization II

Summary

This session describes the important and nowadays framework of on-line learning and estimation. This kind of problem arises in bandit games (see below for details) and in optimization of big data problems that involve so massive data sets that the user cannot imagine exploiting the entire data with a batch algorithm. Instead, we are forced to use some subset of observations sequentially to produce both : tractable algorithms, fast predictions, efficient statistical estimators.

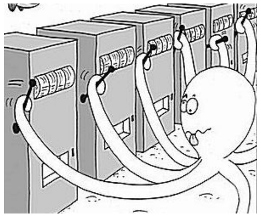
We will describe below some bandit algorithms for stochastic optimization when some important assumptions are made on the structure of the optimization problem. These famous algorithms address a typical situation where we need to adjust sequentially our prediction and our action according to online observations.

At last, we will briefly describe an ultimate method for solving the minimization of a non convex function with multiple local traps, by using simulated annealing. This problem will be illustrated on the so-called travelling salesman problem.

1 Bandit games

1.1 Motivation

- You are in a casino and want to play with slot machines
- Each one can give you a potential gain, but these gains are not equivalent
- You *sequentially* play with one of the arms of the bandit machine

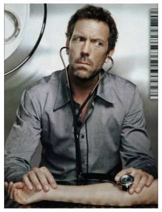


How to design a good policy to sequentially optimize the gain ?

This question is not so obvious because of the possibility you are offered to sequentially adjust your choice of the arm you will play at each round. This “toy” presentation can be indeed examplified in many concrete situations.

1.1.1 Clinical trials

Problem : Optimization of a sequence of clinical trials



Imagine you are a doctor :
- A sequence of patients visit you *sequentially* (one after another) for a given disease
- You choose one treatment/drug among (say) 5 availables
- The treatments are not equivalent
- You do not know where is the best drug, but you observe the effect of the prescribed treatment on each patient
- You expect to find the best drug despite some uncertainty on the effect of each treatment

How can we design a good sequence of clinical trials ?

1.1.2 “Fast fashion” retailer

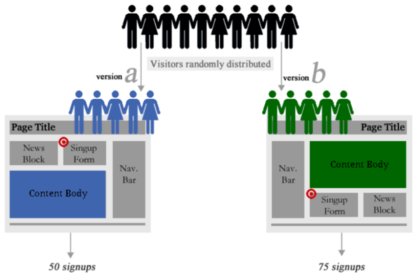
Imagine you are a firm solding clothes :

- A population of customers visit you *sequentially* each week/day
- You observe weekly/daily sales and measure item’s popularity
- You want to restock popular items and weed out unpopular ones *on-line*
- You expect to maximize your benefit while finding the best items

How can we design a good sequence of fast-fashion operations ?

1.1.3 “Web design”

Imagine you want to select a web page design

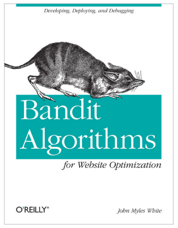


- A population of customers visit you *sequentially* (one after another)
- You randomly propose two designs a and b and measure design’s popularity through the signups you obtain
- You want to propose the popular design to maximize your benefit

How can we build a good sequence of webpage propositions ?

1.1.4 Other motivating examples

- Pricing a product with uncertain demand to maximize revenue
- Trading (sequentially allocate a ratio of fund to the more efficient trader)
- Recommender systems :
  - — advertisement
  - — website optimization
  - — news, blog posts



- Computer experiments
  - — A code can be simulated in order to optimize a criterion
  - — This simulation depends on a set of parameters
  - — Simulation is costly and only few choices of parameters are possible

1.2 Why is this problem difficult ?

Needs a careful mathematical formalization :

What is the underlying optimization problem to be solved ?

The mathematical model probably involves both

- one exploration step (the problem of exploring many possible arms to locate the optimal one)
- one exploitation step (the problem of playing as much as possible the best possible arm)

A trade-off certainly exists between we will have only a finite number of plays (say T in what follows). Hence, we have to manage both views :

Scientist : Explore new ideas / Businessman : Exploit best idea found so far

1.3 Mathematical definition

Environment :

- At your disposal : d arms with *unknown* parameters $\theta_1, \ldots, \theta_d$.
- For any time t, your choice is described by one action $I_t \in \{1 \ldots, d\}$
- For any time t, you receive a reward, that depends on your choice I_t :

$$A_t^{I_t}$$

For example :
— it corresponds to the money obtained by sampling one specific slot machine in a Casino, the number of the machine is I_t.
— it corresponds to the size of a tumor after choosing to test one drug on a patient.

