CS769 Project
Comparison of Multi-Armed Bandits Policies

Prishat Bachhar
213050078

Subir Kumar Parida
21v980019

Yashwant Raghuwanshi
213050002

May 05, 2022

Contents

1 Introduction

In Online Convex optimization, a player iteratively makes certain decisions without knowing their impact. After making the decision, the player suffers a loss. The player's goal is to learn from the decisions and play efficiently to maximize the rewards. This can be formally defined as follows:

Let K be the convex decision set, and F be the set of convex functions. Let us consider a repeated game of T rounds between a player and an adversary, then at each round t ∈ T,

- player chooses a point $x_t \in K$.
- adversary independently chooses a loss function $f_t \in F$.
- player suffers a loss $f_t(x_t)$ and receives a feedback F_t.

Here, if the entire convex loss function oracle is displayed to the player, the OCO setting is known as *Full Information Setting*. In contrast, when only the value of the loss function evaluated at the chosen point x_t is provided back to the player as the feedback, the setting is commonly known as *Bandit Convex Optimization (BCO)*.

The metric used to analyze a player's strategy comes from game theory and is called the regret bound. Let A be an OCO algorithm that maps the game history to a decision within the decision set.

$$x_t^A = A(f_1, f_2, ..., f_{t-1}) \in K$$

Then we can define the regret of the player after T iterations as:

$$Regret_T = \sum_{t=1}^{T} f_t(x_t) - \min_{x^* \in K} \sum_{t=1}^{T} f_t(x^*)$$

The optimization algorithms would aim to minimize the Expected Regret.

2 Online Gradient Descent

The OGD algorithm is a first-order approximation of the convex loss function. It is given as below.

Algorithm 1: Online Gradient Descent

Input: Convex set $K, T, x_1 \in K$, step sizes $\{\eta_t\}$

for $t = 1$ *to* T **do**

  Play x_t and observe loss $f_t(x_t)$

  Update and Project:

    $y_{t+1} = x_t - \eta_t \nabla f_t(x_t)$

    $x_{t+1} = \prod_K (y_{t+1})$

end

The OGD algorithm is similar to the stochastic gradient descent algorithm for the offline convex optimization setting. The input to the algorithm is the convex domain, from where the player

chooses the input points and the set of convex loss functions, one among which is selected by the adversary at each time step t, and the feedback as the value of the loss function is provided back to the player. Fig. 1 shows the projection of the point y_{t+1} back into the convex domain x_{t+1}.

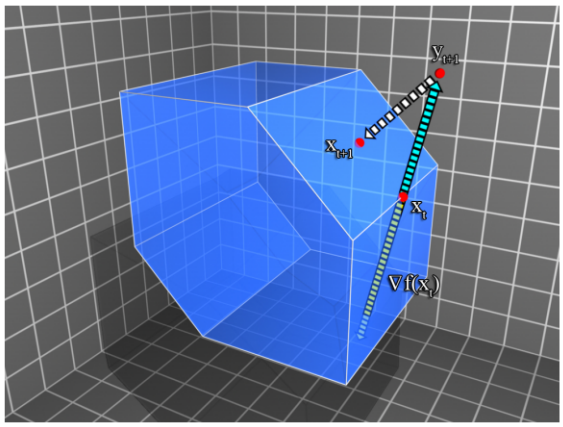


Figure 1: OGD: the iterate x_{t+1} is derived by advancing x_t in the direction of the current gradient ∇_t, and projecting back into K

2.1 Bandit Convex Optimization

It is similar to the basic OCO model with a caveat. The player does not have access to the entire gradient oracle over the decision set K. In the BCO model, the player receives the value of the loss function $f_t(x_t)$ at time t as the only feedback. In the following sections, we explore the Multi-Armed Bandit problem, an instance of the BCO model.

3 Multi Arm Bandit (MAB)

This is a Bandit Convex Optimization setting where the player has arms or decisions to be made. The player's goal is to strike a balance between exploring the options and using the information learned so far to achieve a reward. This is encapsulated in the **Exploration vs. Exploitation trade-off**. The exploration vs. exploitation trade-off, which is an integral part of human lives (having its origin in slot machines), is dealt with in the MAB. One way of dealing with this is to separate the two things and assign a probability of exploring and exploiting the information for the rest of the time (Simple MAB or ϵ-greedy algorithm).

3.1 Algorithm - ϵ-greedy

The simple MAB algorithm or the ϵ greedy algorithm is given in algorithm 2. The Bernoulli random variable denotes the probability of exploring the decisions and exploiting the current loss values known.

