# Online Convex Optimization

M2 Apprentissage & Algorithmes

HACHAMI Mohamed, SURENDRAN Sobihan, VIOLLE Armand

December 2022

---

## Project

---


Academic year 2022-2023

# Contents

# 1 Generalities and structure

## 1.1 Data

As mentioned in the instructions, we worked with the MNIST dataset since it is considered a very basic dataset and is frequently used as a starting point for benchmarks. MNIST (Modified National Institute of Standards and Technology) is a database of handwritten digits. It has a training set of 60,000 28x28 pixel grayscale images of handwritten digits (0-9), and a test set of 10,000 images. Each image is labeled with the correct digit. In the dataset, the first value is the label (a number between 0 and 9), and the following 784 values are the pixel values (a number from 0 to 255).

As we're considering the binary classification problem of 0 versus all other digits, we labeled 0s as 1 and other values as -1, thus $\forall i \in [n],\ y_i \in \{\pm 1\}$. The digits have then been size-normalized and centered in a fixed-size image. We also added a bias (a $\mathbb{1}$-valued pixel).

## 1.2 Code structure

The implementation was done using a GitHub repository that can be found at the following link. The structure is the following :

- `utils.py` : file containing all the tool functions we had to use in our code : importing datasets, defining the SVM gradient and prediction functions, cross-validations and measuring performances.
- `convex_tools.py` : as its name indicates, this file regroups the functions we had to use in order to project the updates on the constrained convex sets (simplex, $l1$-ball and generalized projections).
- `gradient_descent.py` : this file regroups all the optimization algorithms we've implemented.
- `main.py` and `plots.py` : these scripts were used to generate results.

## 1.3 Initialization of coefficients $w$

Recalling the objective function of the SVM optimization problem, $f(x) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i w^T x_i)$, we decided to initialize $w = \mathbf{1}$ in our algorithms unlike what was suggested in the algorithms. This choice has two motivations :

1. We're considering the classification of 0s versus all other digits, which means that even with a random classifier, we could reach an accuracy of 90% (0.9 in decimals).
2. For algorithm having a projection update, we observed that most of the time, even after a large number of epochs, $w$ wasn't updated by projection. This is due to the fact that initializing with 0s gives small coefficients to $w$, and thus the condition $\|w_t\|_1 \leq z$ can be verified nearly throughout the entire algorithm. Initializing with 1s makes it different because the initial norm is $\|w_0\|_1 = 785$, which is big compared to values of $z$ like 10, 50 and 100.

Overall, we've noticed that results were more coherent initializing with 1s, so that's what we did.

## 1.4 Measuring performances

To measure the performances of the algorithms on the test data set, we use the accuracy. The accuracy is defined as the mean number of well predicted labels (written $\hat{y}_i$) over target test labels ($y_i$):

$$ACC = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\hat{y}_i = y_i}.$$

This expression gives a real values in $[0,1]$ that can be interpreted as a percentage in $(0,100)$. In graphs, we used the decimal accuracy (in $[0,1]$), but we'll often make parallels with percentages to be more explicit in our explanations.

# 2 Gradient Descent

## 2.1 A word on SVM

### 2.1.1 Model

In the SVM model, we consider the following Bayes classifier : $g(x) = sign(w^T x + b)$, with here $b = 0$, so :

$$g(x) = sign(w^T x) \tag{1}$$

### 2.1.2 Loss Function

Gradient descent tries to optimize the following optimization, $f$ being the loss function we'll work with from now on :

$$\min_{w \in \mathbb{R}} f(x) = \frac{\lambda}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} max(0, 1 - y_i w^T x_i) \tag{2}$$

To proceed to gradient descent, we need to compute $\nabla f = \nabla_w f$.

$$\nabla_w f(x) = \lambda w - \frac{C}{n}\sum_{i=1}^{n} y_i x_i \mathbf{1}_{y_i w^T x_i \leq 1} \tag{3}$$

That's the expression we are using to compute the gradient in the first algorithms, with the function `grad_svm`.