Reward distribution :

- Of course, the rewards cannot be reasonnably assumed to be deterministic (otherwise I won't be there to talk about it !)
- For a fixed choice of the arm i, the rewards are i.i.d.

$$(A_t^i)_{t\geq 0} \sim \nu_{\theta_i}.$$

- Important assumption : the reward distribution ν_θ belongs to a parametric family of probability distributions (Exponential, Poisson, ...).
- One typical example : the reward of arm i is distributed according to a Bernoulli distribution $\mathcal{B}(\theta_i)$:

$$\mathbb{P}[A_t^i = 1] = \theta_i \qquad \text{and} \qquad \mathbb{P}[A_t^i = 0] = 1 - \theta_i.$$

Expected reward : Since the reward of arm i are assumed i.i.d., the expected reward of arm i is given by

$$\forall t \in \mathbb{N} \qquad \mathbb{E}A_t^i = \mathbb{E}A^i$$

1.4 A simplification assumption

In what follows, we will restrict our lecture to the simplest case of Bernoulli rewards $\nu_p = \mathcal{B}(p)$:

- you obtain a gain of 1 with probability p
- 0 otherwise (with probability $1 - p$).

We use now p instead of θ to refer to the unknown parameters.

What is unknown, the several probability of success : $(p_1, \ldots, p_d)$.

Without l.o.g., we assume that the first arm is the best one :

$$p_1 > \max_{2\leq j\leq d} p_j.$$

Admissible policy :

- The agent's action follow a dynamical strategy, which is defined on-line :

$$I_t = \pi\left(A_{t-1}^{I_{t-1}} \ldots, A_1^{I_1}\right).$$

It means that at step t, we can use all the informations gathered from time 1 to time $t-1$ to make our decision I_t.

- The decision I_t can be driven either by
  — a deterministic function
  — a random function
  of the information from 1 to $t-1$.

Final goal : Maximize (in expectation) the cumulative rewards :

$$\mathbb{E}\left[\sum_{t=1}^{n} A_t^{I_t}\right].$$

1.5 Regret of Bandit algorithms

We introduce the so-called cumulative pseudo-regret of any Bandit policy that compares the average performance of the algorithm used with the one if the optimal arm were known :

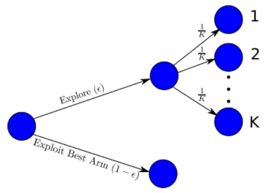
$$\bar{R}_n := \max_{1\leq j\leq d} \mathbb{E}\left[\sum_{t=1}^{n}(A_t^j - A_t^{I_t})\right] = p_1 n - \mathbb{E}\left[\sum_{t=1}^{n} A_t^{I_t}\right],$$

Minimizing $\bar{R}_n$ is equivalent to maximizing the average cumulative rewards of the policy over the n first runs of the algorithm.

2 ϵ-greedy algorithm

2.1 Description of the algorithm

Widely used ϵ-**greedy algorithm**'98 :



- Consider $\epsilon > 0$ and an initial guess of the ability of each arm :

$$\hat{p}_j(0) \qquad \text{is a prior information on} \qquad p_j$$

  If no information, sample p_j at random for example at the initialization step.
- Step t to $t+1$:
  — With probability $1-\epsilon$, use (one of) the best arm according to the belief you have at time t :

$$(\hat{p}_j(t))_{1\leq j\leq d}.$$

  — With probability ϵ/d, pick an arm uniformly among all possibles.
  — Upgrade the estimators of the Bernoulli parameters with the empirical means :

$$\hat{p}_j(t+1) = \frac{1}{n_j(t+1)} \sum_{n=1}^{t+1} A_j^n \mathbf{1}_{\text{arm } j \text{ sampled at time } n},$$

  where $n_j(t+1)$ is the number of times arm j is used from 1 to $t+1$.
- Usually, $\epsilon = 0.1$.

