Online Convex Opitmization Project

Online gradients methods for training linear SVM on MNIST dataset.

Chengcheng GU
Qiaoyu LUO
Yunfei ZHAO

January 2, 2022

In this project we aim at implementing online gradients methods we have learnt for training linear SVM on MNIST dataset and analysing their performance.

Contents

List of Figures

1 Preliminary

1.1 Introduction

The aim of the project is to comparison the different online gradients methods by implement them for training linear SVM on MNIST dataset. The performances of the algorithms, all trained on the same train set, will be quantified using their accuracy scores on the test set.

1.2 MNIST Dataset

MNIST
MNIST is a hand writing digital number dataset that possesses 10 numbers from 0 to 9. **Sample ratio for each class**

Number of samples in data sets											
Type	0	1	2	3	4	5	6	7	8	9	Total
Train	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949	60000
Test	980	1135	1032	1010	982	892	958	1028	974	1009	10000

Classification
Our mission here is to do a binary classification between 0 and other numbers by a SVM model where the parameters optimisation process is achieved by our online optimisation process.

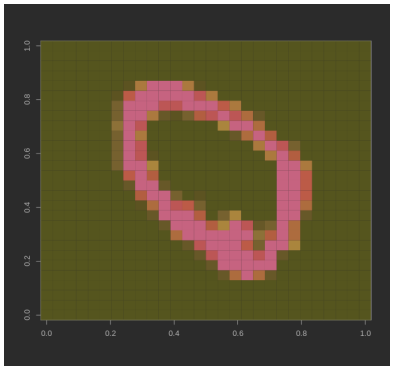

Figure 1: An example of hand written number 0

1.3 Notion essential

All the online convex optimisation algorithm based on:

- The objective is minimization the

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

  at least sub sub-linear regrets $O(T)$

- The function is considered as convex Liptchitzian.

- The accuracy to measure the performance of each algorithm is $h_t = f(x_t) - f(x^*)$ in theory, that is difference between the loss function in t iteration and the optimized loss function. In practice, we take 1 - the rate of the correct classification in the algorithm.

1.4 Cost function

We consider a cost function for the SVM contain two part, the first one is the Hinge lost of the labels that we predict with our parameter between the true labels for the observation (supervisor), and the second is a L2 penalization regularization term adding to avoid over fitting, we add a L2 norm of the parameter so that it doesn't go too big. So the goal of our algorithm is to find the parameter that reach the minimize of the cost function. The cost function finally is as follow :

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} hinge(bx^{\top}a_i) + \frac{\lambda}{2}\|x\|^2$$

.

2 Gradient Descent

2.1 Unconstrained Gradient Descent

Unconstrained Gradient descent is the simplest algorithm, we compute the gradient of our lost function at each iteration for the whole training data set and so that it gives us a direction to decrease the cost which is minus gradient of f, and we multiply by a learning rate who would change at each iteration (when the iteration t increase, the learning rate decrease, which make sense because we start with a large speed but when we are going close to the minimize point that we want, we should slow down the speed to avoid that we might go through the minimize point.), for the L2 regularization term we have λx as it's gradient.

Notice that in the Gradient descent algorithm, we take the mean value of the gradient of the whole data set (observations) for every direction, which takes time to do the computation because we compute all 60000 observations at each step(generally it takes around one minute for a training). The idea of "online" comes from that we set the learning rate $\eta_t = 1/\lambda t$ which changes every iteration step, and a theorem calls that if we have a constant λ that ensures the cost function f_t are λ-strongly convex, then we have an optimal regret bound $O(logT)$ by this learning rate.

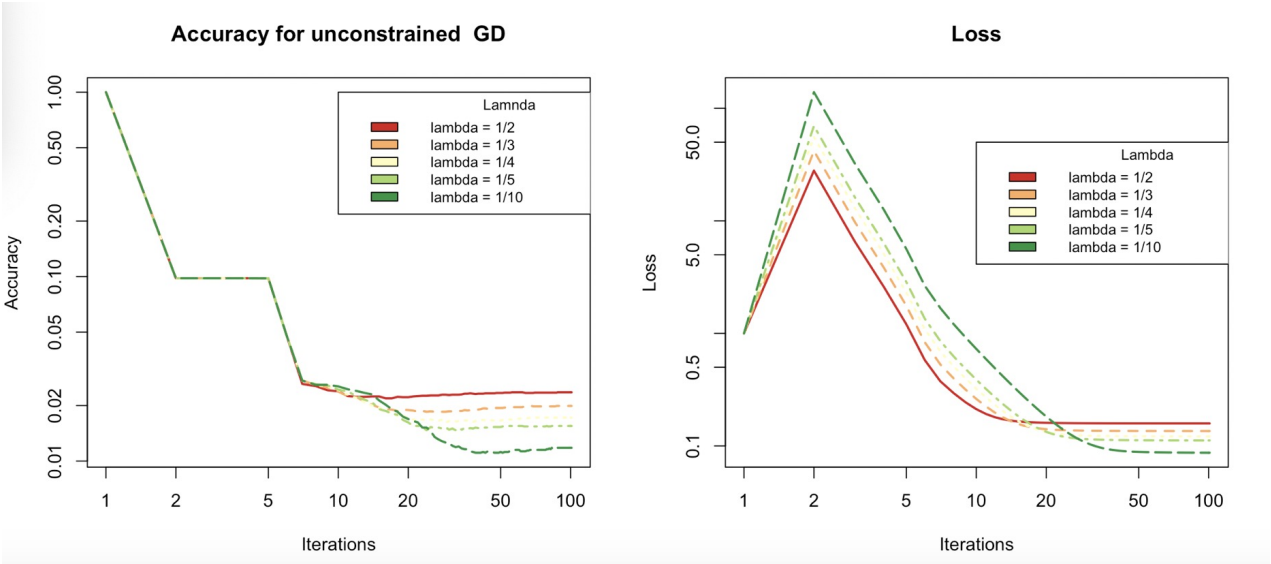

Figure 2: Accuracy and Loss for Un-GD

The left image shows us the accuracy of the parameters we predicted on the testing data set, and the right image is the curves of the cost that we saved when training.(ie. All the images in this chapter are the same arrangement.)

We can see in the image that when the hyper-parameterλ decrease, we have more accuracy(the curve decrease means the convergence is more strictly) and the loss is also decrease. We can consider it as, when λ is big, we pay more attention that our parameter don't go too big in the sense of L2 regularization. But in the right image we can see that with smaller λ, we need more iterations to reach the minimize. And in conclusion to have the best performance of the accuracy, 1/10 is a nice choice of the hyper parameter λ.

2.2 Projected Gradient Descent

For the projection version of gradient descent, we add a step at each iteration that project the point(parameter) we get by the previous step into a space that we want,so that we restrict the parameters in this space, where the space is a L1 ball in this question.

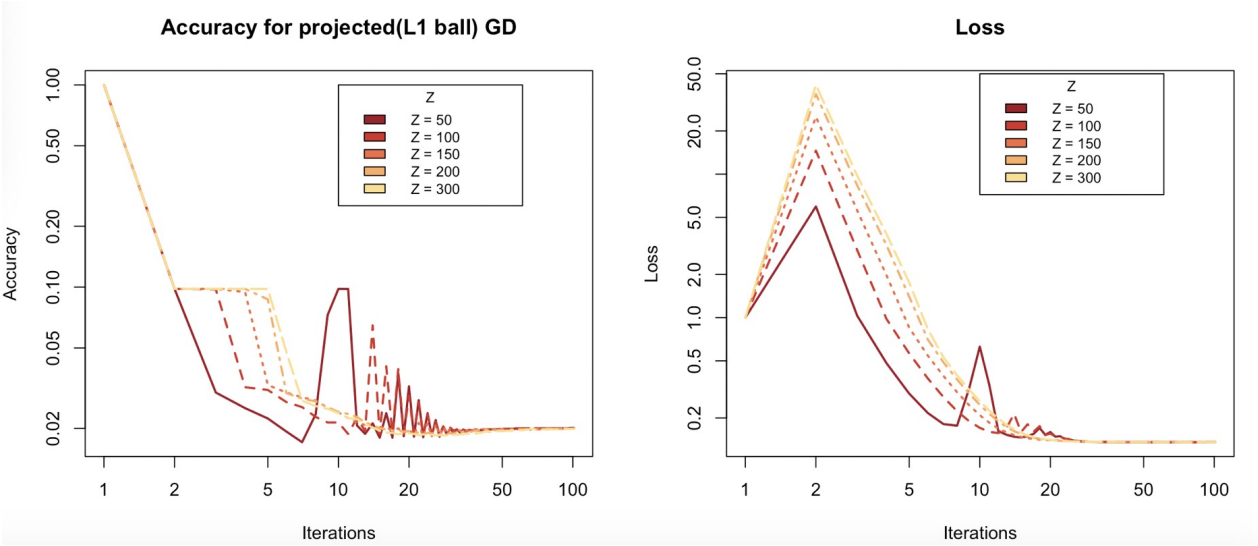

Figure 3: Accuracy and Loss for Projected-GD

The images show that at the end of the convergence, we get the same results when we take the diameter of the L1 simplex as 50,100,150,200,300. But if we look at the curves, it's clear that there is a strong fluctuation when the diameter is small(z=50,100 in our example), that means if we stop training before the algorithm reach the minimize, the results might be absurd if we project the parameter in a simplex with small diameter.
Finally in this chapter, we fix the hyper-parameter $\lambda = \frac{1}{3}$ and the diameter of the L1 ball $Z = 100$ then we set the iteration number T to 1000 to see the performance for this algorithm, we find out that after 50 iteration, we are succeed to reach the minimize for this algorithm.