## 2.2 Unconstrained and Projected Gradient Descents

We had to implement the unconstrained gradient descent and also the projected gradient descent on the $\ell 1$-ball. We tested them on the train set with different values for both $\lambda$ (the regularization/penalization parameter) and $z$ (the radius of the $\ell 1$-ball). Following this comparison, we'll see which values to retain for $\lambda$.

### 2.2.1 Regularization parameter $\lambda$

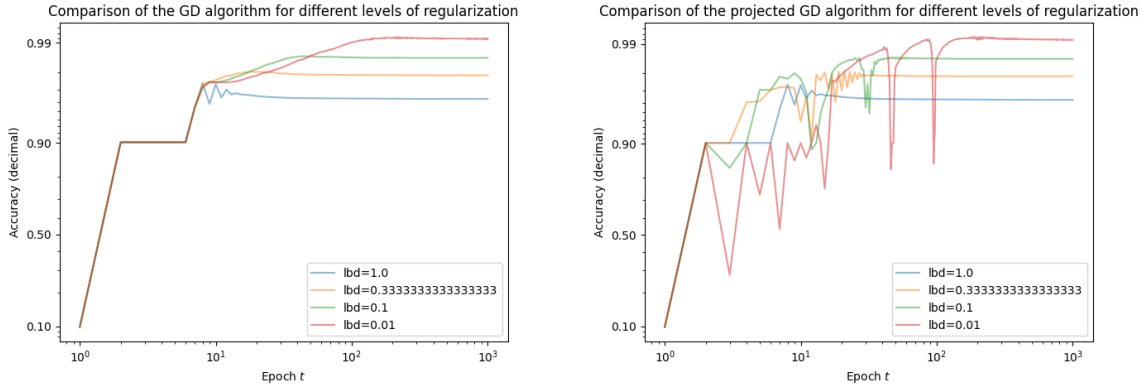


Figure 1: We can observe that for the unconstrained gradient descent, we obtain better performances with $\lambda = 0.01$. It converges slower but with more stability and towards a better accuracy. For the projected gradient descent, we can see that a higher value of $\lambda$ leeds to good stability and a lower value leeds to less stability but better performances. From what we can see, $\lambda = \frac{1}{3}$ seems to give the best tradeoff within the values tested.

$\lambda$ is the regularization parameter for the SVM : the higher it is, the more we will penalize coefficients compared to the empirical risk in the objective function. In learning, we want to avoid overfitting, which means having the empirical risk taking over the regularization ($\lambda$ would be too low). What we've seen in terms of tradeoff between performances and stability can be interpreted as a bias-variance tradeoff to avoid overfitting and underfitting.

Thus, we decided to set $\lambda = \frac{1}{3}$ in the following, as it provided a better tradeoff.

### 2.2.2 Radius $z$ of the $\ell 1$-ball (projected GD)

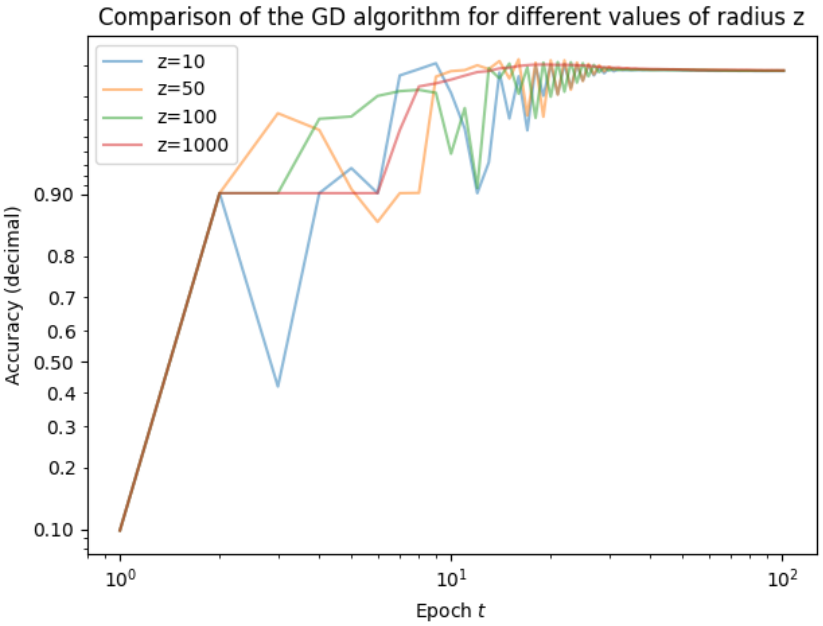


Figure 2: We can see from this experiment that $z$ has an influence over the speed and stability of the convergence. We cannot draw general conclusions for other algorithms, but we can notice that the better option for projected GD seems to be $z = 1000$.