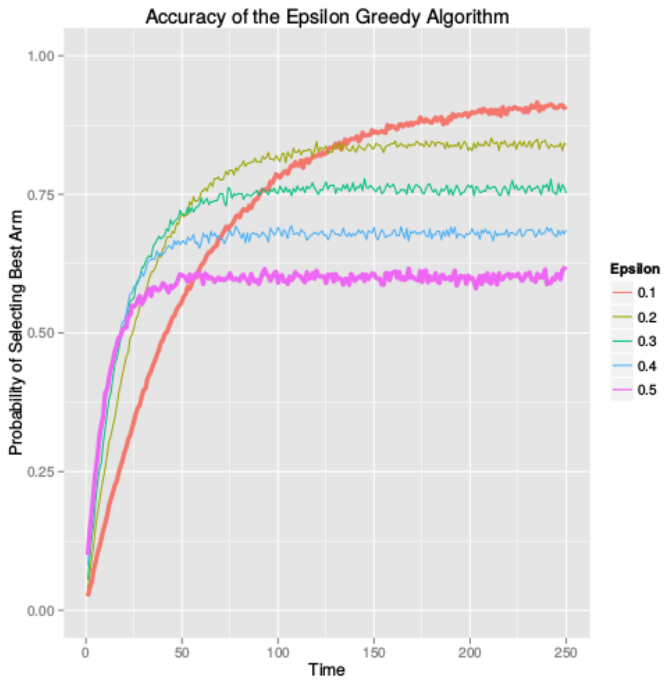
2.2 Pro and cons

Some interesting features of this method :

- Maybe the most simple algorithm to understand
- Maybe the most simple algorithm to program
- Maybe one of the most fastest algorithm to build a recommandation at each step

But unfortunately :

A brief experiment with 5 Bernoulli reward probabilities : $[0.1, 0.1, 0.1, 0.1, 0.9]$



- $\epsilon = 0.1$: Businessman and Learns slowly and Does well at the end
- $\epsilon = 0.5$: Scientist and Learns quickly but Does not exploit at the end

Whatever ϵ is, linear regret with n since the probability to select the best arm does not tend to 1. This probability is indeed $(d-1)/d \times \epsilon > 0$. The regret is at the least

$$\bar{R}_n \geq \epsilon \frac{d-1}{d} \Delta \times n,$$

where

$$\Delta = p_1 - \max_{j \geq 2} p_j.$$

2.3 With Matlab/R

Reproduce the simulation above and change the parameters... You can use the baseline Matlab program :

```
 clear all
close all

T=1000;
eps=0.1;


p=[0.1,0.1,0.48,0.5];
d=length(p);

hatp=rand(1,d);
np=zeros(1,d);
reward=[];
av_reward=[];
for n=1:T

    %Epsilon greedy sampling

    if rand<(1-eps)
     I=find(max(hatp)==hatp);
     i=I(1);
    else
       v=randperm(d);
       i=v(1);
    end

    % Reward
    reward(n)=rand<p(i);

    % Upgrade of the Bernoulli parameter
    hatp(i)=(hatp(i)*np(i)+reward(n))/(np(i)+1);
    np(i)=np(i)+1;

    %Computation of the average reward at time n:
    I=find(max(hatp)==hatp);
    av_reward(n)=mean(p(I))*(1-eps)+eps*mean(p);

end
plot(1:n,(max(p)*(1:n)-cumsum(reward))./(1:n))
xlabel('Iteration')
ylabel('R_n/n')
legend('Trajectorial regret')
figure
plot(1:n,(max(p)*(1:n)-cumsum(av_reward))./(1:n))
xlabel('Iteration')
ylabel('R_n/n')
legend('Average regret')
```

2.4 ϵ-greedy improvement

The idea is to adjust the behaviour of the algorithm with a suitable annealing on ϵ all along the iterations of the algorithm. It can be shown (see [7]) that a suitable scheme is :

$$\forall n \geq 1 \qquad \epsilon_n = 1 \wedge \left\{ c \frac{d}{\Delta^2 n} \right\},$$

where c is a non-negative constant and Δ is an unknown parameters that involves the difference ability between the two best optimal arms :

$$\Delta = p_1 - \max_{j \geq 2} p_j.$$

In practice, Δ is unknown but may be estimated on-line. It can be easily shown that in that case

$$\bar{R}_n \sim \Delta^{-2} \log n,$$

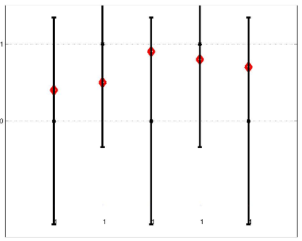
which is the optimal rate that can be achieved when the distributions do not change with n (see [5] for the optimality of this bound).

