Online Mirror Descent Algorithm with and without Expert Advice: A Study

Department of Computer Science, RKMVERI, Belur Math

Pratik Karmakar
MSc BDA 2020 batch, third semester (Aug 2021-Jan 2022)
Under supervision of Mrinmay Maharaj
Course: Online Learning DA225

1 The Framework of Online Learning

In OCO, an online player iteratively makes decisions. At the time of each decision, the outcome or outcomes associated with it are unknown to the player.
After committing to a decision, the decision maker suffers a loss: every possible decision incurs a (possibly different) loss. These losses are unknown to the decision maker beforehand. The losses can be adversarially chosen, and even depend on the action taken by the decision maker.
Already at this point, several restrictions are necessary in order for this framework to make any sense at all:

- The losses determined by an adversary should not be allowed to be unbounded. Otherwise, the adversary could keep decreasing the scale of the loss at each step, and never allow the algorithm to recover from the loss of the first step. Thus, we assume that the losses lie in some bounded region.
- The decision set must be somehow bounded and/or structured, though not necessarily finite.

Surprisingly, interesting statements and algorithms can be derived with not much more than these two restrictions. The online convex optimization (OCO) framework models the decision set as a convex set in Euclidean space denoted as $V \in \mathbb{R}^n$. The costs are modeled as bounded convex functions over V.
The OCO framework can be seen as a structured repeated game. The protocol of this learning framework is as follows.
At iteration t, the online player chooses $x_t \in V$. After the player has committed to this choice, a convex cost function $f_t \in F : V \to \mathbb{R}$ is revealed.
Here, F is the bounded family of cost functions available to the adversary. The cost incurred by the online player is $f_t(x_t)$, the value of the cost function for the choice x_t.
As the framework is game-theoretic and adversarial in nature, the appropriate performance metric also comes from game theory: define the regret of the decision maker to be the difference between the total cost she has incurred and that of the best fixed decision in hindsight. In OCO, we are usually interested in an upper bound on the worst-case regret of an algorithm.

Let A be an algorithm for OCO, which maps a certain game history to a decision in the decision set:

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

We formally define the regret of A after T iterations as:

$$Regret_T(A) = \sup_{\{f_1,...,f_T\}\subseteq F} \left\{ \sum_{t=1}^{T} f_t(x_t^A) - \min_{x\in V} \sum_{t=1}^{T} f_t(x) \right\}$$

If the algorithm is clear from the context, we henceforth omit the superscript and denote the algorithm's decision at time t simply as x_t. Intuitively, an algorithm performs well if its regret is sublinear as a function of T (i.e. $Regret_T(A) = o(T)$), since this implies that on average, the algorithm performs as well as the best fixed strategy in hindsight.
The running time of an algorithm for OCO is defined to be the worst case expected time to produce x_t, for an iteration $t \in [T]^2$ in a T-iteration repeated game. Typically, the running time will depend on n (the dimensionality of the decision set V), T (the total number of game iterations), and the parameters of the cost functions and underlying convex set.

2 Online Mirror Descent

In the convex optimization literature, "Mirror Descent" refers to a general class of first order methods generalizing gradient descent. Online Mirror descent (OMD) is the online counterpart of this class of methods. This relationship is analogous to the relationship of online gradient descent to traditional (offline) gradient descent.
OMD is an iterative algorithm that computes the current decision using a simple gradient update rule and the previous decision, much like OGD. The generality of the method stems from the update being carried out in a "dual" space, where the duality notion is defined by the choice of regularization: the gradient of the regularization function defines a mapping from $\mathbb{R}^n$ onto itself, which is a vector field. The gradient updates are then carried out in this vector field.
We have used two variants of OMD as described below:

In Algorithm 1, we first take a gradient descent step and then project the point on the closed convex action set V. The projection is done w.r.t Bregman divergence. Bregman divergence $B_\psi(x, y)$ is defined as follows:

$$B_\psi(x, y) = \psi(x) - \psi(y) - \langle \nabla\psi(y), x - y \rangle$$

where $\psi : \Omega \to \mathbb{R}$ is a function that is a) strictly convex, b) continuously differentiable, c) defined on a closed convex set Ω.