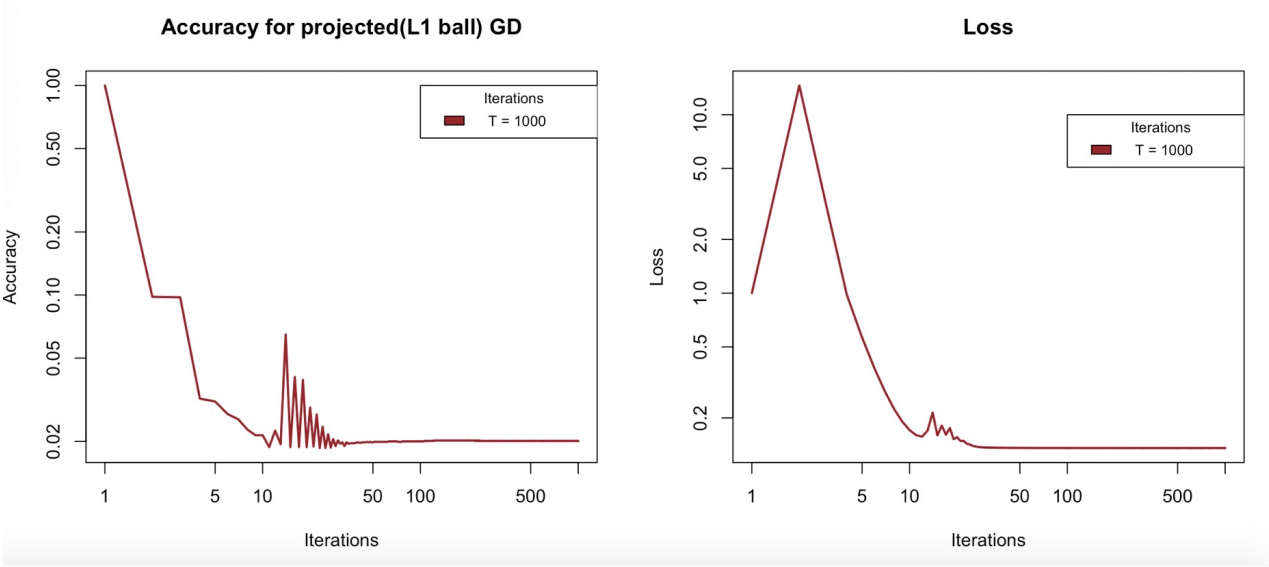

Figure 4: Accuracy and Loss for 1000 iteration

3 Stochastic Gradient Descent

The main idea of stochastic gradient descent is that at each step, we just compute the gradient of the cost function for a little batch of samples of the whole data set, we prove that the expectation of the gradient of this little batch is equals to the expectation of the whole data set, that's the reason why we can do the descent step with the gradient of the little batch of samples. To do so, the method is that we take the data randomly for training. The advantage of this algorithm is that for each iteration we don't have to compute all the gradient but a little of them, and so it takes less time to compute. Here we take randomly uniformly one observation in the 60000 observations. In the code for this method, we have a different function of the gradient which at each iteration, the function takes only one observation to compute the gradient. Since the cost function depends on the variable of the observation, we now have different cost function F_t at each iteration t, which fit the idea of "online" that the cost function shall change with t. We have here

$$\mathbf{E}[\widehat{\nabla f(x)}] = \sum_{i=1}^{n} \nabla l_i(x)\mathbf{P}(I=i) = \frac{1}{n}\sum_{i=1}^{n} \nabla l_i(x) = \nabla f(x) \tag{1}$$

where we sample uniformly $I \in 1, \cdots, n$ and use $\widehat{\nabla f} = \nabla f(x)$

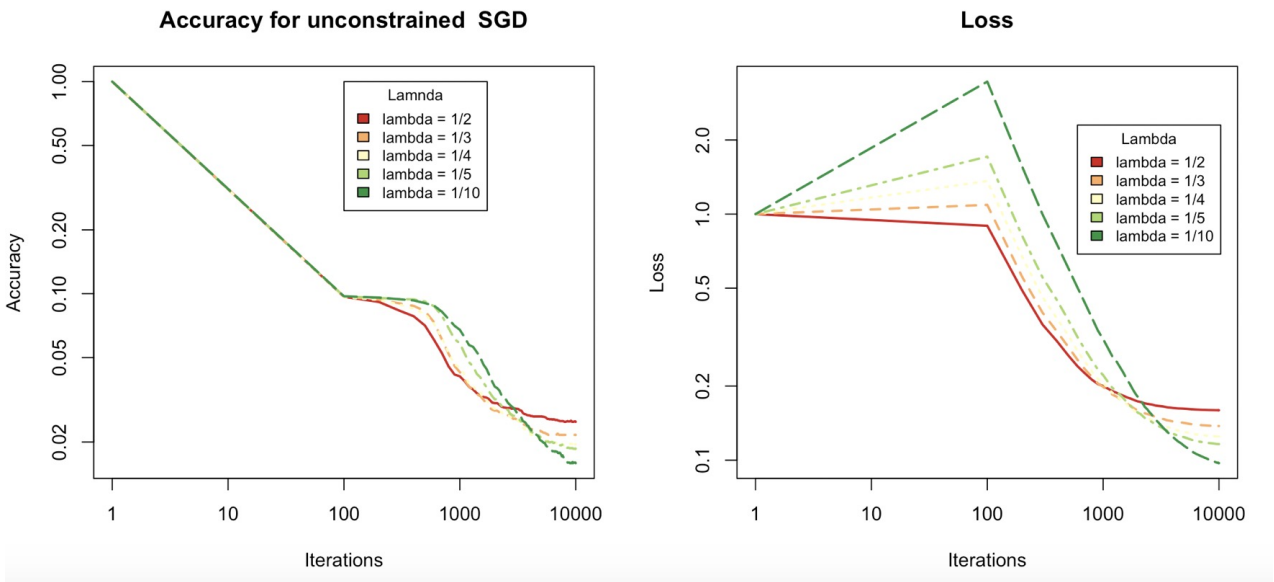

Figure 5: Accuracy and Loss for unconstrained SGD

This is the image for unconstrained SGD with different λ. From the image we obtain the same conclusion as we did for Gradient descent when changing the value of λ, which is that $\lambda = 1/10$ gives out the best accuracy but with a little bit more iterations needed.

We are now interested in the performance of the projected stochastic gradient descent compare with the version without stochastic. Notice that in fact the number of iterations are different between the two algorithm, there are 10000 iterations for the projected stochastic gradient descent and 100 iterations for projected gradient descent. But we save the parameters of the projected SGD once every 100 times of iterations, so that finally we have 100 data to show. The axis of iteration is not rigorous but only and idea of the increase in proportion of the iteration.

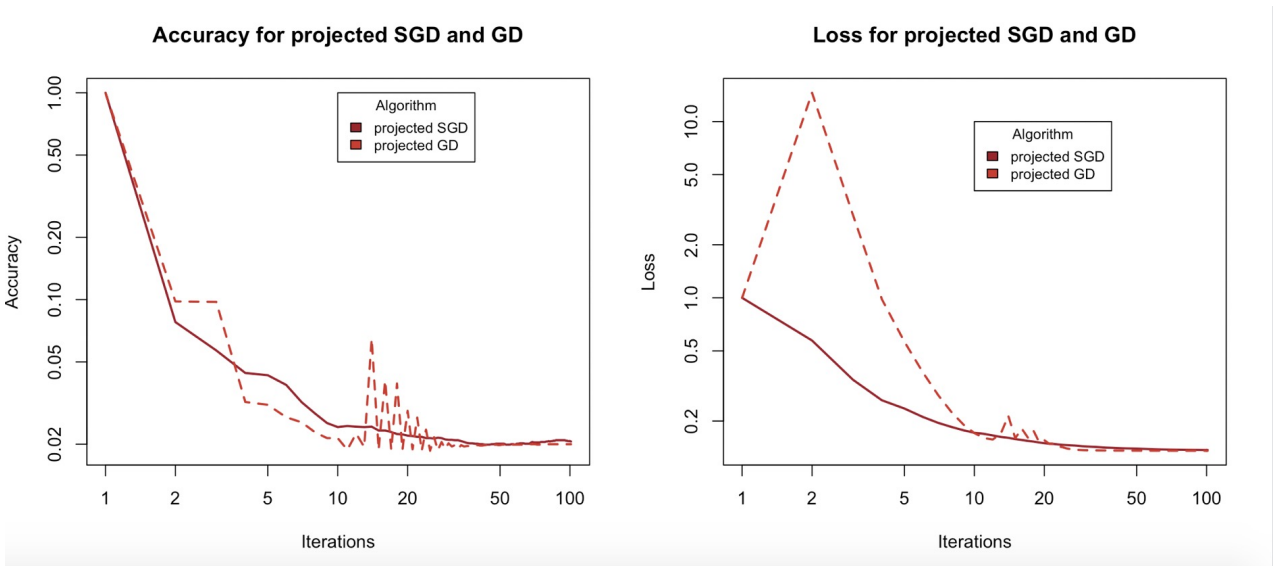

Figure 6: Compare projected SGD ans GD

We know that when the iteration equals to 10000, we only use 10000 observations (1/6 data) for training, but through what shows by the image, the accuracy and Loss for the projected stochastic gradient descent are as good as those for the projected gradient descent, where we train the model with our whole data set for 100 times. It is clear that for each step of the gradient descent, the complexity of computing the gradient is $O(nd)$, on the contrary, the complexity of the stochastic gradient descent is $\emptyset(d)$, where n is the number of our observations in training set(60000 here) and d is the number of variables(784 here). Obviously, training with stochastic gradient descent is much faster then gradient descent, on my computer, it takes around one minutes(running time) to train with gradient descent algorithm for 100 iterations and reach the accuracy show by the image(minimize), but it only takes around 2.2 second(running time) for stochastic gradient descent to reach the same performance. So of cause it is better to use projected stochastic gradient descent in such problem.