The radius $z$ levels the constraint over the SVM coefficients $w$: the lower it is, the more contrained is the problem. For low values of $z$ , the algorithm will have the tendency to annul the coeffients corresponding to pixels not really helping for the classification, giving more importance to meaningful pixels. But we have to be careful : very low values of $z$ will decrease the performances because we won't be able to quantify enough pixel information to build a good classifier.

We will run a validation for all algorithms on the train set to find a good $z$ value among [10, 50, 100, 1000] and compare the influence $z$ has over the algorithms.

# 3 Stochastic Gradient Descent

Using the notations of the course, we have $\ell_{a,b}(w) = \max(0, 1 - bw^T a)$, the gradient updates are made using the gradient $\nabla\ell$. We provided another function to do these gradient updates named `grad_hinge`.

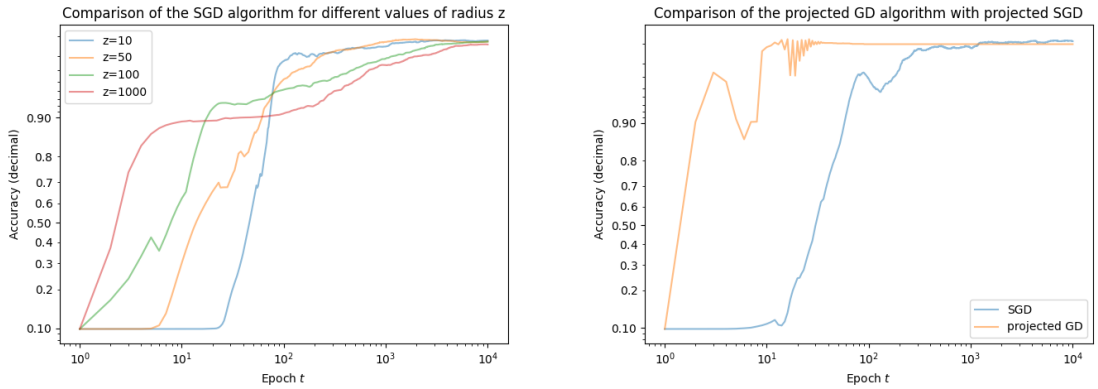


Figure 3: Similarly to what we've seen previously, we can see that from this validation that the higher $z$ is, the more stable the algorithm is and the slower convergence is. From all the values, 50 seems the one suiting best this algorithm. From the comparison between projected GD and projected SGD, we can state that convergence of SGD maybe be a little bit slower but the performaces are slightly better in the end and the stability is much better.

# 4 Regularized Follow the Leader

## 4.1 Stochastic Mirror Descent

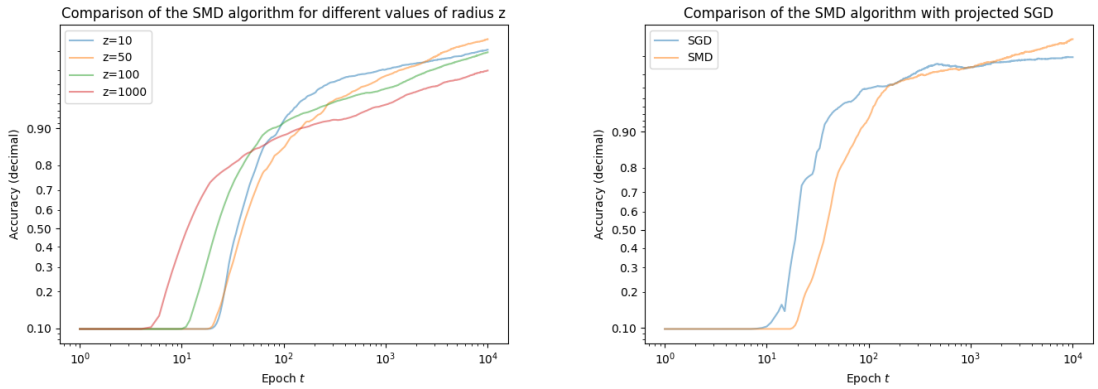


Figure 4: We can see from the train dataset validation on $z$ that the best value for SMD seems also to be 50 like we found in SGD. Comparing the two algorithms, we can state that SMD provides better performances and a better stability than SGD.