Algorithm 1 Online Mirror Descent using Bregman Projection update

```
1: procedure OMD($V, l, \psi, \eta_0$)
2:     for t do=1 to T
3:         Receive $x_t \in V$
4:         Receive $l_t$ and pay $l_t(x_t)$
5:         Set $g_t \in \partial l_t(x_t)$
6:         Gradient descent step: $x_{t+1} = x_t - \eta_t g_t$
7:         Bregman Projection step: $\Pi_V(x_{t+1}) = \arg\min_{y \in V} B_\psi(y, x_{t+1})$
8:         Update $\eta_t$
9:     end for
10: end procedure
```

Algorithm 2 Online Mirror Descent using Bregman Divergence as regularizer

```
1: procedure OMD($\psi, V, l, \eta_0$)
2:     for t do=1 to T
3:         Receive $x_t \in V$
4:         Receive $l_t$ and pay $l_t(x_t)$
5:         Set $g_t \in \partial l_t(x_t)$
6:         Update using Bregman Divergence as regularizer:
           $x_{t+1} = argmin_{x \in V} \langle g_t, x \rangle + B_\psi(x, x_t)/\eta_t$
7:         Update $\eta_t$
8:     end for
9: end procedure
```

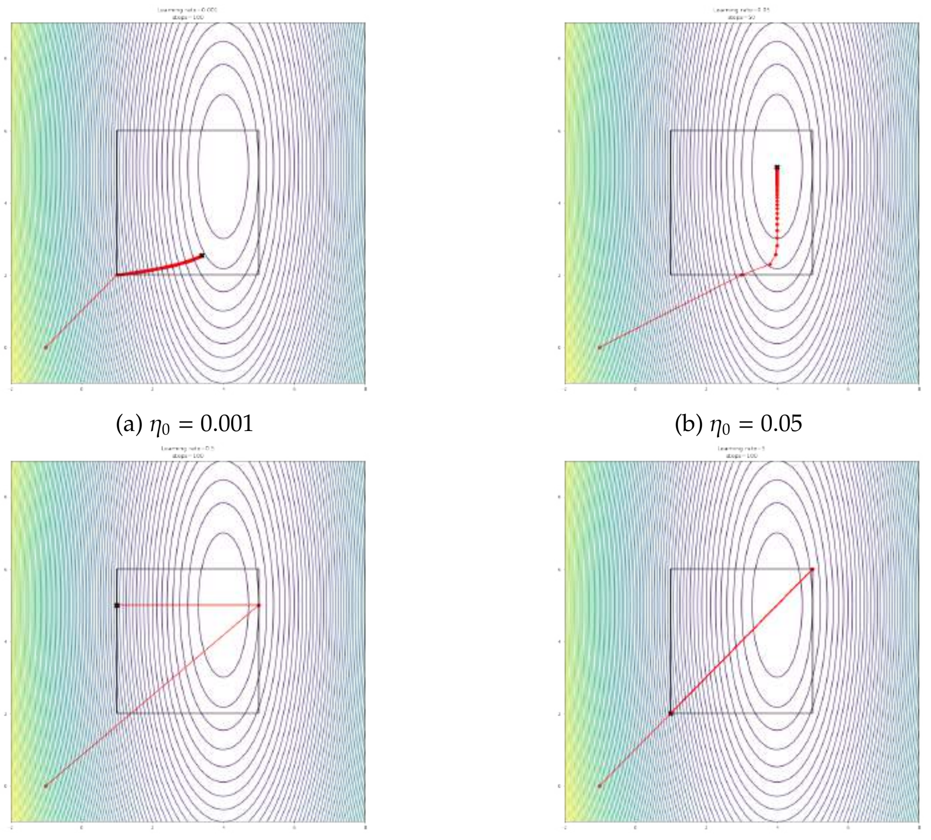
Thus Bregman divergence turns out to be the difference between the value of ψ at x and the first order Taylor expansion of ψ around y evaluated at point x.

The difference in these two algorithms is that, in the first one we perform two optimization problems: one is unconstrained and the other is constrained, while in the second one, only one constrained optimization problem is to be solved in each iteration. We shall see in our next simulation and analysis, that both work perfectly for the right hyperparameters.

2.1 Exploring Algorithm 1

In this section we observe and analyse the simulation results of Algorithm 1. We observe different cases as below:

- Check behaviour for different learning rates
- Check behaviour for different starting points



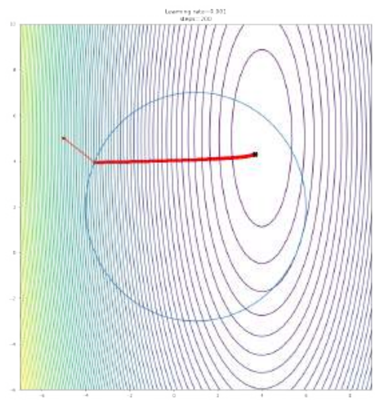
(a) $\eta_0 = 0.001$ (b) $\eta_0 = 0.05$

(c) $\eta_0 = 0.5$ (d) $\eta_0 = 5$

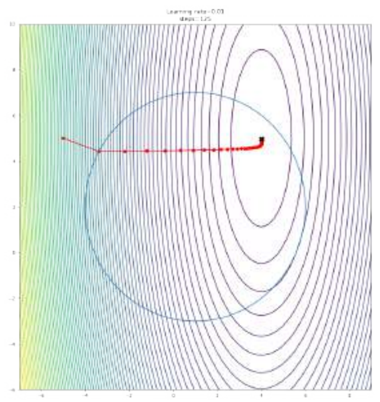
Fig. 1: Optimization trajectory for 4 different learning rates. Bregman function used is $\psi(x_1, x_2) = x_1^2 + x_2^2$. Here the solution lies inside the action set (Linear box action set).
Contour: $l(x_1, x_2) = 8(x_1 - 4)^2 + (x_2 - 5)^2$

In the figures above we see for four different scenarios which can be seen as linear box action sets and circular action sets.