3 Optimistic algorithms

3.1 Rough description

"Optimal" algorithms may be produced by using an optimistic approach. These optimistic algorithms are the so-called *Upper Confidence Bound* methods (UCB for short). They have been introduced in 1985 in [5]. One of the recent advances on these methods may be found in [2].

We propose a description of the general principle of the algorithm.



Strategy :

- Build a confidence bound around each empirical estimation of the probability of success

$$\hat{p}_i(t) \in [l_i(t); u_i(t)], \forall 1 \leq i \leq d$$

- at time t, select the arm with the highest upper confidence bound :

$$I_t = \arg\max u_i(t).$$

- Get the reward, and update the empirical estimator and the confidence bounds

$$\hat{p}_i(t+1) \in [l_i(t+1); u_i(t+1)]$$

3.2 Pro and Cons

UCB-like algorithm are shown to be optimal and satisfy

$$\limsup_{n \longrightarrow +\infty} \frac{\bar{R}_n}{\log n} \leq \sum_{p<p_1} \frac{1}{2(p_1 - p)},$$

when the distributions of success are kept fixed with the horizon n. Hence, they are (minimax) optimal with respect to the lower bound on all possible strategies (see [5]).

Moreover, they also achieve a distribution free upper bound on the regret which is also optimal (see [1]).

$$\forall (p_1, \ldots, p_d) \in [0,1]^d \qquad \bar{R}_n \lesssim \sqrt{d \log(d) n}$$

In the initial paper, the UCB and LCB computation are obtained with the Gaussian approximation :

$$u_j(t) = \hat{p}_j(t) + \sqrt{2 \frac{\log(t)}{n_j(t)}}$$

where n_j is the number of times arm j is played before time t. The tuned UCB algorithm may take into account the variance of each reward through

$$u_j(t) = \hat{p}_j(t) + \sqrt{\frac{\log(t)}{n_j(t)} \times (1/4 \wedge V(j,t))},$$

with

$$V(j,t) = \hat{p}_j(t)(1 - \hat{p}_j(t)) + \sqrt{2 \log(t)/n_j(t)}.$$

Even though optimal, the initial implementation of the UCB method in [5] does not obtain good enough numerical performance, mainly because of the way the confidence bound u_i and l_i are computed.

This problem is solved in more recent implementations of the UCB algorithm (see for example KL-UCB of [2]), which really transforms the numerical abilities of the algorithm.

Nevertheless, the modification of this method induces a slight weakness of the method : the computation of the UCB and LCB makes the algorithm slower than other methods. Hence, its use crucially depends on the range of application of the method and the time constraints.

3.3 Numerical implementation

The Matlab package may be downloaded through the Aurélien Garivier website : http://www.math.univ-toulouse.fr/~agarivie/Telecom/bandits/.

The package proposes some implementations of some recent Bandit policies (UCB-like, Thompson sampling, Gittin's Bayesian strategy among them [4]). This is an example of use of the KL-UCB algorithm with Matlab for a Bernoulli game with 2 arms.

```
MC=100;
T=1000;
M=10;
policies = {policyUCB()};
tsave = round(linspace(1,T,M));
fname = 'results/KLUCB_bernoulli'
% Run everything one policy after each other
defaultStream = RandStream.getDefaultStream;
savedState = defaultStream.State;


mu=[0.8,0.5]
tic
game = gameBernoulli(mu);
toc
experiment(game, T, MC, policies{1}, tsave, fname);
load ([fname '_n_' num2str(T) '_N_' num2str(MC)
                                  '_policyUCB.mat']);
regret = (0.8*ones(N,1)*tsave)-cumReward;

plot(tsave,mean(regret))
xlabel('Horizon time')
ylabel('Cumulative regret')
legend('Average regret over MC simulations of
                                          KL-UCB')
```

4 Narendra Shapiro algorithm

4.1 Initial algorithm (1969)

The so-called Narendra-Shapiro bandit algorithm (NSa for short) was introduced in [6]. It defines a probability vector of $\mathcal{S}_d$

$$X_t = (X_t^1, \ldots, X_t^d) \qquad | \qquad \sum_{j=1}^{d} X_t^j = 1.$$

Idea : Use X_t to sample one arm at step t and then upgrade this probability X_t.