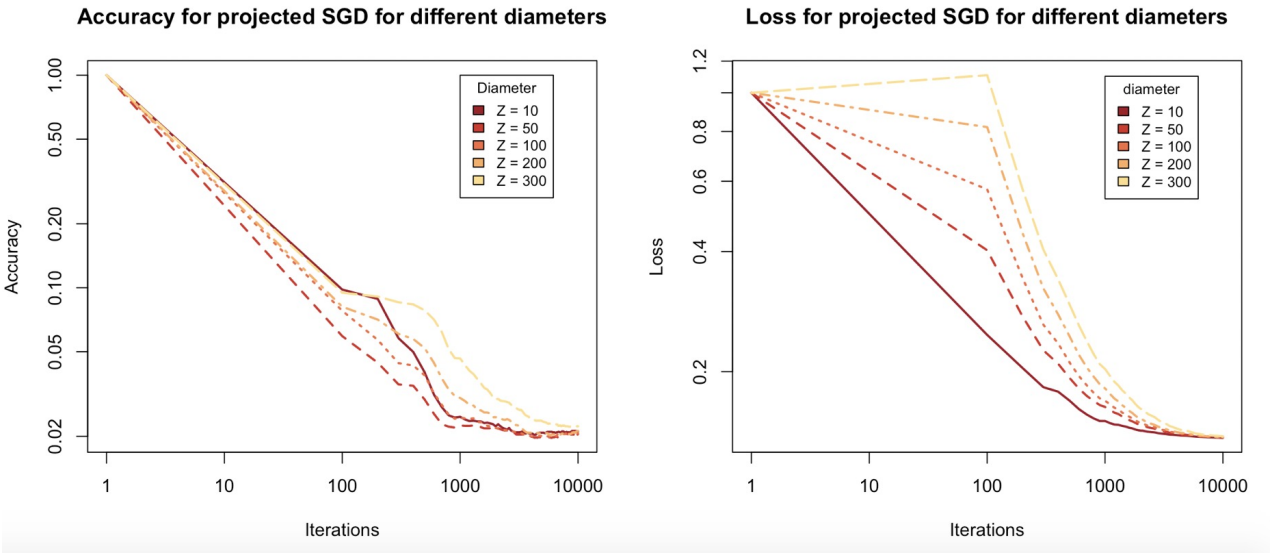

Figure 7: different diameters Z

Let's have a look of the performance of projected stochastic gradient descent with different diameters Z of the L1 ball. We can see that at the end all the 5 algorithms could reach the minimize but with the speed a little bit different.

4 Regularized Follow The Leader

The FTL (follow the leader) is the basic strategy which was to take the best over the past, that is in formulas:

$$x_{t+1} = x_t^* \in arg\min_{x \in \mathcal{K}} \sum_{k=1}^{t} f_k(x) \tag{2}$$

However, this strategy doesn't works. In the above of this report, we have taken the alternative strategy OGD (Online Gradient Descent) and SGD (Stochastic Gradient Descent) that guarantees sublinear regret for convex lipschitz functions. Beginning of this section, we will take

another alternative strategy RFTL (Regularized Follow The Leader) which reserve FTL but add a regularization term at each step to predict our parameter x_t, the explicit formula is as follow:

$$x_{t+1} = x_t^* = \arg\min_{x \in \mathcal{K}} \sum_{k=1}^{t} \nabla f_k(x_k)^\top x + \frac{1}{\eta} R(x) \tag{3}$$

where R is a strongly convex regularization function twice continuously differentiable.

In the following of this section, we will apply OMD (Online Mirror Descent),SMD (Stochastic Exponentiated Gradient +/-), SAG (Stochastic AdaGrad) on Mnist for SVM, and comparing the performance with projected SGD(projected Stochastic Gradient Descent).

4.1 Online Mirror Descent

Online Mirror Descent is an Online Gradient Descent for RFTL strategy with a regularization convex function R defined in its dual space $\mathcal{K}^* = \{\Delta R(x)^T; x \in \mathcal{K}\}$, which has a convex conjugate $R^*(x) = \max_{x \in \mathcal{K}} y^T x - R(y)\}$ equivalent

$$\nabla R^*(x) = \arg\min_{y \in \mathcal{K}} \{y^T x - R(y)\} \tag{4}$$

The main idea of OMD is insteading of taken directly the primal space $\mathcal{K}$, the online gradient descent is done in the dual space $\mathcal{K}^*$, then the projection back to the primal space $\mathcal{K}$ is driven by the Bergman divergence of R; that is, with initialisation x_1,y_1 such that $\nabla R(y_1) = 0$, for each recursion, the convex lipchitz loss function $f_t(x_t)$ is obtained by predicted x_t, then doing the online gradient descent on $\nabla R(y_{t+1})$,where $\nabla R(y_{t+1}) = \nabla R(y_t) - \eta \nabla f_t(x_t)$. Then using the propriety of Bergman divergence, we take $x_{t+1} = \arg\min_{x \in \mathcal{K}} B_R(x||y_{t+1}) = \arg\min_{x \in \mathcal{K}} \{R(y) - R(x) - \nabla R(x)^T(y - x)\}$

Comparing with Projected SGD, there are some differences in theory. First, OMD construct another space named "dual space" to do the gradient descent, but the Projected SGD takes the gradient descent directly in the space original. Second, the projection of each updated point is from the point in dual space to primal space, and the gradient descent is not doing for projected point, but only for each point in the dual space. However, the Projected SGD takes the projection for each iteration and updates the gradient descent based on the projected point. Third, the projection of SMD is based on the minimisation Bergman divergence of regularization R rather than usual euclidean norm for projected SGD.

4.2 Stochastic Mirror Descent

Our implementation SMD (Stochastic Mirror Descent) is based on a specific OMD with the quadratic regularization function

$$R(x) = \frac{1}{2}||x - x_1||^2, \ \ \forall x_1 \in \mathcal{K} \tag{5}$$

So the $B_R(x||y) = \frac{1}{2}||x - y||^2$, and

$$x_{t+1} = \arg\min_{x \in \mathcal{K}} B_R(x||y_{t+1}) = \Pi_{\mathcal{K}}(y_{t+1})$$

,
Since R is twice continuously differential, so y_{t+1} is obtained by $\nabla R(y_{t+1}) = 0$, that is $y_{t+1} = y_t - \eta_t(x_t)$, and $y_1 = x_1$ for initialisation.

The implemented SMD and Projected SGD has both taken randomly 10000 sampling with replacement among the 60000 observations, the reason "Stochastic" we have mentioned above, that is the expectation of the gradient of this little batch is equals to the expectation of the whole dataset. Without the loss generation,we take the same iteration T=10000, the same radius $z = 100$, and the same seed to initialisation.

The learning rate η for SMD is fixed in theory, but in the implementation we take the $\eta_t = \frac{1}{\sqrt{i}}$ changed for each iteration i, that is because OMD is equivalent to OGD with a small step size $\eta_t < \eta$ only when $\mathcal{K} = B_2(x)$ for $x > 0$. Moreover, the learning rates for implemented SMD $\eta_t = \frac{1}{\sqrt{t}}$ and projected SGD $\eta_t = \frac{1}{\lambda_t}$ with $\lambda = \frac{1}{3}$ for each iteration i are different, that is the Projected SGD has the slower learning rate.

The optimal regret bound for projected SGD is $O(1/\sqrt{T})$,while for Stochastic Mirror Descent is $\sqrt{T}$, that is Projected SGD has faster the rate of convergence in theory. Regarding the accuracy for Projected SGD and SMD in the left Fig. 8, we can see that the projected SGD has a better accuracy than SMD in the beginning of iterations, that is correspondent with the theory, but in the end, SMD has the better accuracy than projected SGD. In another hand, the loss rate per iteration in the right Fig. 8 is the same correspondent performance: at the beginning, the loss rate for projected SGD is lower than SMD. $O(\sqrt{T})$ for SMD and $O(1/T)$ for Projected SGD, however the loss rate for SMD is lower than projected SGD.

Because the OMD algorithms construct the dual space by the regularization term to do the gradient descent,and projection into the primal space, so the cost of each iteration for OMD algorithms are more expensive. But due to the projection in SVM is explicit, the cost is only require a quite little time to get. On my computer, with the same seed, same iteration 10000, the Projected SGD takes around 4.298 secs, which is 1.369 secs faster than 5.667 secs for SMD (Tab.1)