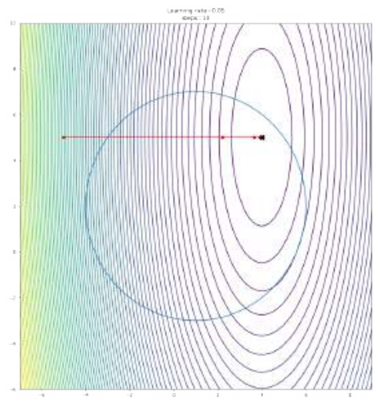
Variation of learning rate: For very low learning rates, reaching the desired point is difficult as per our observation. For very high rates, we see oscillating nature of the trajectory, which doesn't ensure finding the optimum. There's always a learning rate which takes us to the optimum in a stable manner. So, selection of learning rate is one of the most vital points here.



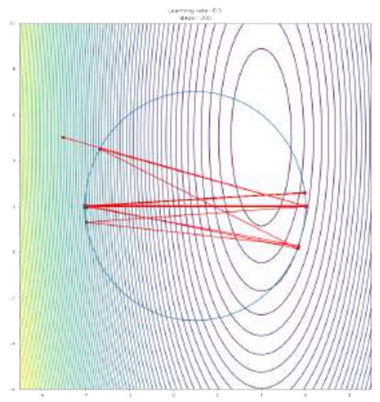
(a) $\eta_0 = 0.001$



(b) $\eta_0 = 0.01$



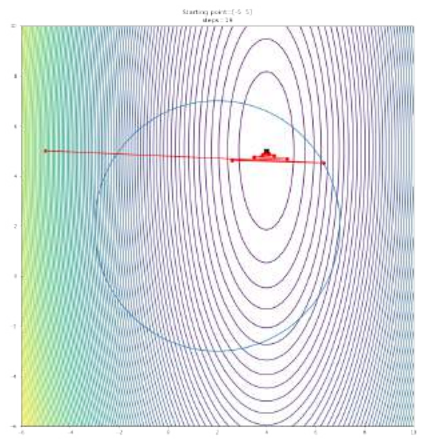
(c) $\eta_0 = 0.05$



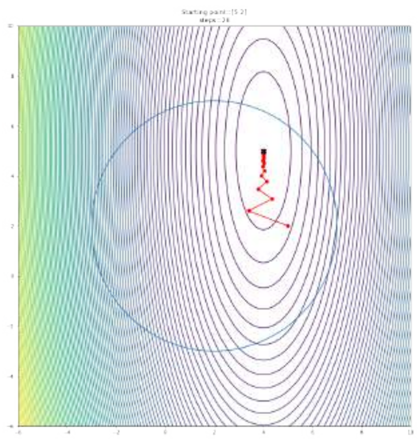
(d) $\eta_0 = 0.5$

Fig. 2: Optimization trajectory for 4 different learning rates. Bregman function used is $\psi(x_1, x_2) = x_1^2 + x_2^2$. Here the solution lies inside the action set (Circular action set).
Contour: $l(x_1, x_2) = 8(x_1 - 4)^2 + (x_2 - 5)^2$

Variation of starting point: In Figure 3 we see two different starting points and how even for a point outside the action set initially, it takes slightly less steps to reach the optimum than a starting point inside the action set. This happens because of the Bregman projection taking place at the first step and bringing the trajectory inside the action set, and thus reducing the difference in number of steps for the two cases.