4.1.1 *Two arm NSa*

- In the two-armed situation with $p_2 < p_1$, denote $X_t = (x_t, 1 - x_t)$
- $X_t(1) = x_t$ is the probability to choose the first arm at step t.
- $X_t(2) = 1 - x_t$ is the probability to choose the second arm at step t.
- Upgrade formula

$$x_{t+1} = x_t + \begin{cases} \gamma_{t+1}(1 - x_t) & \text{if arm 1 is selected and wins} \\ -\gamma_{t+1} x_t & \text{if arm 2 is selected and wins} \\ 0 & \text{otherwise} \end{cases}$$

- Common step size :

$$\gamma_t = (1 + t/C)^{-\alpha}, \qquad \alpha \in (0,1) \qquad \text{with large enough } C.$$

- Same idea :
  — If you win : reinforce the probability to sample I_t w.r.t. the remaining weights $(X_t^j)_{j \neq I_t}$ and decrease the probability to sample the other arms accordingly.
  — If you loose $(A_t^{I_t} = 0)$: do nothing.

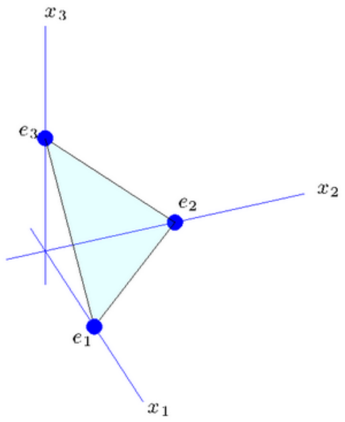
4.1.2 *Multi-armed arm NSa*

Multi-armed situation, I_t : arm sampled at time t, $A_t^{I_t}$: obtained reward. Upgrade

$$\forall j \in \{1 \ldots d\} \qquad X_t^j = X_{t-1}^j + \gamma_t \left[\mathbf{1}_{\{I_t = j\}} - X_{t-1}^j \right] A_t^{I_t}$$

To sum up :

- If you win : reinforce the probability to sample I_t and decrease the probability of others.
- If you loose ($A_t^{I_t} = 0$) : do nothing.



Unfortunately, this algorithm does not perform very well from the regret viewpoint : it has a strictly positive probability to converge towards a wrongly optimal arm, leading to a linear regret.

4.2 Over-penalized NSa (2015)

4.2.1 Two armed over-penalized NSa

To bypass the difficulty of convergence towards a wrong target, we increase the exploration of the algorithm by adding a penalty effect to reinforce the escape from local traps (see [3]).

$$X_{t+1} = X_t + \begin{cases} +\gamma_{t+1}(1-X_t) - \rho_{t+1}\gamma_{t+1}X_t & \text{if arm 1 is selected and wins} \\ -\gamma_{t+1}X_t + \rho_{t+1}\gamma_{t+1}(1-X_t) & \text{if arm 2 is selected and wins} \\ -\rho_{t+1}\gamma_{t+1}X_t & \text{if arm 1 is selected and loses} \\ +\rho_{t+1}\gamma_{t+1}(1-X_t) & \text{if arm 2 is selected and loses} \end{cases}$$

Whatever happens with the selected arm, it is penalized (escape from local traps).

4.2.2 Multi-armed over-penalized NSa

A multi-armed version :

$$\begin{aligned} X_t^j &= X_{t-1}^j + \gamma_t \left[\mathbf{1}_{I_t=j} - X_{t-1}^j \right] A_t^{I_t} \\ &\quad - \gamma_t \rho_t X_{t-1}^{I_t} \left[\mathbf{1}_{I_t=j} - \frac{1-\mathbf{1}_{I_t=j}}{d-1} \right] \end{aligned}$$

4.3 Pro and cons

4.3.1 Theoretical result

It can be shown that the OPNSa has an optimal distribution free minimax regret :

$$\bar{R}_n \lesssim \sqrt{n},$$

up to multiplicative constants that depend on the number of arms. It is a positive result.