Algorithm 2: Simple MAB Algorithm

Input: OCO Algorithm A, parameter δ

for $t = 1$ *to* T **do**
  The choice b_t is a Bernoulli random variable with parameter δ.
  if $b_t == 1$ **then**
    Move $i_t \in \{1, 2, ...n\}$ is chosen uniformly at random and that move is played.
    Let

$$\hat{l}_t(i) = \begin{cases} \frac{n}{\delta} l_t(i_t), & if\ i = i_t \\ 0, & otherwise \end{cases}$$

    Let $\hat{f}_t(x) = l_t^T x$ and update $x_{t+1} = A(f_1, f_2, ..., f_{t-1})$
  end
  else
    choose $i_t \sim x_t$ and play i_t
    Update $\hat{f}_t = 0, \hat{l}_t = 0, x_{t+1} = x_t$
  end
end

3.1.1 Mathematical Bound

The following theorem is used to derive the bound on the performance of a simple BCO algorithm. Here, G is the Lipschitz continuity factor bounding the gradient, and D is the maximum distance between any 2 points in K, bounding the points.

Theorem: *Online gradient descent with step sizes* $\{\eta_t = \frac{D}{G\sqrt{t}}\}, t \in [T]$ *guarantees the following for all* $T \geq 1$

$$Regret_T = \sum_{t=1}^{T} f_t(x_t) - \min_{x^* \in K} \sum_{t=1}^{T} f_t(x^*) \leq \frac{3}{2} GD\sqrt{T}$$

Lemma

$$\boldsymbol{E}[l_t(i_t)] \leq \boldsymbol{E}[\hat{l}_t^T(x_t)] + \delta$$

Proof:

$$\boldsymbol{E}[l_t(i_t)] = Pr[b_t = 1]\boldsymbol{E}[l_t(i_t)|b_t = 1] + Pr[b_t = 0]\boldsymbol{E}[l_t(i_t)|b_t = 0]$$

$$\leq \delta + Pr[b_t = 0]\boldsymbol{E}[l_t(i_t)|b_t = 0]$$

$$= \delta + (1 - \delta)Pr[b_t = 0]\boldsymbol{E}[l_t^T x_t|b_t = 0]$$

$$\leq \delta + \boldsymbol{E}[l_t^T x_t]$$

$$= \delta + \boldsymbol{E}[\hat{l}_t^T x_t]$$

[Hence Proved]

This can be used to put a bound on the regret attained by our algorithm.

$$\begin{aligned}
\boldsymbol{E}[Regret_T] &= \boldsymbol{E}[\sum_{t=1}^{T} l_t(i_t) - \sum_{t=1}^{T} l_t(i^*)] \\
&= \boldsymbol{E}[\sum_{t=1}^{T} l_t(i_t) - \sum_{t=1}^{T} \hat{l}_t(i^*)] \\
&\leq \boldsymbol{E}[\sum_{t=1}^{T} \hat{l}_t(x_t) - \min_i \sum_{t=1}^{T} \hat{l}_t(i)] + \delta T \\
&\leq \frac{3}{2} GD\sqrt{\delta T} + \delta T \leq 3\frac{n}{\sqrt{\delta}}\sqrt{T} + \delta T \\
&= O(T^{2/3} n^{2/3}) \; for \; \delta = n^{2/3} T^{-1/3}
\end{aligned}$$

3.2 ϵ-decay Algorithm

In the previous ϵ-greedy algorithm, the value of ϵ plays a vital role. With a low ϵ value, the algorithm may not explore enough to know the best choice, and with a high ϵ value, the algorithm is forced to choose other options even when it is somewhat sure of the best one reducing the total potential reward. For a high ϵ, the algorithm has $\frac{\epsilon}{n}$ chance to choose a bad arm even when it is almost sure of the best one. The idea is to keep reducing the value of ϵ over time (i.e., with iterations). The value of *epsilon* is taken as $\frac{1}{(1+\frac{t}{no.of arms})}$.

3.3 UCB Algorithm

This algorithm is based on reducing the uncertainty about the reward associated with all possible choices. It is given as:

$$Q(a) + \sqrt{\frac{2 \ln t}{N_t(a)}}$$

where $Q(a)$ is the reward of action a, $N_t(a)$ is the number of times action a has been chosen till time t. The term under the square root determines the uncertainty. As the value of $N_t(a)$ increases (action a has been chosen a few times), the uncertainty of the reward associated with a reduces. The first term signifies exploitation, while the second exploration.