(a) Starting point=(-5,5)



(b) Starting point=(5,2)

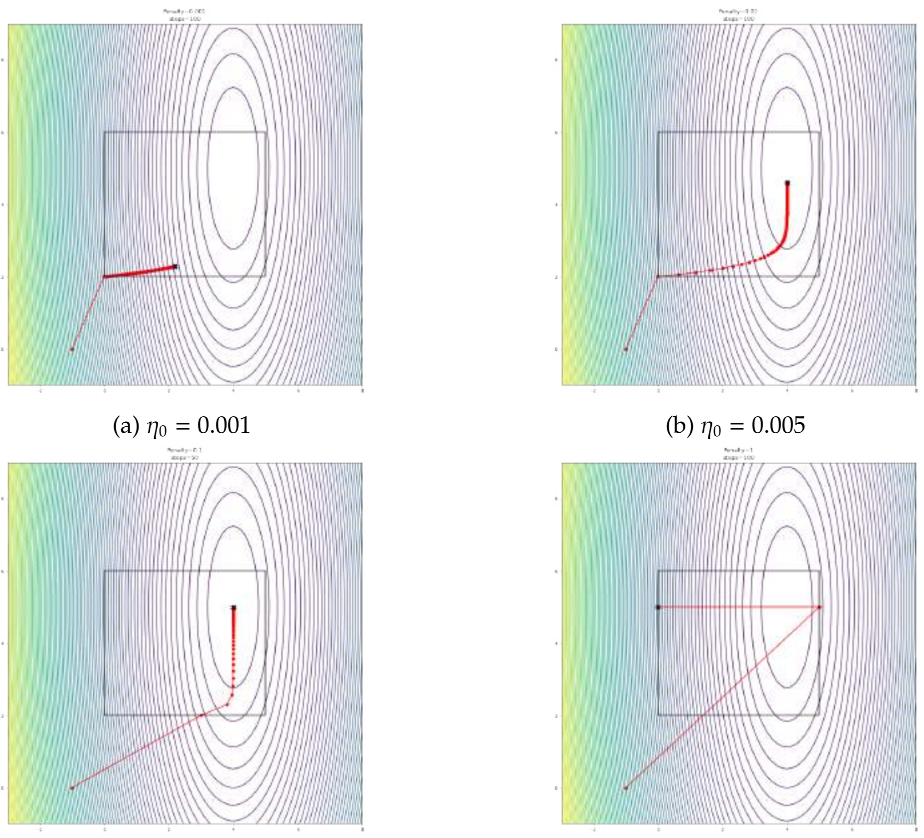
Fig. 3: Optimization trajectory for 2 different starting points. Bregman function used is $\psi(x_1, x_2) = x_1^2 + x_2^2$. Here the solution lies inside the action set (Circular action set).
Contour: $l(x_1, x_2) = 8(x_1 - 4)^2 + (x_2 - 5)^2$

2.2 Exploring Algorithm 2

In algorithm 2, instead of taking a gradient step and then projecting it, we minimize the expression in action set V $\langle g_t, x\rangle + B_\psi(x, x_t)/\eta_t$ to find x_{t+1}. To go along the direction of descent, we could minimize $\langle g_t, x$ only, but in that case the step would take us far from the optimizer or in cases, to the boundary of the action set. The Bregman divergence term here stops the algorithm from such behaviour and holds it back from running far down the slope g_t. The η_t that divides the Bregman divergence term decides how much to amplify the importance of the Bregman divergence. So, in simpler terms, $(1/\eta_0)$ plays the role of a regularization hyperparameter.

Variable penalty (η_0): In figure 4 we observe that lower the value of η_0, higher is the importance of the Bregman divergence term. And this higher importance holds the algorithm back from taking longer steps. If we keep increasing η_0 we see that for very high values, the importance of the Bregman divergence term reduces and the inner product with the gradient dominates. The iner product being an affine term, tends to roll down the slope in an unbounded manner and thus oscillation comes in.

Variation in starting point: An important difference in behaviour is noticed, when we compare Figure 5(a) and Figure 5(b). In the latter case we see that the trajectory has a tendency to stick to the portion of the boundary of the action set which is closer to the optimum. The reason behind such behaviour is the absence of the gradient step that might take the point far from the closer boundary and project on some other part of the action set. This behavior is specifically seen if we start from a point that is closer to the optimum and if the optimum is



