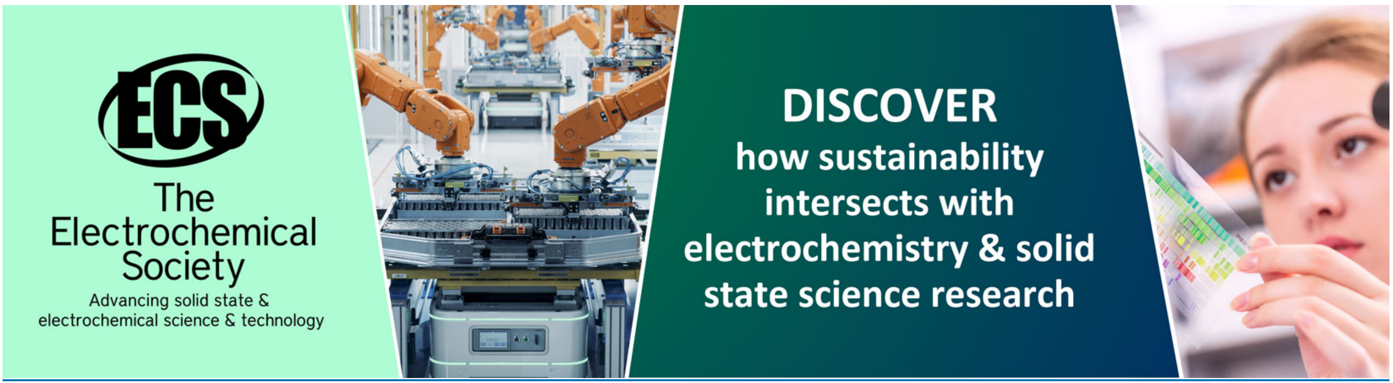
Control in a weakly inhomogeneous two-alternative random environment using the mirror descent algorithm


ECS
The Electrochemical Society
Advancing solid state & electrochemical science & technology
DISCOVER
how sustainability intersects with electrochemistry & solid state science research

Control in a weakly inhomogeneous two-alternative random environment using the mirror descent algorithm

D N Shiyan and A V Kolnogorov

Yaroslav-the-Wise Novgorod State University, ul. B. St. Petersburgskaya, 41
173003 Veliky Novgorod, Russia

E-mail: dsqq.tm@yandex.ru

Abstract. The operation of the mirror descent algorithm in a weakly inhomogeneous random environment is considered. A weakly inhomogeneous random environment describes a data stream each item of which can be processed by two alterative methods, and probabilities of successive processing of both methods may vary during the control process. Algorithms for one-by-one data processing and batch data processing are given. Using numerical modeling, the behavior of the presented algorithms in a weakly inhomogeneous environment is studied and the obtained results are presented.

1. Introduction

Two-armed bandit problem in the minimax setting is considered. The name «two-armed bandit» is used to describe a random environment in which two actions are available, similarly to a slot machine with two arms. In the literature, the problem is described in detail, see, e.g. [1–2]. The important feature of the problem is that the algorithm, operating in the random environment, does not know which of the actions is the most profitable; it can only evaluate them in the control process, gradually receiving information from the random environment. The main goal of the algorithm in this case is to maximize the average expected income for a certain control horizon. This problem has applications in various fields, such as data processing and machine learning.

In what follows, we will consider a two-alternative random environment. Let $X = \{1,2\}$ be the set of actions available. At each instant of time $n = 1,2,\dots,N$ we can select only one of the available actions denoted by x_n. The choice of the action can bring income (1) with probability ρ_{x_n} or bring nothing (0) with probability $1-\rho_{x_n}$. In previous articles, for example, in [3], strictly stationary probability distributions were considered $\overline{\rho} = (\rho_1, \rho_2)$, in which $\overline{\rho}$ remains unchanged throughout the control horizon. However, in real-life problems such a condition is rarely performed because physical processes tend to have some randomness. Therefore, below we will consider a weakly inhomogeneous environment, defined as a Markov chain with two states S_1 and S_2:

$$\begin{aligned} S_1 &: \overline{\rho} = (\rho_1' - s; \rho_2' - s), \\ S_2 &: \overline{\rho} = (\rho_1' + s; \rho_2' + s), \end{aligned} \tag{1}$$

where ρ_x' – is the fixed probability of winning when choosing action x, $s \ge 0$ – the deviation of probability of winning. We believe that the chain is homogeneous and has the following transition matrix

$$P = \begin{pmatrix} 1-\Delta & \Delta \\ \Delta & 1-\Delta \end{pmatrix}, \tag{2}$$

where $0 < \Delta < 1$ is fixed transition probability.