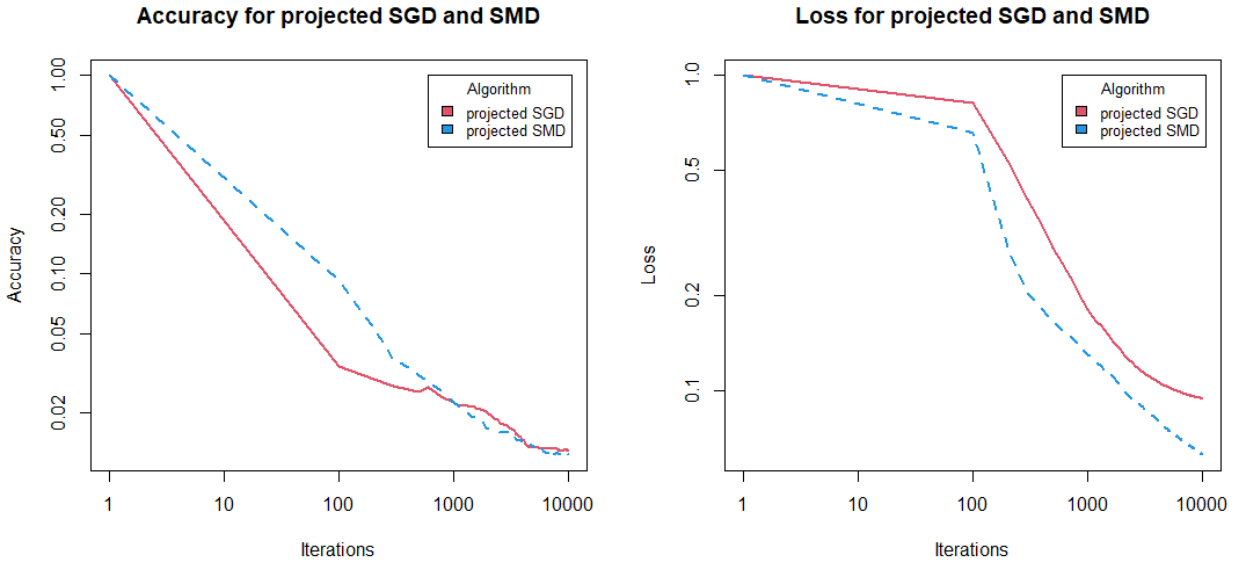

Figure 8: Comparison SGD and SMD

4.3 Stochastic Exponentiated Gradient +/-

Another specific Online Mirror descent is Exponentiated Gradient. In this algorithm, the regularization function is the negative entropy function.

$$R(x) = x^\top log(x) = \sum_{i=1}^{d} x_i log(x_i), \quad \forall x \in \mathbb{R}^d \tag{6}$$

The solution for the dual problem Eq. 3 with the constrained condition Eq. 6 and $x_t > 0$ is

$$x_{t+1} = \frac{exp(y_{t+1})}{\sum_{i=1}^{d} exp(y_{t+1})} \tag{7}$$

In the setting of Learning with Expert Advice, the advice is from d experts with the assignment of confidences weight $x_{t,i}$ on each round. But the minimization the regret cannot be solved because of the adversary has too much power. The way to reduce this power is using randomized strategy that picks randomly an expert i_t with probability $x_{t,i}$ because the randomization makes this problem convex with constrained space $L1$-ball.

The main idea of Exponentiated Gradient +/- is splitting the each coordinate x_i of $x \in \mathbb{R}^d$ into two partitions, $x_{i+} = max(x_i, 0)$ and $x_{x_{i-}} = max(-x_i, 0)$. so $x_i = x_{i+} - x_{i-}$. This algorithm introduces an variable $\theta \in \mathbb{R}^{2d}$ in the dual space L^1-ball $\mathcal{K} = B_1(z)$ of radius $z > 0$ to do the gradient descent, then in each iteration, θ_{t+1} is projected to the point w_{t+1} primal space by Eq. 7. Finally update $x_{t+1,i}$ by multiplier z with $w_{t+1,i} - w_{t+1,i+d}$. Notice that the projection didn't participate the gradient descent in $\mathbb{L}_1$-ball, the gradient descent is only done for the θ in dual space. It is the same as the General Online Mirror Gradient, that is why Exponentiated Gradient +/- is a specific OMD.

Our implementation is the Stochastic version of Exponentiated Gradient +/− on MNIST with radius of the L_1-ball $z = 100$. In order to expectation of the gradient of this little batch is equal to the expectation of the whole dataset,the implemented SED also randomly 10000 sampling with replacement as Projected SGD. With the same iterations 10000, we obtained the results as following Fig. 9.

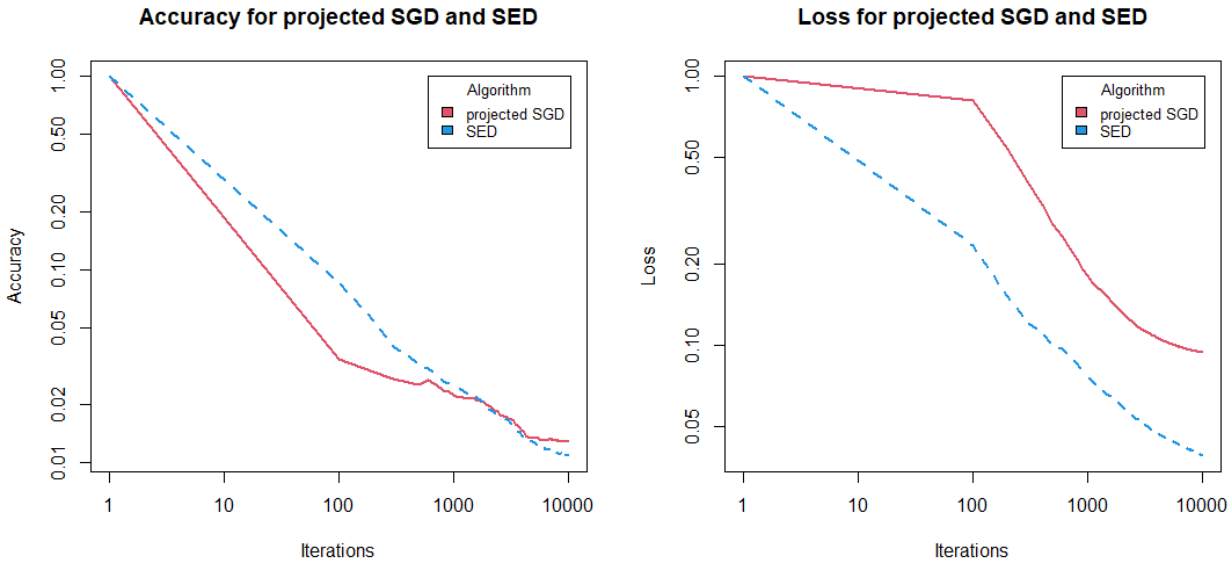

Figure 9: Comparison Projected SGD and SED

Thanks to the regret bound for SEG+/- is $G_\infty z\sqrt{log(2d)/T}$, the optimal regret bound is $O(\sqrt{1/T})$. So the projected SGD has the faster convergence than SED +/- because the optimal regret of projected SGD is . From the left figure in Fig. 9, we can observe that the projected SGD has a better accuracy than SED +/- in the beginning of iterations,which is the same as theory. However even if SED +/- has the better Projected SGD, the accuracy of SED +/- is better than projected SGD.

The loss rate per iteration in the right Fig. 9 is the same correspondent performance: at the beginning, the loss rate for projected SGD is lower than SED +/-. However the loss rate for SED+/- is lower than projected SGD. Moreover, the loss dropped around 0.1 and then stable around this rate for a long time till the end of iterations, but at the same time, the loss for SED+/- continue to dropping till the end of 10000 iterations.

The training time on my computer, with the same seed, same iteration 10000, the Projected SGD takes around 4.157 secs, which is 0.753 secs faster than 4.910 secs for SED +/- (Tab.1). Cost of this algorithm is a little expensive than Projected SGD because the projection on the dual space is explicit in SVM, that is the projection didn't need to take so much cost to approximation.

4.4 Stochastic AdaGrad

AdaGrad is a specific OMD with restrict R to the class of weighted quadratic regularization function $R \in \mathcal{H}$ satisfying

$$\forall x \in \mathcal{K},\ \nabla^2 R(x) = D = Diag(s),\ \ s \in (0,\infty)^d, ||s||_1 \leq 1. \tag{8}$$

Notice that the specific OMD with the quadratic regularization function Eq. 5 is in H, the negative entropy regularization function Eq. 6 is not in H. In fact AdaGrad is more general version of quadratic regularization named adaptive regularization functions:

$$R_t(x) = \frac{1}{2}||x - x_1||^2_{D_t} \tag{9}$$

so that Bergman divergence associated is:

$$B_{R_t}(x||y) = \frac{1}{2}||x - y||^2_{D_t} \tag{10}$$

Different than the other two specific OMD algorithm presented above, this algorithm will be adaptive to characteristics of the sequence of loss functions: the multiple adaptive learning rates. For each iteration, each coordinate of this algorithm have its own gradient step determining by diagonal of root of sum of square gradient,then normalized the gradient,finally taken the projection on the basic adaptive gradient descent. Comparing the Projected SGD, Adagrad learning the sparsity via the gradient, but Projected SGD fixed the radius of the L_1 ball is fixed with $z = 100$. So Adagrad takes the better ways to gradient descent.

Like the other algorithms, our implementation of Adagrad takes the same iterations, the same seed and the same version of "stochastic" and the same radius $z = 100$. Comparing to the Projected SGD, the rate of convergence is slower because the optimal regret bound is $O(\sqrt{T})$ while $O(1/\sqrt{T})$ for SGD. Taken the same seed in my computer, even if the accuracy of Stochastic AdaGrad has the lower accuracy rate in the beginning, the better convergences in the end.