(a) $\eta_0 = 0.001$

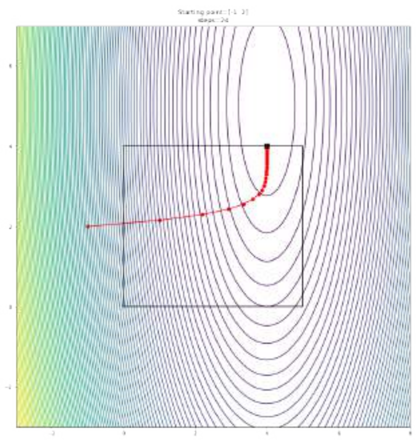
(b) $\eta_0 = 0.005$

(c) $\eta_0 = 0.01$

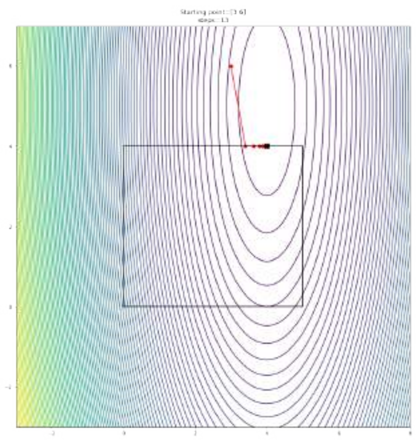
(d) $\eta_0 = 1$

Fig. 4: Optimization trajectory for 4 different penalties (η_0) for the Bregman divergence. Bregman function used is $\psi(x_1, x_2) = x_1^2 + x_2^2$. Here the solution lies inside the action set (Linear box action set).
Contour: $l(x_1, x_2) = 8(x_1 - 4)^2 + (x_2 - 5)^2$

outside the action set. In Figure 8 we present this particular fact.



(a) Starting point=(-1,2)



(b) Starting point=(3,6)

Fig. 5: Optimization trajectory for 2 different starting points. Bregman function used is $\psi(x_1, x_2) = x_1^2 + x_2^2$. Here the solution lies outside the action set (Linear box action set).
Contour: $l(x_1, x_2) = 8(x_1 - 4)^2 + (x_2 - 5)^2$

3 Online Learning with Expert Advice

In this setting, we have d experts that give us some advice on actions in each round. In turn, in each round we have to decide which expert we want to follow. After we made our choice, the losses associated to each expert are revealed and we pay the loss associated to the expert we picked. The aim of the game is to minimize the losses we make compared to cumulative losses of the best expert. This is a general setting that allows to model many interesting cases. For example, we have a number of different online learning algorithms and we would like to close to the best among them.
If we put ourselves in the adversarial setting, even with 2 experts, the adversary can force on us linear regret. This means that the cumulative loss of the algorithm over T rounds is proportional to T. The problem above is due to the fact that the adversary has too much power. One way to reduce its power is using randomization. We can allow the algorithm to be randomized and force the adversary to decide the losses at time t without knowing the outcome of the randomization of the algorithm at time t (but it can depend on the past randomization). This is enough to make the problem solvable. Moreover, it will also make the problem convex, allowing us to use any OCO algorithm on it.

We incorporate the randomization in three ways. The three cases are described below:

1. We simulate the whole game with vectors in probability simplex. We pick up a vector $x_1 \in \{x \in \mathbb{R}^d : x_i \geq 0, \|x\|_1 = 1\}$. Then we select the expert from a set of d e_is where e_i is a d dimensional vector with i^{th} element set to 1 and all other elements zero. This selection is done according to the distribution of x. Here comes the stage of randomization. Our goal is to reduce the expected regret

$$\mathbb{E}[Regret_T(e_i)] = \mathbb{E}\left[\sum_{t=1}^{T}\langle g_t, x_t\rangle - \sum_{t=1}^{T}\langle g_t, e_i\rangle\right]$$

2. Here instead of taking the expert from e_is, we choose them in a totally randomized manner from $e \in \{x \in \mathbb{R}^d : x_i \geq 0, \|x\|_1 = 1\}$.

3. In this case we take the expert from $e \in \{x \in \mathbb{R}^d : x_i \geq 0, \|x\|_1 = 1\}$, but e is close to x_t as we take randomized projections of $(x_t + \epsilon)$ on the probability simplex, where ϵ is a d-dimensional random normal vector.

Algorithm 3 Online Learning with expert advice using randomization (1)

```
1: procedure Expert advice 1($x \in \mathbb{R}^d : x_i \geq 0, ||x||_1 = 1$)
2:     for t do=1 to T
3:         Draw $i_t$ according to $P(i_t = i) = x_{t,i}$          ▷ Randomization step
4:         Select $e_{i,t}$
5:         Pay the loss of the selected $e_{i,t}$
6:         Update $x_t$ with an OCO algorithm (Exponentiated Gradient)
7:     end for
8: end procedure
```

Algorithm 4 Online Learning with expert advice using randomization (2)

```
1: procedure Expert advice 2($V$=$\{x \in \mathbb{R}^d : x_i \geq 0, ||x||_1 = 1\}$)
2:     for t do=1 to T
3:         Draw a random normal vector $a$          ▷ Randomization step
4:         $a'_i = \Pi_{V,i}(a)$
5:         Select expert $e_t$ randomly from $a'_i$s
6:         Pay the loss of the selected $e_t$
7:         Update $x_t$ with an OCO algorithm (Exponentiated Gradient)
8:     end for
9: end procedure
```

Algorithm 5 Online Learning with expert advice using randomization (3)