## 4.2 Stochastic Exponentiated Gradient

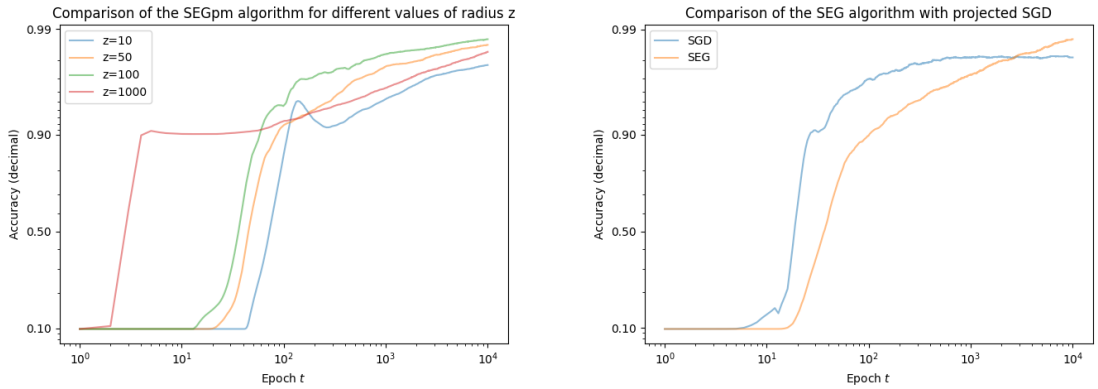


Figure 5: SEG doesn't actually include a projection update within its steps, but still uses $z$ in $w_t$ updates. From what we can observe on the validation curves, $z = 100$ seems the better option for SEG. Comparing with SGD, SEG is a little slower to converge but provides better performances ultimately.

## 4.3 Adagrad

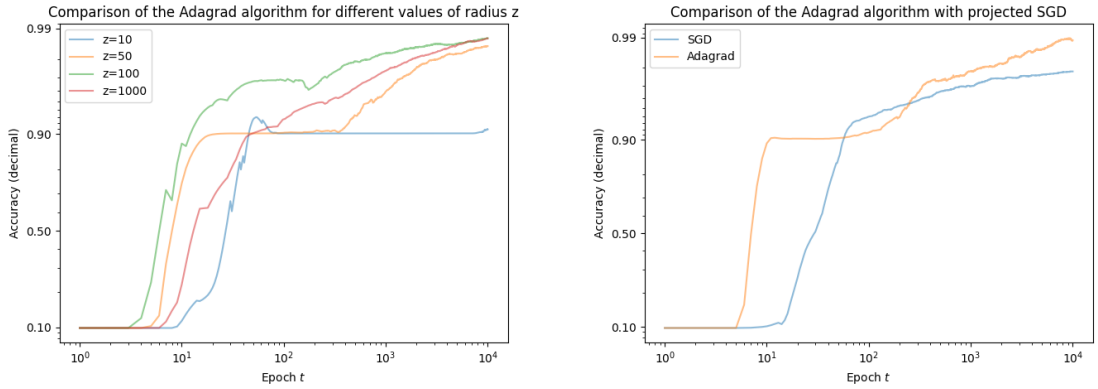


Figure 6: We can see from the left validation graph that the *optimal* radius $z$ for Adagrad seems to be 100, as it provides fast convergence and good stability. Compared to SGD, it reaches a higher level of accuracy (close to 99%).