Since the cost of each iteration for OMD algorithms are more expensive, the training time for the Projected SGD is faster than Adagrad, which are verified as Tab.1: Adagrad takes 4.673 secs, which is 0.375 secs slower than Projected SGD 4.298. The cost didn't takes so much time, it is due to the projection from dual space to primal space is explicit in SVM by solving the dual problem $arg\min_{w \in B_1(z)} \sum d_{i=1}||w - x||^2_D$:

$$\begin{aligned} \theta^* &= \frac{1}{\sum_{j=1}^{d_0} D^{-1}_{(j)}}(\sum_{j=1}^{d_0} x_{(j)} - 1) \\ w^* &= D^{-1} SoftThreshold(Dx, \theta^*) \end{aligned} \tag{11}$$

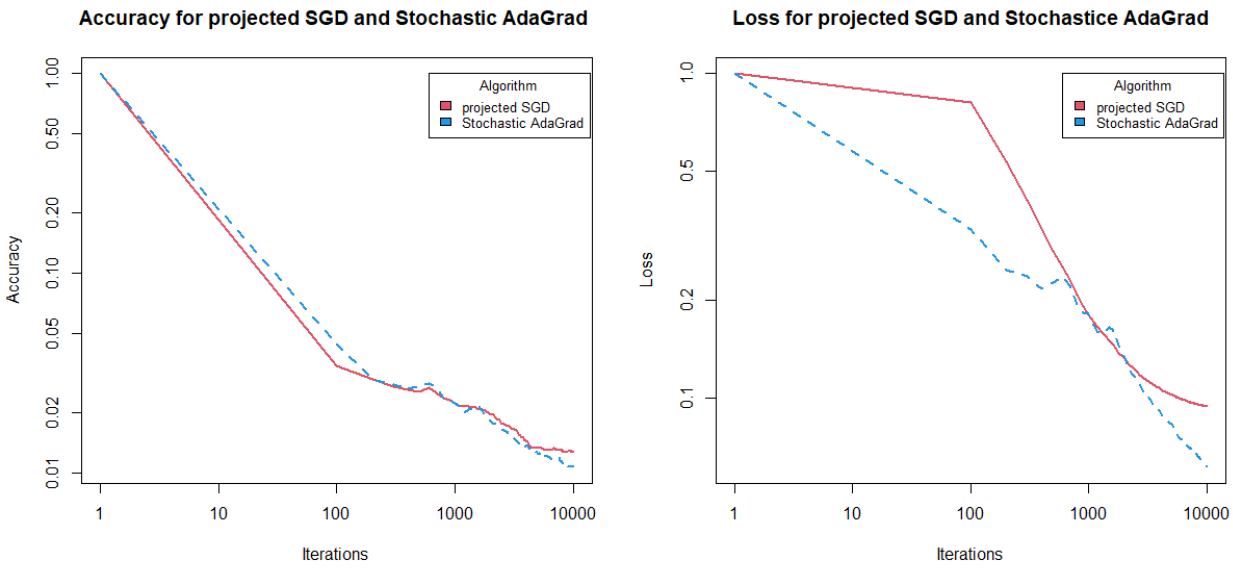

Figure 10: Comparison Projected SGD and Stochastic AdaGrad

5 Online Newton Step

Online Newton Step is an accelerated OCO algorithms without extra assumption on the loss function. It is based on the classic Newton-Raphson algorithm: $x_t + 1 = x_t - \eta H_t^{-1} \nabla f(x_t)$, where $\eta > 0$ and replace H_t by $\frac{1}{t} \sum_{k=1}^{t} \nabla f_k(x_k) \nabla f_k(x_k)^\top$.

The idea comes from classic Newton-Raphson method for any function f finding it's point zero, which is x^*, *such that* $f(x^*) = 0$, we might interpret it with the Taylor formula, the function might rewrite as $f(x^*) = f(x_0) + f'(x_0)(x^* - x_0)$, where x_0 is a point near the point x^*.and so we have $x^* = x_0 - \frac{f(x_0)}{f'(x_0)}$ as we take $f(x^*) = 0$. Then the iteration for finding the zero point x^* for function f can be express as follow:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

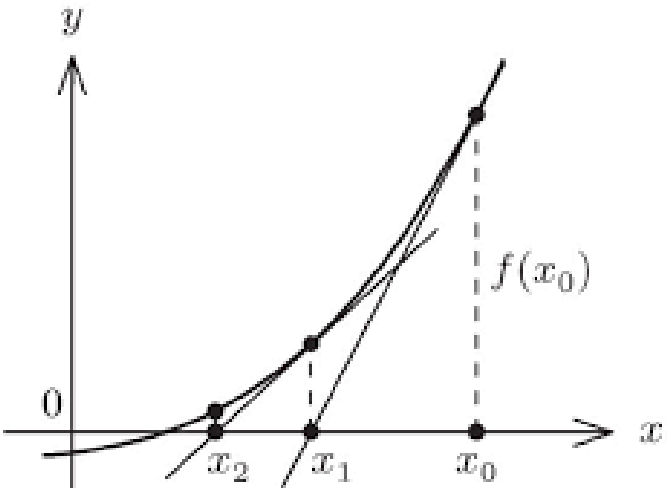

Figure 11: Illustration of Newton method

From the illustration above, we can easily understand the newton method, we take straight line with the gradient of our starting point on the function and get our new starting point for the next iteration where the straight line cross the axis X and doing the same thing, till we reach the point where the function equals to zero.

And as in our problem in this course, what we want to find is the minimum point instead of the zero point for the function f_t which is our loss function. So with Newton's method, we replace the function f by the gradient of our loss(cost) function f_t, noted ∇f_t , we replace the gradient of f in Newton's method by the Hessian matrix which is the second order gradient. It is easy to see that we are here finding the zero point for the gradient function of our loss function, which is x^* *such that* $\nabla f_t(x^*) = 0$, this point lead us to the minimum of the loss function.

Without loss the generations,our implementation for this algorithm is chose the same hyper-parameters as previews $\lambda = \frac{1}{3}$, but there is another hyperparmater need to to decide γ: the hyperparameter to construct H, more precisely:

$$y_{t+1} = x_t - \frac{1}{\gamma} A_t^{-1} \nabla f_t(x_t) \tag{12}$$

As the course mentioned that, it is very complicated to calibrate and the choice of γ is tricky, we tuned it as following Fig. 12, we can see that $\gamma = 1/3$ takes the best accuracy in the end. Moreover since the optimal regret bound for ONS is $O(logT)$ while the optimal regret bound for projected SGD is$O(1/\sqrt{T})$, so the rate of convergence for Projected SGD is faster in theory. From The Fig. 12, $\gamma = 1/3$ has lower the rate of accuracy than Projected SGD in the beginning that correspond with the rate in theory. Hence, we chosed $\gamma = 1/3$ for this algorithm.

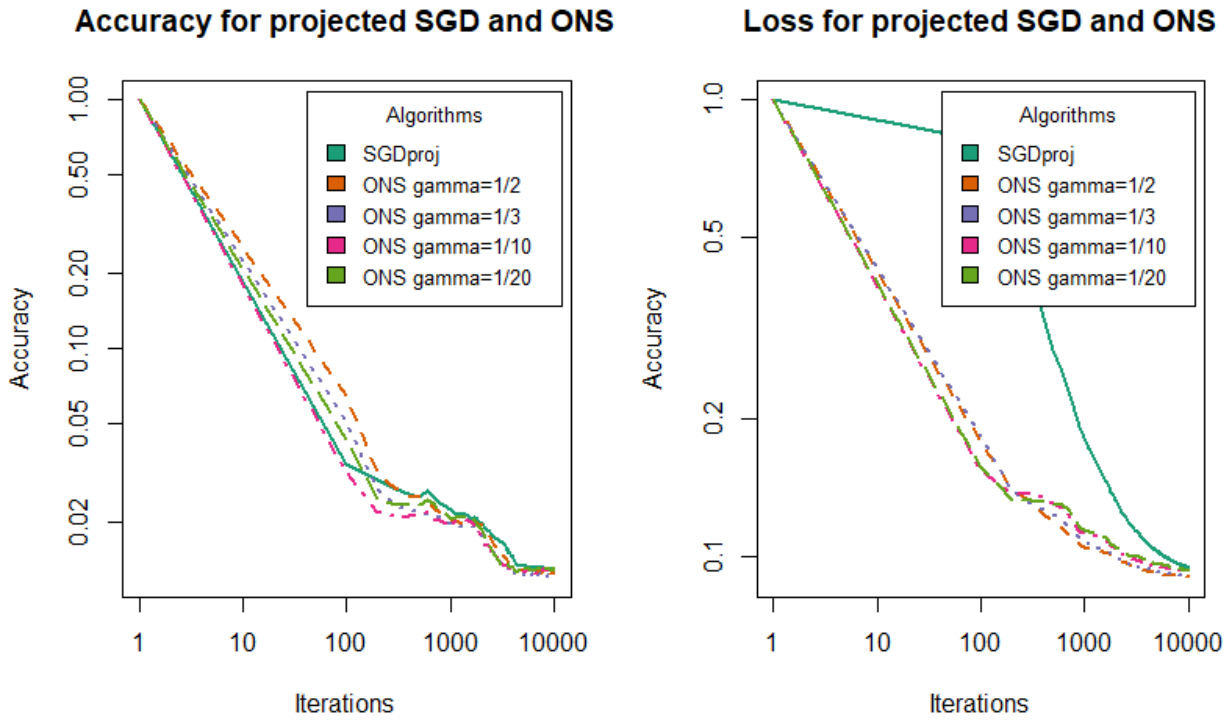

Figure 12: Comparison SGD and ONS with different gamma

Since the projection of algorithm ONS for linear SVM is not explicit, more precisely the $arg\min_{x\in K}||x-y_{t+1}||^2_{A_t}$ is not explicit for A_t non diagonal. so we should appproximate it with $arg\min_{x\in K}||x-y_{t+1}||^2_{Diag(A_t)}$ where $Diag(A_t)$ is the diagnonal matrix extracted from A_t, this approximation required more time to calculate. Moreover, since the x_{t+1} i.e diagonal A_t is taken account into the calculate for the next iterations $t+1$, the cost of each iteration $\mathcal{O}(d^2)$, which is much longer than projected SGD $\mathcal{O}(d)$, From my computer, with the same seed, ONS takes 70 secs while projected SGD takes 4 secs (Tab.1). In fact, the cost of ONS is the most expensive comparing all the other methods mentioned above.

As we mentioned above, all the specific OMD algorithm construct the dual space to do the gradient descent,and projection into the primal space, so the cost of each iteration for OMD algorithms are more expensive than projected SGD. On the other hand, since la projection in these three methods are all explicit, it is not require the time to approximation.Hence the training time of three stochastic version of specific OMD algorithms in Tab.1 took only more 0.4-1.2 secs than projected SGD (4.298 secs).