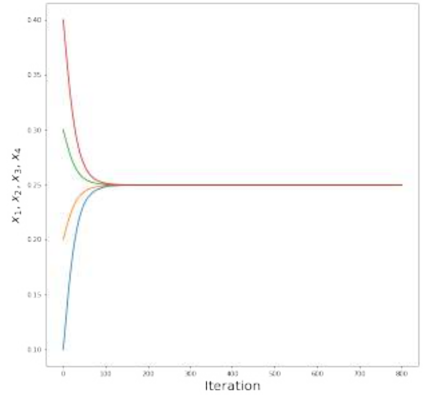
```
1: procedure Expert advice 3($V$=$\{x \in \mathbb{R}^d : x_i \geq 0, ||x||_1 = 1\}$)
2:     for t do=1 to T
3:         $a = x_t + \epsilon$          ▷ Randomization step
4:         Project $a$ on V
5:         Select expert $e_t$ randomly from projections of $a$
6:         Pay the loss of the selected $e_t$
7:         Update $x_t$ with an OCO algorithm (Exponentiated Gradient)
8:     end for
9: end procedure
```

Algorithm 6 Online Learning with expert advice using Bregman Divergence regularizer

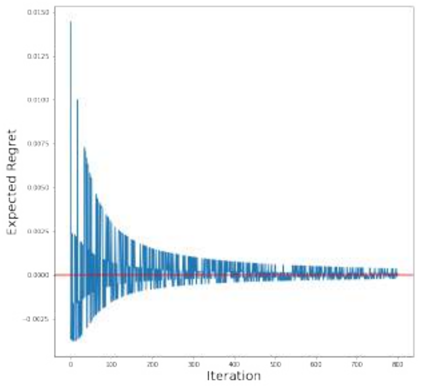
```
1: procedure EXPERT ADVICE 4($V$={$x \in \mathbb{R}^d : x_i \geq 0, \|x\|_1 = 1$})
2:     for t do=1 to T
3:         Draw $i_t$ according to $P(i_t = i) = x_{t,i}$          ▷ Randomization step
4:         Select $e_{i,t}$
5:         Pay the loss of the selected $e_{i,t}$
6:         Update $x_t$ with Bregman divergence regularizer
7:     end for
8: end procedure
```

3.1 Analysis and comparison of the algorithms

In figures 6,7 and 8 we see that all three converge to the minimum at the same time, but our choice of expert determines the nature of expected regret. In Expert Advice 1 and Expert advice 3 we see that the expected regrets grow sublinearly and they converge towards zero with iterations. This happens because in both the cases we are using our knowledge or previous actions to choose our next expert. In case of Expert Advice 3, we choose our expert randomly from the whole action set and thus the regret is very random in nature. This shows that using knowledge from previous experience for predicting the next correct move is better than going in an entirely random fashion.
In figure 9,10,11,12 and 13 we are seeing the results of Online learning with expert advice using Bregman divergence regularizer. We vary the divergence functions and with the change of the functions, we observe changes in rate of convergences and we also observe that the solution paths are different. The most drastic change is seen when we use $\psi(x) = \sum |x_i|$ as the regularizer. On this loss function, $l(x) = \prod x_i$, this regularizer shows fastest convergence but with a solution path which is very different from the other ones.