# 5 Online Newton Step

## 5.1 Validation on $z$

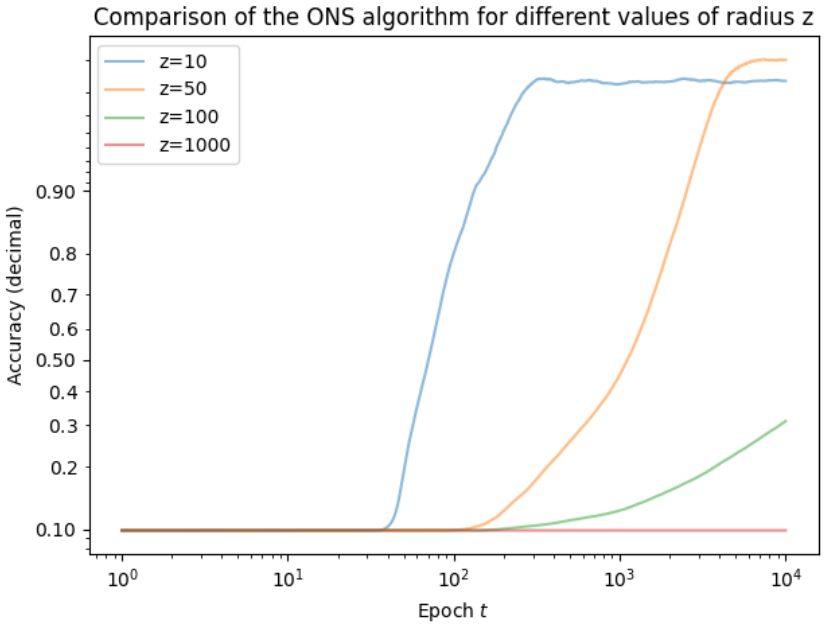


Figure 7: As we can see, $z$ has a huge influence over the ONS algorithm. The lower $z$ is, the faster convergence is. Nevertheless, the best performances are obtained for $z = 50$. In practice, we did some more tests and found that $z = 15$ was offering the better tradeoff for ONS.

## 5.2 Overall performances for algorithms

We regrouped execution times of all projected algorithms with their *best* parameters $z$ inside a single table (except exploration-based algorithms) to compare their time efficiency over 10000 epochs.

| Algorithm | Radius $z$ | Execution time |
|---|---|---|
| GD | 50 | $\sim$ 26 min |
| SGD | 50 | 0.21 sec |
| SMD | 50 | 7.94 sec |
| SEG | 100 | 0.37 sec |
| Adagrad | 100 | $\sim$ 4 min |
| ONS | 15 | $\sim$ 6 min |

Table 1: Comparison of execution times of projected algorithms

We can see that the gradient descent takes a lot of time compared to other algorithms is really big, which must be due to the fact that gradient updates are made in a full batch way. Also, more elaborated algorithms like Adagrad and ONS using different kinds of matrix computations are slower than the others.

Let's take a closer look now at the performances of their respective classifiers throughout the descent. GD wasn't included to this graph due to it's very high execution time, so we compared only stochastic algorithms.