This environment describes the inhomogeneity of the input data with which the algorithm operates. In a weakly inhomogeneous environment, the probabilities of winning can vary within certain limits, and this causes additional problems in evaluating these probabilities.

We introduce the loss function on the control horizon N as follows

$$L_N = \sum_{n=1}^{N} (\max(\rho_n^{(1)}, \rho_n^{(2)}) - \xi_n), \tag{3}$$

where ξ_n is the income at instant of time n. Next, we will consider the loss function in a normalized form.

$$L'_N = \frac{L_N}{\sqrt{DN}} = \frac{1}{\sqrt{DN}} \sum_{n=1}^{N} (\max(\rho_n^{(1)}, \rho_n^{(2)}) - \xi_n), \tag{4}$$

where $D = 0.25$ is the maximum possible variance of income.

We will consider the problem of minimizing the loss function L'_N. To solve the problem, we will use the mirror descent algorithm. This algorithm and theoretical estimates for it in the case of one-by-one data processing are given in [4]. In [5], batch versions of the mirror descent algorithm are given.

The structure of the article is as follows. Section 2 describes the basic mirror descent algorithm. Section 3 discusses descriptions of algorithms optimized for batch processing. Section 4 contains the results of numerical simulation of the operation of algorithms in a weakly inhomogeneous random environment.

2. Algorithm for one-by-one data processing

Consider the basic mirror descent algorithm based on the algorithm from [4], but with some changes.

We introduce a probability vector $\bar{p}_n = (p^{(1)}; p^{(2)})$, dual vector $\bar{\zeta}_n = (\zeta^{(1)}; \zeta^{(2)})$ and vector of stochastic gradient $\bar{u}_n = (u^{(1)}; u^{(2)})$. The description of the algorithm uses the Gibbs distribution which is defined as follows

$$\bar{G}_\beta(\bar{\zeta}) = \frac{1}{S_\beta(\bar{\zeta})} \cdot \left(e^{-\zeta^{(1)}/\beta}; e^{-\zeta^{(2)}/\beta} \right), \; S_\beta(\bar{\zeta}) = e^{-\zeta^{(1)}/\beta} + e^{-\zeta^{(2)}/\beta}. \tag{5}$$

Algorithm 1

Set the initial values $\bar{p}_0 = (0.5; 0.5)$ and $\bar{\zeta}_0 = (0; 0)$.

For all $n = 1, \dots, N$ perform the following operations:

1) Choose the action y_n according to the vector $\bar{p}_{n-1}$: $P(y_n = l) = p_{n-1}^{(l)}$, $l = 1,2$;

2) Get the income ξ_n for the use of action y_n:

$$\begin{aligned} P(\xi_n = 1 \mid y_n = l) &= \rho_l, \\ P(\xi_n = 0 \mid y_n = l) &= 1 - \rho_l; \end{aligned} \tag{6}$$

3) Calculate the stochastic gradient $\bar{u}_n(\bar{p}_{n-1})$ according to the following formula:

$$\bar{u}_n(\bar{p}_{n-1}) = \begin{cases} \left(\frac{1-\xi_n}{p_{n-1}^{(1)}};0 \right), & y_n = 1, \\ \left(0;\frac{1-\xi_n}{p_{n-1}^{(2)}} \right), & y_n = 2; \end{cases} \tag{7}$$

4) Update vectors $\bar{\zeta}_n$ and $\bar{p}_n$ as follows:

$$\begin{aligned} \bar{\zeta}_n &= \bar{\zeta}_{n-1} + \bar{u}_n(\bar{p}_{n-1}), \\ \bar{p}_n &= \bar{G}_{\beta_n}(\bar{\zeta}_n), \end{aligned} \tag{8}$$

where $\beta_n = \beta_0 \sqrt{D(n+1)} = \beta_0 \cdot 0.5\sqrt{n+1}$, $\beta_0 > 0$ is a tunable parameter of the algorithm.