But, from a minimax regret point of view, when the distribution of rewards is kept fixed independently of n, the rate is still the same, which is not the case for the KL-UCB method for example. In this settings, the OPNSa is not optimal (regret of the order $\sqrt{n}$ instead of $\log n$).

4.3.2 Numerical results

The (OP)NSa are maybe the fastest (minimax distribution free) optimal Bandit algorithms, and thus may be well suited in some bigdata problems where new observations and decisions should be made in very short times (less than 10^{-3} seconds). This can be the case in finance or website optimization for example.

The implementation is very easy, we just have to design a careful choice for the sequence $(\gamma_t)_{t\geq 0}$ and $(\rho_t)_{t\geq 0}$. A theoretical/practical recommandation from [3] is :

$$\gamma_t = \frac{\gamma}{\sqrt{t}} \quad \text{and} \quad \rho_t = \frac{\rho}{\sqrt{t}} \quad \text{with} \quad \gamma = 1 = 4\rho.$$

You will find below an example of simulation with 2 arms :

```
function Regret=opb2(pa,pb,T,rhot,sigma)
%% Sigma=0: Systematic Over-penalization ([GPP15])
%% Sigma=1: Penalization without over-penalization (LP09])
%% MC:      Number of MC simulation
%  Rhot: value of gamma_n/rho_n
%% Initial value of gamma_n=1/sqrt(C+1)
C=4;

x=1/2;Regret=0;

for i=1:T
    %%% U for the arm choice, V for the reward
    U=rand;V=rand;
    gam=1/sqrt(i+C);rho=rhot*gam;
    if U<x(i)     %% Arm A is chosen
           res=(V<pa);
           x(i+1)=x(i)+gam*(1-x(i))*res
                -rho*gam*x(i)*(1-res*(rand<sigma));
    else       %% Arm B is chosen
           res=(V<pb);
           x(i+1)=x(i)-gam*x(i)*res
           +rho*gam*(1-x(i))*(1-res*(rand<sigma));
    end
   Regret(i+1)=Regret(i)+ (1-x(i+1))*(pa-pb);
end

tic;reg=opb2(0.8,0.5,1000,4,0);toc;

plot(1:1000,reg)
xlabel('Horizon time')
ylabel('Cumulative regret')
legend('Regret of the OPNSa on one trajectory')
```

With the implementations above of KL-UCB and OPNSa, compare the regret attained by each algorithms (repeat a certain number of MC simulations) and the speed of execution of each methods.

Références

[1] Jean Yves Audibert et Sébastien Bubeck, *Regret Bounds and Minimax Policies under Partial Monitoring*, Journal of Machine Learning Research **11** (2010), 2635–2686.

[2] O. Cappé, A. Garivier, O. Maillard, R. Munos et G. Stoltz, *Kullback-Leibler Upper Confidence Bounds for Optimal Sequential Allocation*, Annals of Statistics **41** (2013), 1516–1 541.

[3] S. Gadat, F. Panloup et S. Saadane, *Regret bound for Narendra-Shapiro bandit algorithms*, Stochastic Processes and Applications (2016).

[4] Gittins J.C., *Bandit Processes and Dynamic Allocation Indices*, Journal of the Royal Statistical Society. Series B (Methodological) **41** (1979), 148–177.

[5] T.L. Lai et H. Robbins, *Asymptotically efficient adaptive allocation rules*, Advances in Applied Mathematics **6** (1985), 4–22.

[6] K. S. Narendra et I. J. Shapiro, *Use of stochastic automata for parameter self-optimization with multi-modal perfomance criteria*, IEEE Trans. Syst. Sci. Cybern. **5** (1969), 352–360.

[7] Auer P., Cesa Bianchi N. et Fischer P., *Finite-time Analysis of the Multiarmed Bandit Problem*, Machine Learning **47** (2002), 235 ?256.