From Tab.1, the optimal regret bound for Projected SGD $O(1\sqrt{T})$ is the smallest in theory. which is also verified from the Fig. 14: the line red for Projected SGD is under all the other lines at the beginning. But at the end of experiments, the version stochastic of specific OMD method has the better accuracy than Projected SGD and ONS, Among the three specific OMD algorithms, Adaproj and SEG+/- have the better performances than SMDproj, and Adaproj has a little bit better accurary than SEG+/-. In fact, Adaproj has the best performance among all the algorithms.

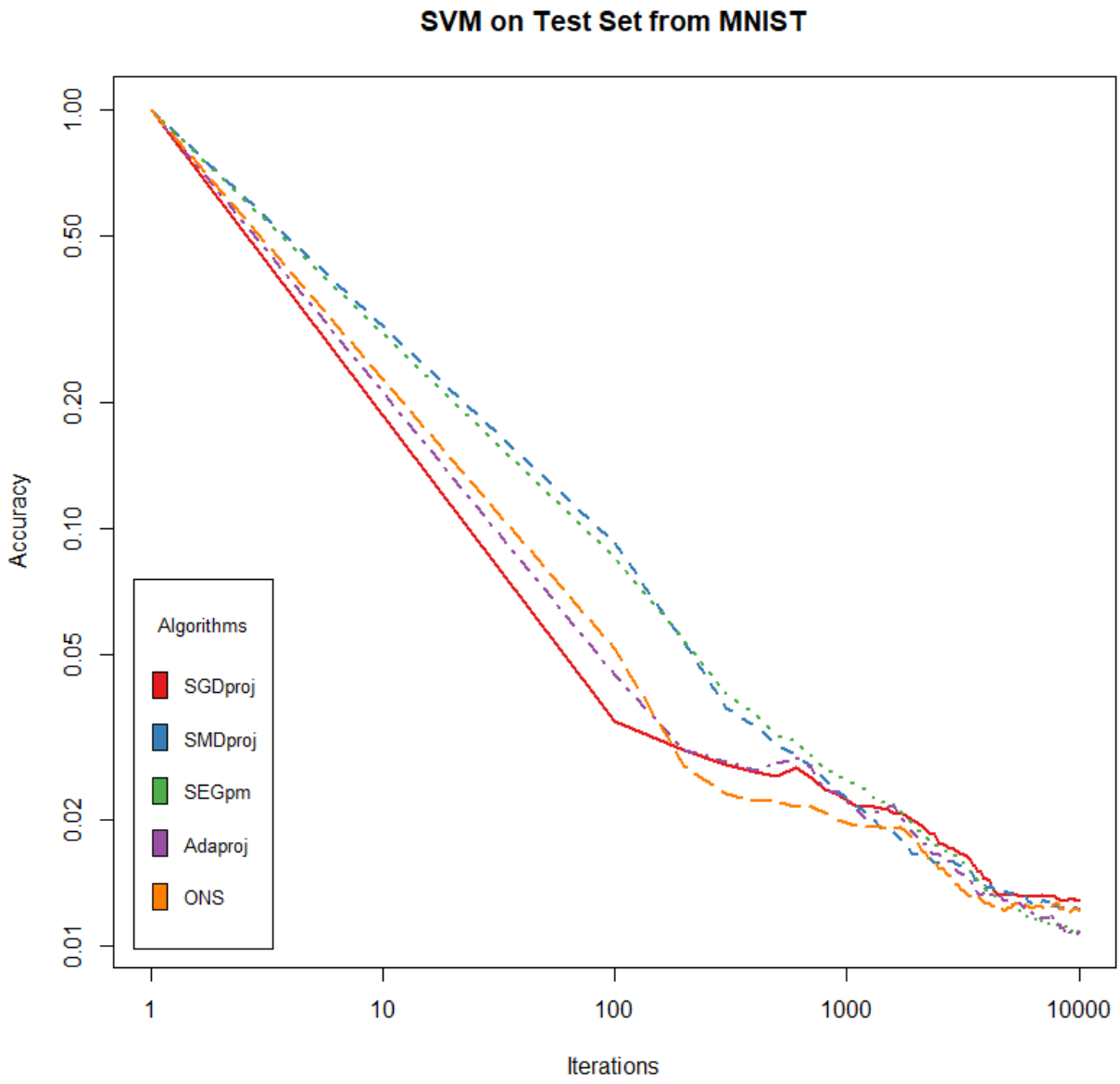

Figure 13: Comparison ONS and the other methods

Algo	Training Time	Regret Bound
SGDproj	4.298 secs	$\frac{3GD}{2\sqrt{T}}$
SMD	5.667 secs	$GD\sqrt{T/2}$
SEG	4.918 secs	$G_\infty z\sqrt{log(2d)/T}$
AdaGrad	4.673 secs	$2D_\infty\sqrt{cT}(1+log\, d)$
ONS	69.902 secs	$2(4GD+1/\mu)d\, log\, T$

Table 1: Comparison Training Time and Regret bound

Considering the training time and performance, we can conclude that AdaGrad is the best Algorithm among the method mentioned above.

6 Exploration methods

We introduce here the Multi-Armed Bandit (MAB) setting. MAB is a very famous game we can play in Casino. In this game, we do not have prior knowledge of a bandit before we play it. The more we play a bandit the better we learn its performances meanwhile we need to explore other bandits at the same time. Generally speaking, bandit setting is used when the information is not complete.
The lose is partially observed as l_{t,i_t} but the regret is not changed as in the Expert Advice setting.

$$\mathbb{E}[Regret_T] = \sum_{t=1}^{T} \mathbb{E}_{x_t}[l_t] - \min_{1\leq i\leq d} \sum_{t=1}^{T} l_{t,i}.$$

During the optimisation process, sometimes that the dimension of problem may be too high, let's say **d** dimension. We take one direction of the gradient for each iteration randomly. While the experience of the gradient is unbiased given:

$$\mathbb{E}[\widehat{\nabla f_t(x_t)}] = \sum_{j=1}^{d} d\nabla f_t(x_t)_j e_j \mathbb{P}(i_t = j) = \sum_{j=1}^{d} \nabla f_t(x_t)_j e_j = \nabla f_t(x_t) \tag{13}$$

For this kind of problem, we need to make a trade off between the exploration and exploitation. While for SREG +/-, the exploration and exploitation are totally independent as the algorithm keeps exploring new directions randomly at each time step.
The regret bound we got here is :

$$\mathbb{E}[Regret_T] \leq z\sqrt{2dGTlog(d)}$$

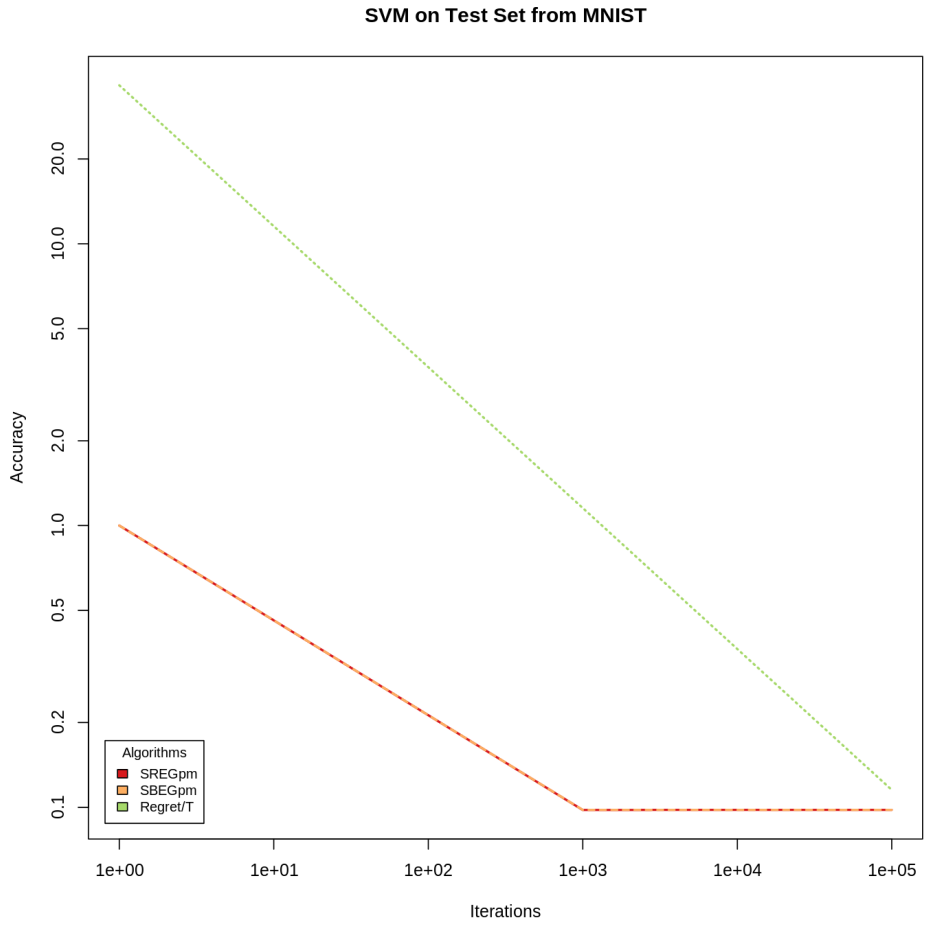

Figure 14: Accuracy of SREG +/- and SBEG+/- for SVM test data set.

We found that, even thought it is accordance with the regret bounds established during OCO lectures, we can not achieve a good accuracy.
We think that it is due to the imbalance of the classes during training. As there are only 10% of sample of digital 0 in data set, the algorithm explores at each round and it is hard to make a good trade off between exploration and exploitation.