3. Algorithms for batch data processing

The above algorithm demonstrates good results, but it processes the data one-by-one, that is, after applying each action, it waits for the environment to respond. However, there are real-life tasks in which the reaction time of the environment can take quite a long time. One of the approaches to optimization in this case is the use of batch processing. For a batch, the execution of actions can be run in parallel, after receiving the response of the environment to the execution of each of them, the algorithm evaluates and proceeds to process the next batch.

With batch processing, denote $N = TM$, where M is a batch size, T is a number of stages. Thus, the total processing time is no longer dependent on the control horizon, but depends only on the number of stages T.

Consider the first version of the algorithm for batch processing. It involves the consistent use of actions within the batch without recalculating the vector $\bar{p}_n$.

Algorithm 2

Set the initial values $\bar{p}_0 = (0.5;0.5)$ and $\bar{\zeta}_0 = (0;0)$.

For all $t = 1,\dots,T$ perform the following operations:

1) Introduce the vector $\bar{\chi}_t = (\chi^{(1)};\chi^{(2)}) = (0;0)$.

2) Processing group t. For all $n = (t-1)M+1,\dots,tM$:

a) Choose the action y_n according to the vector $\bar{p}_{n-1}$: $P(y_n = l) = p_{n-1}^{(l)}, l = 1,2$;

b) Get the income ξ_n for the use of the action y_n using (6);

c) Update vector $\bar{\chi}_t$: $\chi_t^{(y_n)} = \chi_t^{(y_n)} + (1-\xi_n)$.

3) Calculate the stochastic gradient $\bar{u}_t(\bar{p}_{t-1})$ according to the following formula:

$$\bar{u}_t(\bar{p}_{t-1}) = \left(\frac{\chi_t^{(1)}}{p_{t-1}^{(1)}}, \frac{\chi_t^{(2)}}{p_{t-1}^{(2)}} \right) \tag{9}$$

4) Update vectors $\bar{\zeta}_t$ and $\bar{p}_t$ using (8).

Algorithm 2 is very similar to Algorithm 1 (in the particular case when $M = 1$ it replicates it completely), but it already allows one to optimize the processing time due to parallel execution of actions.

Consider another variant of the algorithm, which divides the batch in proportion to the vector $\bar{p}$ and then performs the appropriate action for each of the parts.

Let vector $\overline{M}_t = (M_t^{(1)}; M_t^{(2)})$, $M_t^{(1)} > 0,\ M_t^{(2)} > 0$ and $M_t^{(1)} + M_t^{(2)} = M$. Denote by $[M_t^{(1)}], [M_t^{(2)}]$ the nearest integers to $M_t^{(1)}, M_t^{(2)}$.

Algorithm 3

Set the initial values $\overline{p}_0 = (0.5;0.5)$ and $\overline{\zeta}_0 = (0;0)$.

For all $t = 1,\dots,T$ perform the following operations:

1) Calculate for which part of the batch the first action should be used and for which the second: $M_t^{(l)} = p_{t-1}^{(l)} \times M,\ l = 1,2$;

2) Apply the l –th action $[M_t^{(l)}]$ times and calculate the total vector of income $\overline{\eta}_t$, where ξ_n is calculated by (6):

$$\eta_t^{(l)} = \sum_{n=(t-1)M+1}^{tM} (1 - \xi_n \mid y_n = l) \tag{10}$$

3) Calculate the stochastic gradient $\overline{u}_t(\overline{p}_{t-1})$ according to the following formula:

$$\overline{u}_t(\overline{p}_{t-1}) = \left(\frac{\eta_t^{(1)}}{p_{t-1}^{(1)}}, \frac{\eta_t^{(2)}}{p_{t-1}^{(2)}} \right) \tag{11}$$

4) Update vectors $\overline{\zeta}_t$ and $\overline{p}_t$ using (8).

Algorithm 3, in addition to the advantages of parallel processing, also makes it possible to reduce total expected losses compared with Algorithm 1.

In some cases, with a small number of stages, we can combine Algorithm 1 and Algorithm 3, first performing the evaluation using Algorithm 1 in a small initial stage, and then continue processing with Algorithm 3.

Algorithm 4

1) Apply Algorithm 1 for $n = 1,\dots,M_0$ and get estimates $\overline{p}_{M_0}$ and $\overline{\zeta}_{M_0}$.