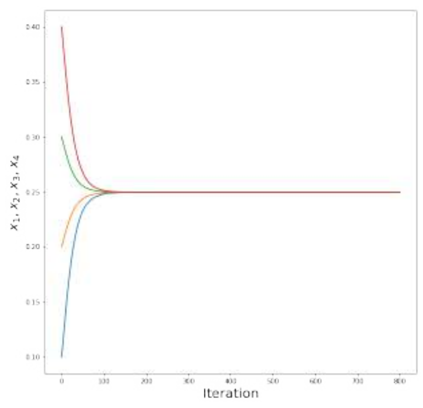


(a) x_t vs. Iteration

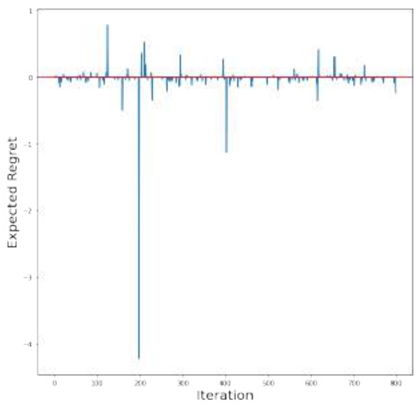


(b) Expected Regret vs. Iteration

Fig. 6: Plots for Expert Advice 1
Loss function: $l(x) = \prod x_i$

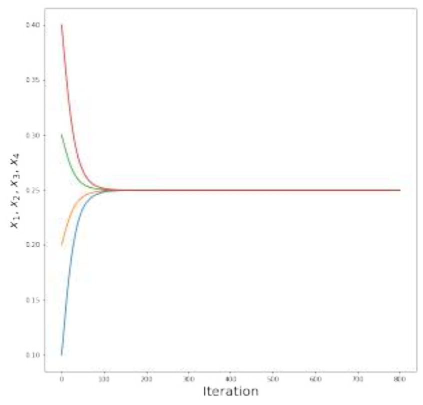


(a) x_t vs. Iteration

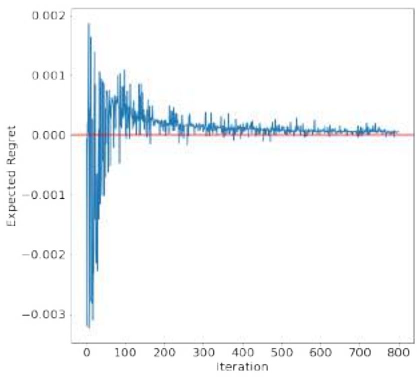


(b) Expected Regret vs. Iteration

Fig. 7: Plots for Expert Advice 2
Loss function: $l(x) = \prod x_i$

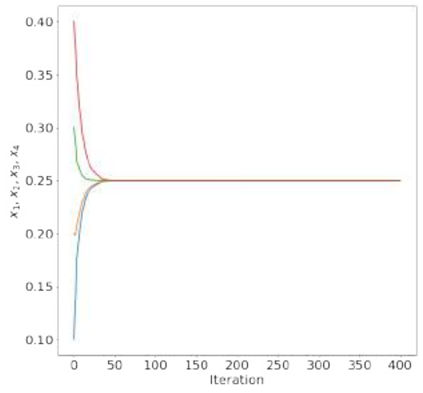


(a) x_t vs. Iteration

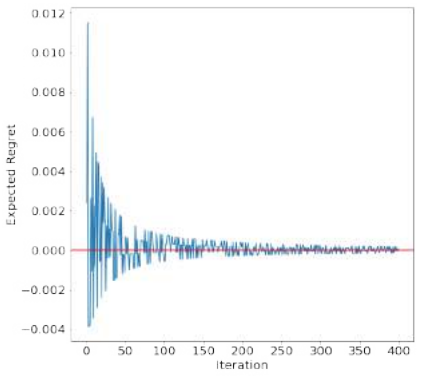


(b) Expected Regret vs. Iteration

Fig. 8: Plots for Expert Advice 3
Loss function: $l(x) = \prod x_i$

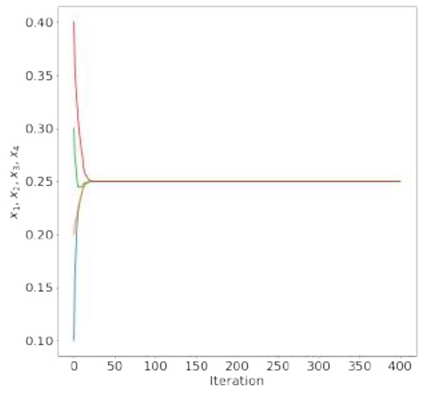


(a) x_t vs. Iteration

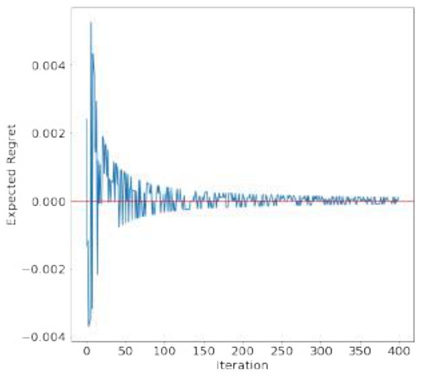


(b) Expected Regret vs. Iteration

Fig. 9: Plots for Expert Advice 4
Loss function: $l(x) = \prod x_i$
Divergence function $\psi(x) = \sum x_i^2$

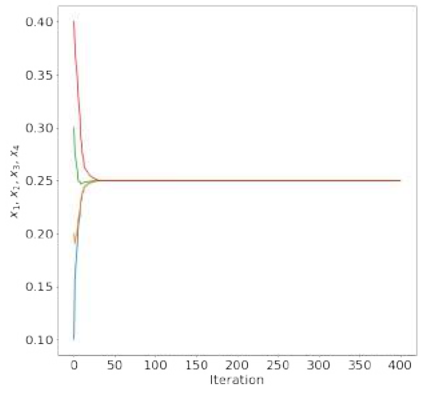


(a) x_t vs. Iteration

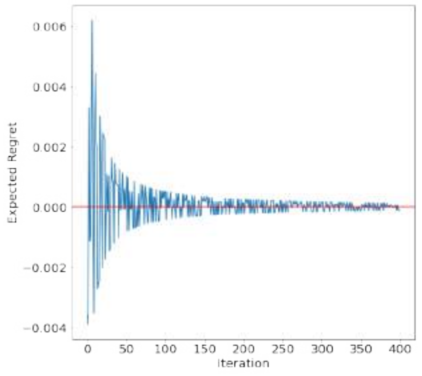


(b) Expected Regret vs. Iteration

Fig. 10: Plots for Expert Advice 4
Loss function: $l(x) = \prod x_i$
Divergence function:$\psi(x) = \sum x_i^4$