3.4 Thompson Sampling

The Beta distribution given as:

$$Beta(\theta; \alpha; \beta) = \frac{1}{C} \theta^{\alpha - 1} (1 - \theta)^{\beta - 1}$$

Where C is the normalization constant, in this algorithm, for each action, the number of times a high reward is obtained and the no. of times a low reward is obtained is recorded and used as α and β. This Beta distribution approximates how likely it is to get a high reward for choosing the action. Thus, the next action is chosen as the one with the highest value of $Beta$ given their respective α and β.

4 Contextual Bandits

Contextual bandits can be referred to as an extension of multi-armed bandits but with some contextual information about the arms. The algorithm will observe the context of the arms, make a decision and choose an action from the various possible actions. It will then observe the outcome of the decision taken.

The dataset used in the experiment is the Bibtex dataset. It is a multi-label text classification dataset for automatic tag suggestion. The labels correspond to the tags that people have assigned to various papers in the dataset. The goal is to learn to suggest the tags (labels) based on features (input) from the papers.

Prepossessing: For each training instance of the data-set (for each paper), there can be a different number of tags associated with that instance compared to any other instance. Also, the suggested tags corresponding to a data-set instance are concatenated with the training instance. Hence, prepossessing of the Bibtex data-set needs to be done before actually implementing the epsilon-greedy algorithm to get a binary classification for each label as we do the OneVsRest approach of logistic Regression.

Algorithm:
As stated below, in every iteration on the underlying data-set, the algorithm makes a random choice between exploiting the known distributions or exploring the distributions to know them better, the probability of which path is chosen depends on the epsilon parameter being passed.

Explore: If exploration is being Chosen, we will Randomly choose a label/bandit, add the reward that we get by choosing this bandit, and add the corresponding training instance and the reward gained into the history of the bandit. Also, fit the bandit Model when "n" number of new instances are added in the history of the bandit.

Exploit: While Exploiting, we will Predict the reward corresponding to each label model, choose the one with max probability and get the corresponding output with respect to the selected label/bandit. Also, **MAB-FIRST** algorithm is employed, which gives a probability score for a particular label if it has not seen enough instances to distinguish them.

Algorithm 3: $\epsilon - greedy$ for Contextual Bandits

Input: Probability $p \in (0, 1]$, decay rate $d \in (0, 1]$, oracles $\hat{f}_{1:k}$
for *each successive round t with context* $\boldsymbol{x}^t$ *do* **do**
  with probability (1-p):
    select action $a = argmax_k \hat{f}_k(x^t)$
  Otherwise:
    select action a uniformly randomly from 1 to k.
  Update $p := p * d$
  obtain reward r_a^t, add observation x^t, r_a^t to history for arm a
  update oracle $\hat{f}_a$ with its new history
end

Algorithm 4: MAB First

Input: const a,b, threshold m, contextual bandit policy π_k, covariates x
Output: score for arm $\hat{r}_k$

if $|\{r \in R | r = 0\}| < m$ *or* $|\{r \in R | r = 1\}| < m$ **then**
  sample $\hat{r}_k \sim Beta(a + |\{r \in R | r = 1\}|, b + |\{r \in R | r = 0\}|)$
end
else
  set $\hat{r}_k = \pi_k(x)$
end
return $\hat{r}_k$

5 Experiments

5.1 Section 1

Here the data contains five arms with a loss obtained while choosing an action is randomized between 1 to 5 for each arm. The experimental results in Fig. 2 show the loss obtained over time using the simple MAB algorithm with Online Gradient Descent as the OCO algorithm for different values of exploration probabilities.