2) Apply Algorithm 3 for $n = M_0 + 1,\dots,N$, using $\overline{p}_0 = \overline{p}_{M_0}$ and $\overline{\zeta}_0 = \overline{\zeta}_{M_0}$ as initial values.

4. Simulation results

We will perform numerical simulations of the operation of algorithms in a weakly inhomogeneous random environment. In the simulation, we will consider the case when the success probabilities in choosing actions differ by a small amount of the order of $d/\sqrt{N}$, since otherwise the task of choosing the optimal action is greatly simplified. Define fixed values for success probabilities $\overline{\rho}'$ as follows

$$\overline{\rho}' = (\rho_1', \rho_2') = (0.5 + d\sqrt{D/N}, 0.5 - d\sqrt{D/N}), \tag{12}$$

where $d > 0$ – is the difference between the probabilities of winning when choosing actions. Next, we will use the values of $\overline{\rho}'$ in expression (1) to determine the outcomes of the Markov chain describing a weakly inhomogeneous environment.

On figure 1, simulation results for Algorithm 1 with $s = 0.2$ and different Δ are presented. Control horizon is equal to $N = 10000$.

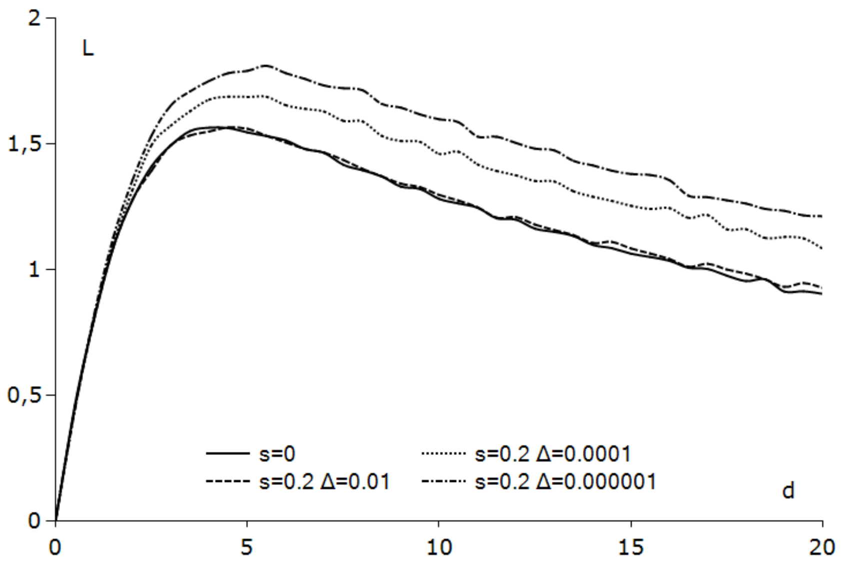


Figure 1. Normalized losses for Algorithm 1.

One can see that the most difficult for the algorithm are cases of small inhomogeneity with $\Delta \leq 1/N$ in these cases, the greatest increase in losses is observed relative to the losses of the algorithm in a stationary environment ($s=0$).With such values of Δ, for the entire time of control, the switching of states occurs no more than once and the increase in losses is quite explicable. However, another interesting fact is that when $\Delta > 1/N$, the losses of the algorithm in a weakly inhomogeneous environment virtually do not differ from the losses in a stationary environment, that is, the algorithm adapts to changes in the environment, despite the fact that they occur quite often.

On figure 2, simulation results for Algorithm 2 with $s=0.2$ and different Δ are presented. Control horizon $N=10000$, batch size $M=100$, number of stages $T=100$.