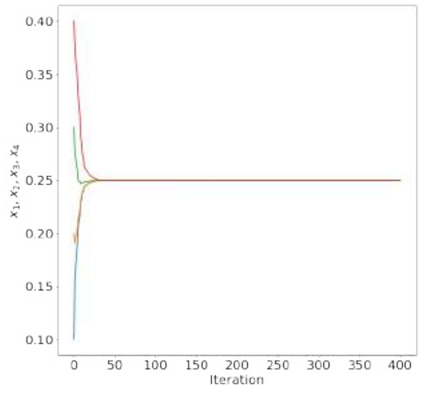


(a) x_t vs. Iteration

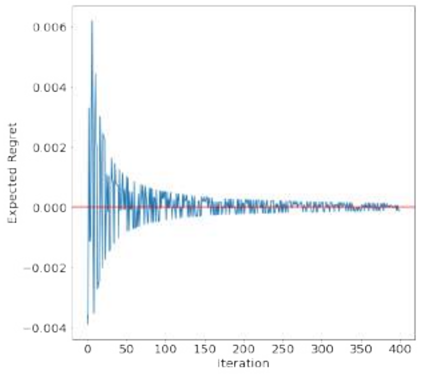


(b) Expected Regret vs. Iteration

Fig. 11: Plots for Expert Advice 4
Loss function: $l(x) = \prod x_i$
Divergence function:$\psi(x) = \sum x_i^3$

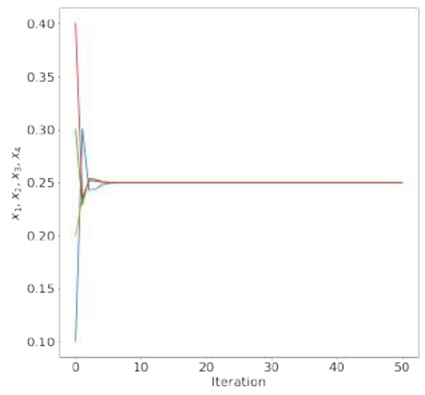


(a) x_t vs. Iteration

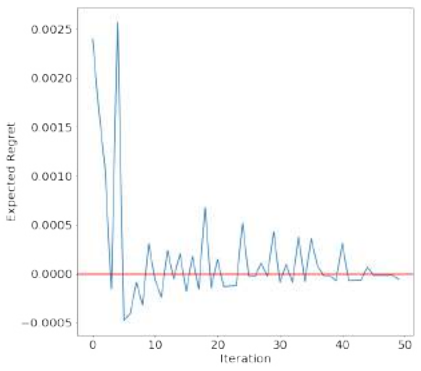


(b) Expected Regret vs. Iteration

Fig. 12: Plots for Expert Advice 4
Loss function: $l(x) = \prod x_i$
Divergence function:$\psi(x) = \sum x_i^3$



(a) x_t vs. Iteration



(b) Expected Regret vs. Iteration

Fig. 13: Plots for Expert Advice 4
Loss function: $l(x) = \prod x_i$
Divergence function:$\psi(x) = \sum |x_i|$

The Exponentiated Gradient algorithm for online convex optimization

Algorithm 7 Exponentiated Gradient Descent

```
1: procedure EXPONENTIATED GRADIENT DESCENT($x \in \mathbb{R}^d : x_i \geq 0, \|x\|_1 = 1, \eta > 0$)
2:     for t do=1 to T
3:         Output $x_t$
4:         Receive $l_t$ and pay $l_t(x_t)$
5:         Pay the loss of the selected $e_{i,t}$
6:         set $g_t \in \partial l_t(x_t)$
7:         $x_{t+1,j} = \frac{x_{t,j} exp(-\eta g_{t,j})}{\sum_{i=1}^{d} x_{t,i} exp(-\eta g_{t,i})}$
8:     end for
9: end procedure
```

4 Conclusion

Our observation of the Online Mirror Descent using Bregman divergence and Online Learning with Expert Advice reveals one important fact that we had mentioned earlier: In online learning paradigm, observing the regret and minimizing it is more crucial than dealing solely with the loss function(s). In both OMD and Expert Advice, as we do not have batch data points in one go, we tend to utilize the last decision and its regret for the next decision. Processing with only one data point speeds up the computation but takes longer to reach the right decision strategy.