7 Faster Projection-free Online Learning

In the previous methods, almost all of them are projection version methods with a projecting step, we project the results(parameters) we get into the constraint set $\mathcal{K}$ every iteration, and then at the end, we have a result more efficiently. But the computation for the projecting step cost a lot of computing power, which might cause to bottleneck in some case, especially in high dimensional problem. If we replace projections by linear optimizations, we have a best regret bound scales as $T^{3/4}$, but this paper gives a projection-free algorithm that guarantees $T^{2/3}$ for convex optimization with smooth cost functions.

7.1 Idea of projection-free

Generally in our algorithm, whenever we take a step that causes the current iterate to leave the convex domain of interest (the constraint set $\mathcal{K}$), thus leading to an infeasible point, we project the point back into the domain $\mathcal{K}$ in order to restore feasibility. The projection step is to find the nearest point in the domain in ℓ_2 distance, and to do so, we usually solving a convex quadratic program over the domain. But in several learning domains of interest, projection steps are computationally very expensive. For the reason of that, there is a way to avoid the projection step in the algorithm, that is replace the projection step by a linear optimization step. The linear optimization programming cost less then the projection step computationally. A standard linear optimization problem is as follow:

$$maximize \ \ c^\top x$$

$$subject \ \ to \ \ Ax \leq b, x \geq 0$$

And in the algorithm, replacing the projection step by linear optimization refer to solve the linear optimization problem $\arg\max_{x\in\mathcal{K}} < y, x >$ instead of the projection problem that solves $\arg\min_{x\in\mathcal{K}} \|x-y\|$, where the norm is euclidean norm. In fact, for the second optimization problem $\arg\min_{x\in\mathcal{K}} \|x-y\| = \arg\min_{x\in\mathcal{K}} \|x-y\|^2$, $\|x-y\|^2$ can be written as $\|x\|^2 - 2 < x, y > +\|y\|^2$, so it is equivalent to solves $\arg\min_{x\in\mathcal{K}} \|x\|^2 - 2 < x, y >= \arg\max_{x\in\mathcal{K}} 2 < x, y > -\|x\|^2$, which is similar to linear optimization but with a L2 norm penalisation. Notice that in fact, the exact solution for the two problem above cannot be computed directly and so that the solutions we get are approximation solutions, note that the two objectives are homogeneous.
According to the reference [Complexity of Linear Minimization and Projection on Some Sets], the complexity of these two algorithm are as shown in the following image:

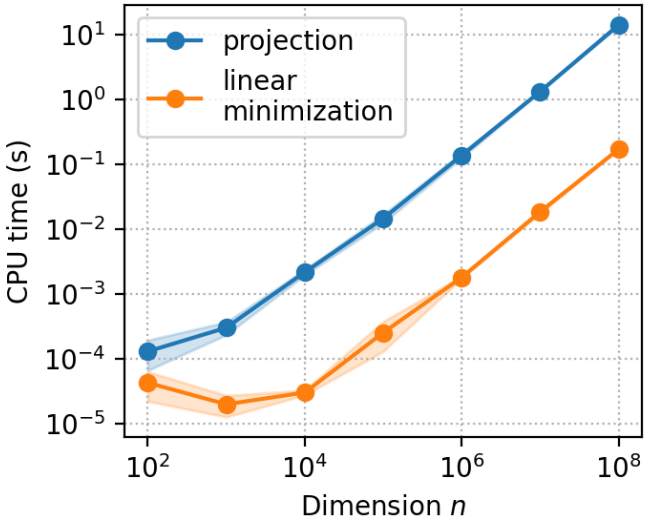

Figure 15: Complexity of Projection and Linear optimization

This image shows us the running time of the different methods, projection and linear minimization on the ℓ_1-ball respectively, which is the case when we assume sparsity and also the general case in our course. In this situation, the plots suggest that linear minimization is about 100

times faster to solve the problem than projection when the dimension n is large enough. So the reliance on linear optimization is the key motivation of this paper.
With the new algorithm proposed in this paper, we aim to implement online gradients methods for training linear SVM on MNIST dataset, replacing the projection step in previous algorithm by the linear optimization programming and compare the computing speed to those algorithms contain projection step.

7.2 Construction of Linear Optimization Oracle

We are now going to construct a linear optimization oracle, in this project, we only pay attention to the constrained set $\mathcal{K}$ is ℓ_1-ball. The reference [Complexity of Linear Minimization and Projection on Some Sets] shows that for linear minimization, for all $y \in \mathbb{R}^d$,

$$\arg\min_{\|x\|\leq 1} < y, x >= -sign([y]_{i_{max}})e_{i_{max}}$$

where $\{e_1, e_2, ..., e_d\}$ denote the standard basis in $\mathbb{R}^d$, $i_{max} \in \arg\max_{i\in\{1,..,d\}} |[y]_i|$ the axis of the biggest absolute value of the coordinate of y, we can understand this by the illustration in dimension 2.

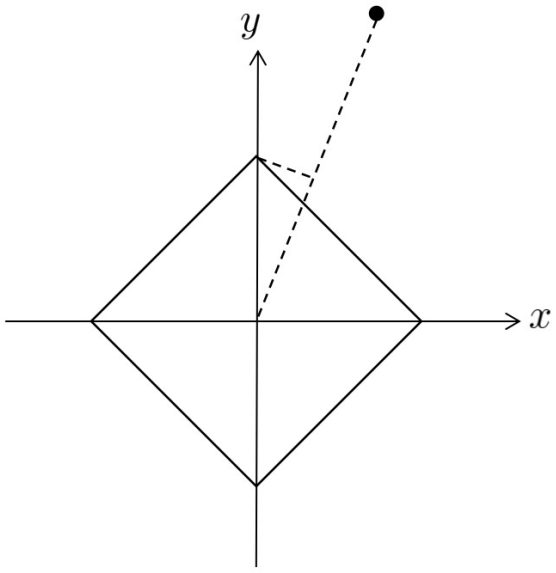

Figure 16: illustration of linear optimization in dim 2

We observed in this image that, the vector x in the ℓ_1-ball who maximize the Dot product $< z, x >$ is the vector $(0, 1)$ instead of the "real projection" who is on the border (1,0) to (0,1). This different comes from the L2 norm penalisation we have mentioned above.

To transform to the form that we want in this paper, which is $\arg\max_{\|x\|\leq 1} < y, x >$, we can obtain

the result as follow:

$$\begin{aligned} \max_x < x, y > &= \min_x - < x, y > \\ &= \min_x < -x, y > \\ so\ that \quad \arg\max_x < x, y > &= \arg\min_x < -x, y > \\ we\ pose\ z = -x, \quad &= \arg\min_{-z} < z, y > \\ &= -\arg\min_z < z, y > \end{aligned}$$

And then we get

$$\arg\max_{\|x\| \leq 1} < y, x >= -\arg\min_{\|x\| \leq 1} < y, x >= sign([y]_{i_{max}}) e_{i_{max}}$$

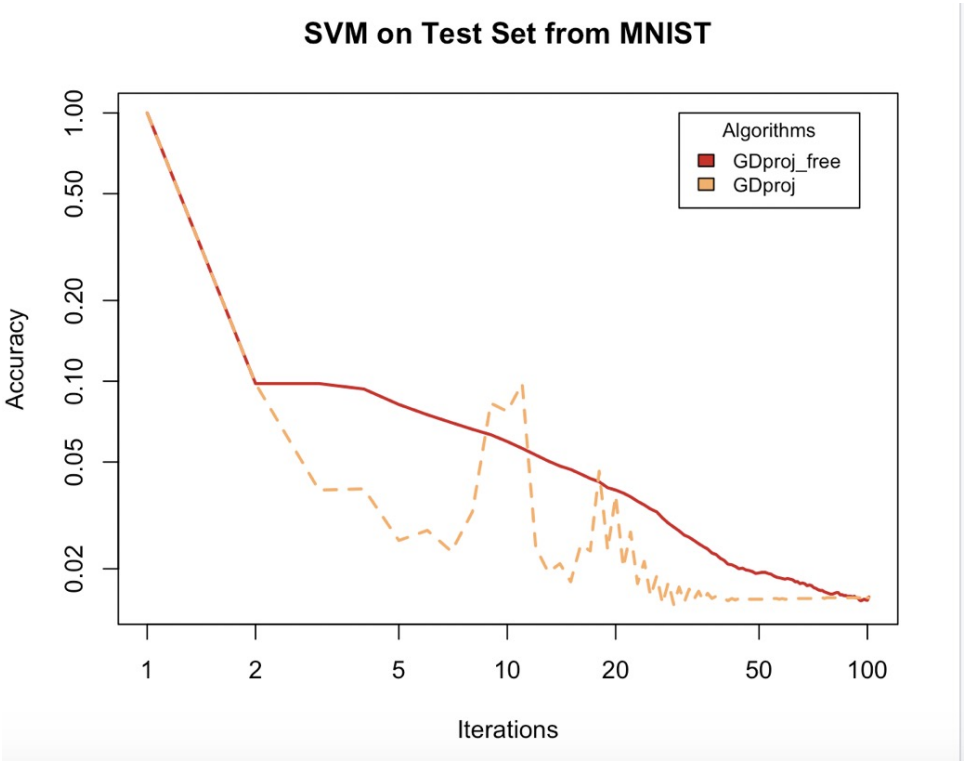

Figure 17: Comparison of projection free GD and projected GD

The image above is we replace the projection step (function pib1) by a linear optimization function we write as defined above, we can see that both method can reach the minimum around 0.015, but for the projection free method, it needs more iterations T to reach that, which is mentioned by the paper.