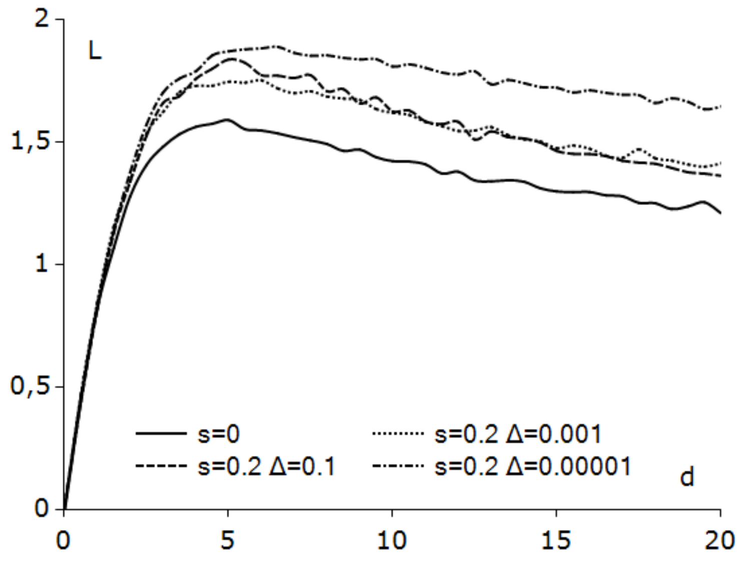


Figure 2. Normalized losses for Algorithm 2.

Algorithm 2 shows a similar behavior with Algorithm 1, however, it shows higher losses due to the fact that it is optimized for batch processing and applies actions more often than it evaluates probabilities.

On figure 3, simulation results for Algorithm 3 with $s = 0.2$ and different Δ are presented. Control horizon $N = 10000$, batch size $M = 100$, number of stages $T = 100$.

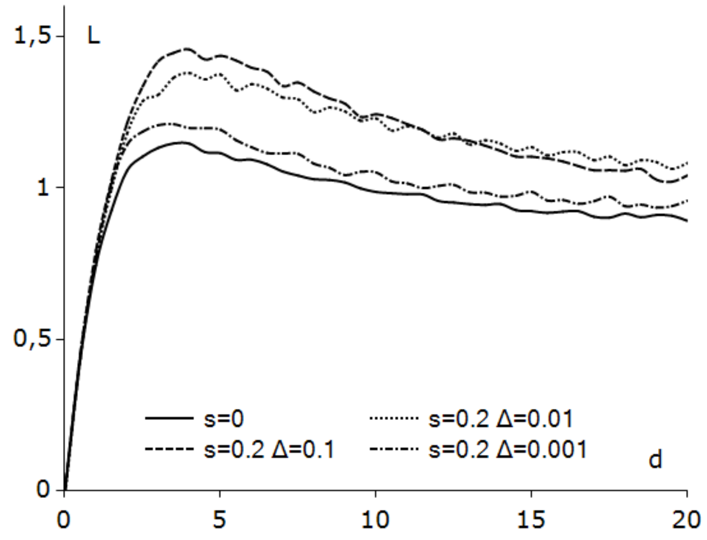


Figure 3. Normalized loss for Algorithm 3.

Algorithm 3 is more resistant to single state switching for small Δ, since it uses a different approach to splitting a batch. In addition, Algorithm 3 retains an advantage over Algorithm 1, demonstrating lower losses, not only in a stationary environment, but also in a weakly inhomogeneous environment.

5. Conclusion

We reviewed the operation of the mirror descent algorithm on inhomogeneous data sets. Four algorithms were presented, one for one-by-one data processing and three modifications of the algorithm for batch data processing.

Batch data processing allows one to optimize the total control time by applying actions in parallel for each group. With Algorithm 3, batch data processing can also reduce total losses.

Using numerical simulations, the operation of the algorithms on sufficiently large control horizons in different inhomogeneous environments was considered. An important result is the stable work of the presented algorithms with inhomogeneous data sets. Algorithms are able to adapt to changes in input data without a significant increase in expected losses.

References

[1] Berry D A and Fristedt B 1985 *Bandit Problems: Sequential Allocation of Experiments* (Chapman & Hall: London)
[2] Cesa-Bianchi N and Lugosi G 2006 *Prediction, Learning, and Games* (Cambridge: Cambridge Univ. Press)
[3] Kolnogorov A V and Shiyan D N 2016 *Proc. of the 3rd International Conference on Mathematics and Computers in Sciences and in Industry MCSI (Chania)* 241–245
[4] Juditsky A *et al* 2008 *Proc. of the 17th World Congr. of the International Federation of Automatic Control (Seoul)* vol 17 11560–11563
[5] Kolnogorov A V, Nazin A V and Shiyan D N 2017 Two-armed bandit problem, data processing, and parallel version of the mirror descent algorithm *Preprint* arXiv:1705.09977