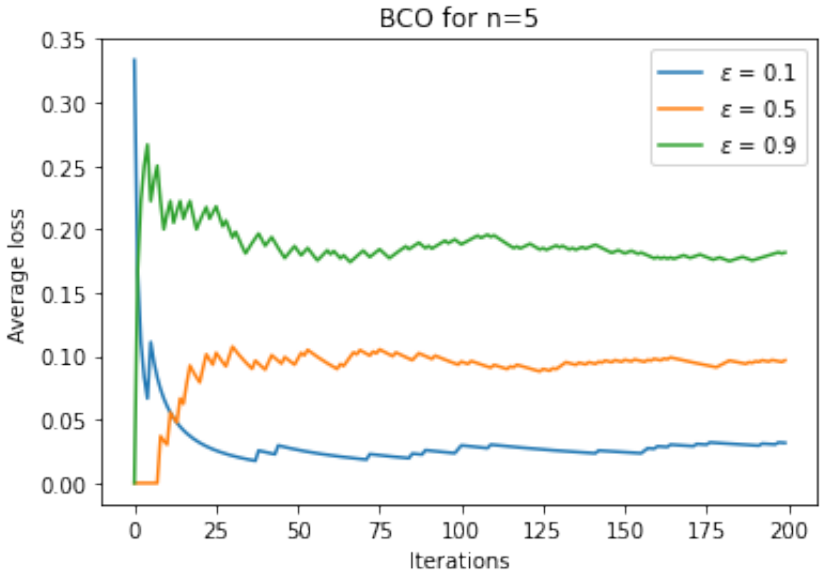
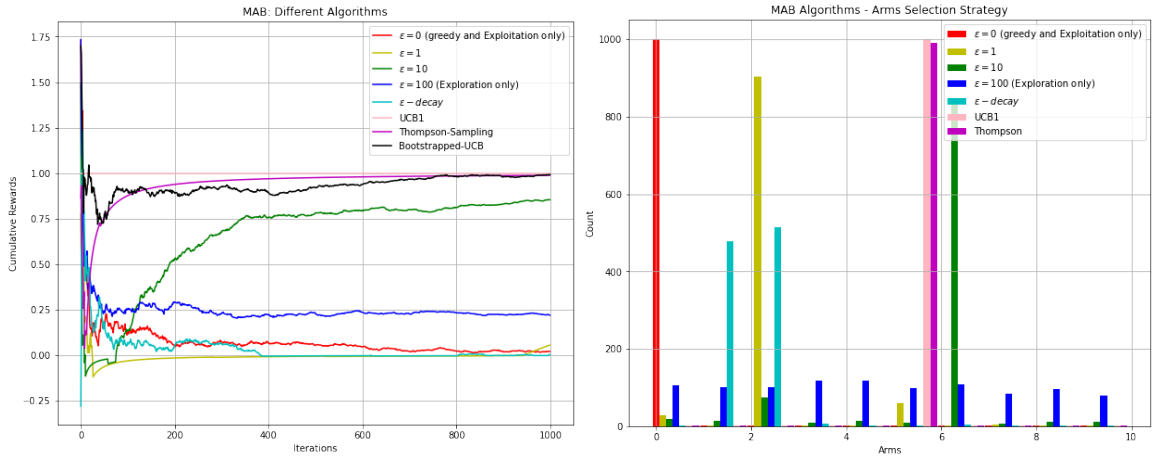


Figure 2: BCO with OGD

It can be seen from Fig. 2 that a 10% exploration rate is compared to higher exploration rates. This is due to the fact that for high exploration, the algorithm is forced to choose an action at random even though it is almost sure of the optimal choice.

5.2 Section 2

The data consists of 10 arms, each with a randomized reward of mean 0 to 1 and a standard deviation of 0 to 1. The next Fig. 3a shows the different algorithms compared with each other and different values of ϵ for the MAB problem setting. The histogram plot in Fig. 3b shows the number of times the given algorithm chooses an arm.



(a) Various MAB algorithms

(b) MAB Arms selection

Figure 3: Multi-Arm Bandits

As evident from the Fig. 3a, Thompson Sampling, Bootstrapped UCB and UCB1 policies gives the highest cumulative rewards at 1000 iterations. UCB1 rises logarithmically due to the presence of log term in its equation. The Exploitation only policy performs the worst here, which justifies that it choose an arm at the early stage and never bothered to explore for maximizing returns.

6 Critical Analysis

Our analysis of the experimentation is provided as below:

1. In the context of **MAB**, a higher probability of exploration leads to more regret as the algorithm is forced to explore bad choices.
2. **ϵ-greedy family:** The algorithms are not at all concerned towards obtaining the true mean of an arm for exploitation
3. **UCB, Thompson:** The algorithms tries to reduce the uncertainty about the true reward of an arm
4. **UCB**
    - eliminates the randomness of the -greedy family algorithms
    - The algorithm gives the benefit of doubt to a less explored arm at a certain time t, and explores it to identify the true distribution/reward.

5. **Thompson:**

   - Tries to select the arm which has previously given more number of high rewards.
   - It tries to understand the true distribution of the rewards associated with the arms.

7 Link to source code

The Github link to the source code is here.

References

[1] Hazan E. (2016). Introduction to online convex optimization. Foundations and Trends® in Optimization. 2(3-4):157-325.

[2] Cortes, D. (2018). Adapting multi-armed bandits policies to contextual bandits scenarios. arXiv preprint arXiv:1811.04383.

[3] Lattimore, T., Szepesvári, C. (2020). Bandit algorithms. Cambridge University Press.

[4] The intuition behind Thompson Sampling. Last accessed on 3^{rd} May, 2022. `https://analyticsindiamag.com/thompson-sampling-explained-with-python-code/`