7.3 Idea of Follow-the-Perturbed-Leader

The algorithm in this paper is an lazy version algorithm because for the gradient descent step, we keep using the point that didn't go through the "projection" step which is now in this paper the linear optimization step.

Since the variable v_t is randomly uniformly generated in a unit ball, we interpret it as a random direction with a value less then 1, not equals to one because we count all the space in the ball. We consider it as a random initial point which match the word "Perturbed" in it's name.

At each iteration t, we randomly select m variable v_t as our initial points and start our gradient descent, but the gradient is form the first iteration to the current one, that means we will go the same trajectory for the 1 to $t-1$ step and a new step depends on the new gradient of $f_t(x_t)$, but the initial point is different for any iteration t and every $j \in \{1, ..., m\}$ in each iteration. And then we compute the average location for the m points of x_t, which is meaning the expectation of the location where x_t should be.

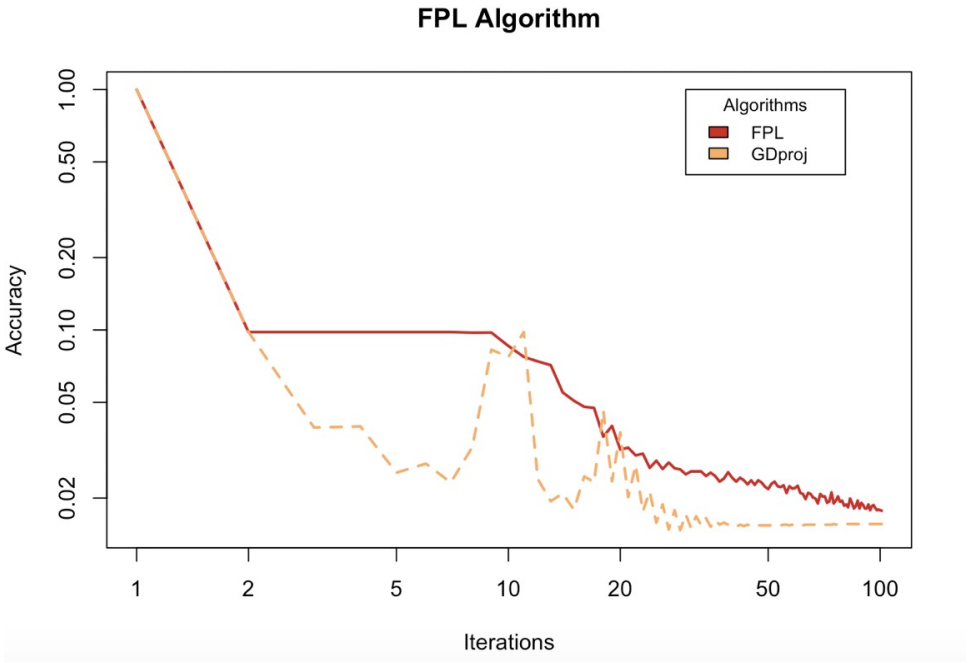

Figure 18: Comparison of FPL and projected GD

The image above shows us the comparison of Follow-the-Perturbed-Leader algorithm and classic projected gradient descent with setting $m = 1$. We can see that due to the randomly sampling of the initial points, there are some fluctuation at the end of the curve. And i need to mention that, in our experimentation, if we do not add a learning rate multiplying the gradient(which is the case in the paper) every time we iterate, the movement of each iteration might be too big that it will escape from the minimum and has a very big fluctuation and never settle to a minimum point, so we add a fix learning rate $\eta = 1/(\lambda * iters)$, and then we get the curve for FPL algorithm above.

For the following step we replace the projection step by linear optimization with our the function oracle that we defined above, and now the algorithm is the same as the algorithm 1 proposed

in the paper, which is Sampled Follow-the-Perturbed-Leader algorithm and we compare it with the classic projected Gradient descent, what's more, we now set the number of sample $m = 100$ to see what happen if we take $m = O(T)$ which mention in the paper. we have the image as follow, where the curve of FPL with linear optimization is different but almost the same as FPL with normal projection step, and the augmentation of m brings the stability of the curve.

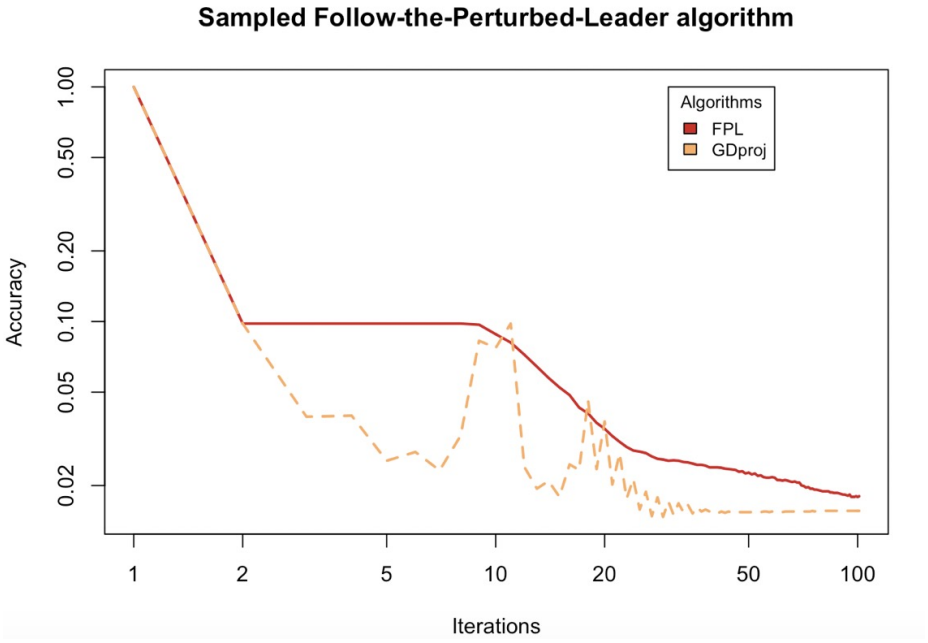

Figure 19: Sampled Follow-the-Perturbed-Leader algorithm

The idea of the algorithm 2 in this paper is that we do not update the parameters x_t, the gradient $\nabla f_t(x_t)$ and do the linear optimization for each iteration t, but we do this once every block size k, and every time when we do the linear optimization, we sample k random points v which can be consider as the initial points, and we compute the average location as we did before. This helps to do reduce the real iteration times but with the same length of step as before, so according to the paper, under this algorithm, we reach a faster descent of the regret with respect to the iteration times T, which is $O(T^{2/3})$, faster then the classic projection free method.

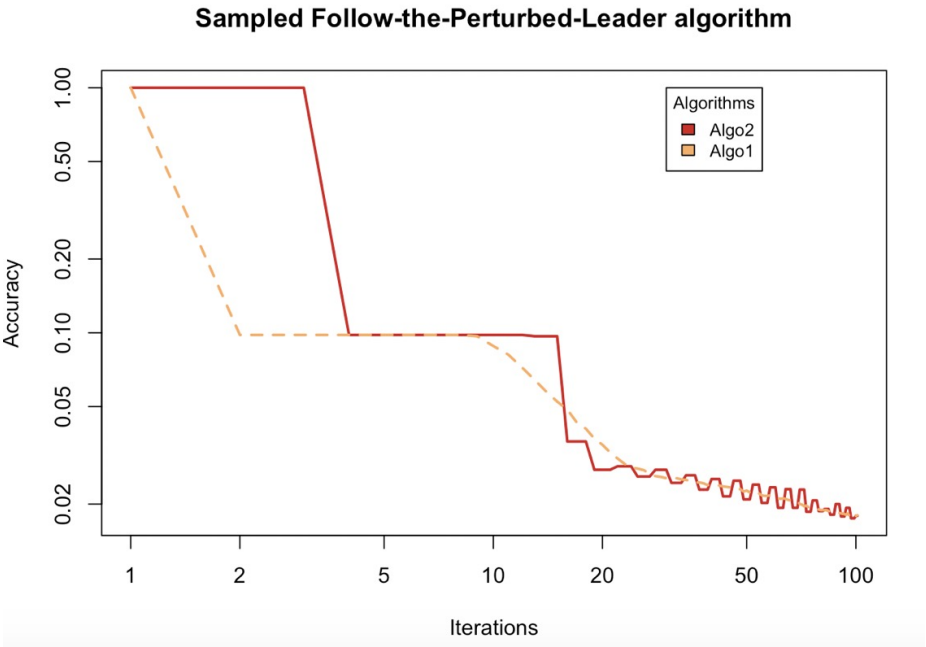

Figure 20: OSPE Algorithm

This image shows us the curve of the Online smooth Projection Free algorithm where we set $k = 3$, that means we sample the vector v every 3 times of iterations, the reason why k is small is because here we use all our data (observations) at each iteration step since in this algorithm we need to calculate the real gradient $\nabla f_t(x_t)$ rather then the expectation one in stochastic methods, the total number of iterations is only 100, so we can't set a k too large here. We can see that at the end this method can reach the same minimum as the previous method but thanks to the less computation, it's faster then the one before.

7.4 Discussion

The author is proposing a projection-free method, for the reason that to have less think to compute at each iteration, by replacing the projection step by a linear optimization problem, and for the limit set constraint set $\mathcal{K} = L_1$ we have the explicit formula defined above. And the author use this linear optimisation oracle in a Follow-the-Perturbed-Leader algorithm which is new for me. At first, we do the sample step, linear optimization step, computing average step and update parameters step in every iteration, that doesn't give us a faster descent of regret bound with respect to classic projection free method, and then the author proposed that we do the four step every k iteration, and for the other iteration, we just keep the same parameter as the previous iteration and compute the gradient. That can bring us a faster descent of the regret bound.