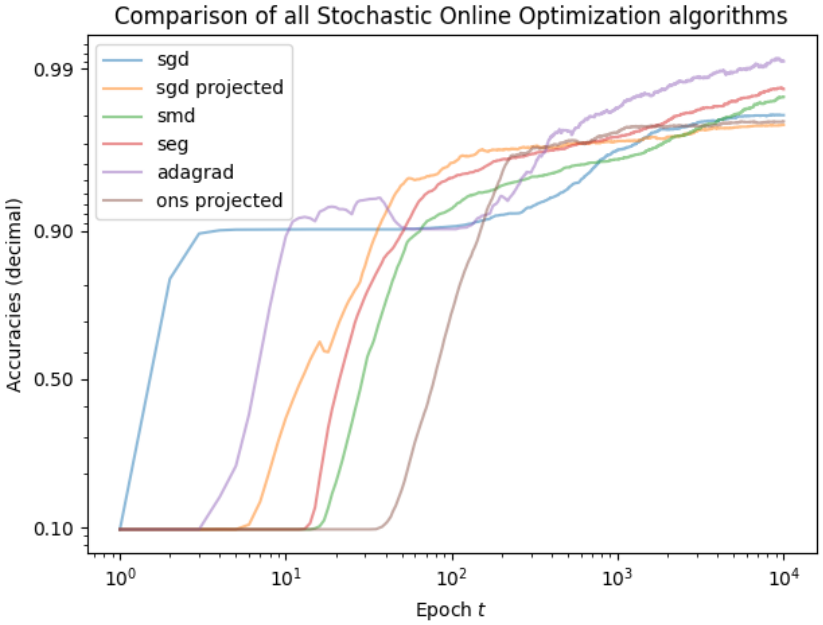


Figure 8: We can see that the best performing algorithm seems to be Adagrad, even if it doesn't seem to be the more stable, with an accuracy above 99% reached in the end. Overall, all algorithms show satisfactory levels of performances above 97%. Maybe with more epochs we could have witness changes in this *hierarchy*, but that was technically impossible due to the high execution time of test predictions and accuracy computing.

# 6 Exploration Methods

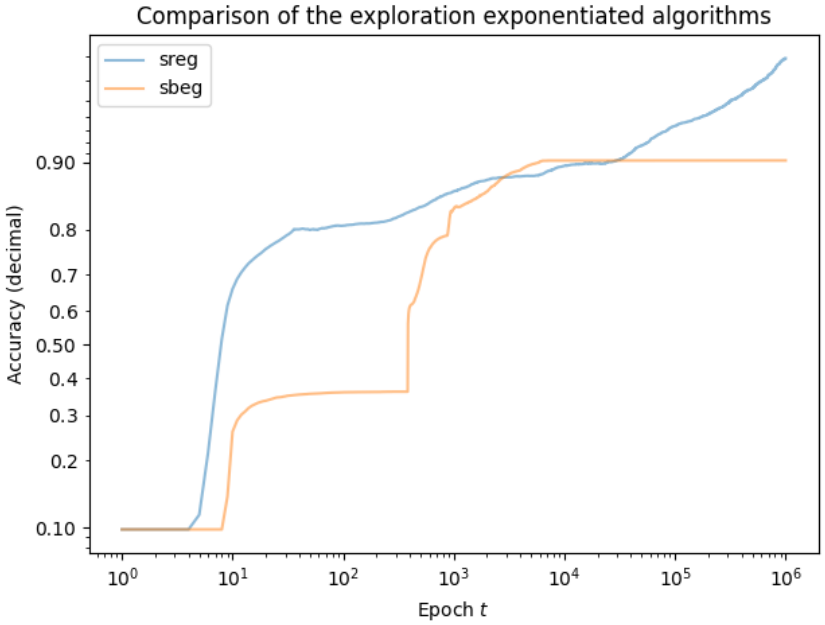


Figure 9: The curves were drawn using a radius $z$ of 100 for both algorithms. We can see that these methods are slower to converge compared to stochastic algorithms, with a little bit worse performances. The convergence of SREG is smoother than SBEG and it ultimately provides better performances. Nevertheless, comparing these curves to the ones provided in class, SBEG reaches the *plateau* of 90% accuracy, but SREG accuracy continues to increase pass that value, which is interesting to highlight.

The regret bound established during OCO lectures on the accuracy for exploration methods is (page 60) :

$$\mathbb{E}[h_T^R] \leq \frac{zdG_\infty\sqrt{2\log d}}{\sqrt{T}} \text{ for SREG}$$
$$\mathbb{E}[h_T^R] \leq \frac{zG\sqrt{2\log 2d}}{\sqrt{T}} \text{ for SBEG}$$

In the lectures, what was called accuracy was the 0-1 loss, so in our case :

$$\begin{aligned} ACC = 1 - \mathbb{E}[h_T^R] \geq 1 - \frac{zdG_\infty\sqrt{2\log d}}{\sqrt{T}} \text{ for SREG} \\ ACC = 1 - \mathbb{E}[h_T^R] \geq 1 - \frac{zG\sqrt{2\log 2d}}{\sqrt{T}} \text{ for SBEG} \end{aligned} \tag{4}$$

Here, we have $z = 100$, $d = 785$ and $T = 10^6$. To approximate $G$ and $G_\infty$, the right thing to do would be to compute all the gradient of $f_t$ over the epochs and return the maximum value of the norm. By lack of time, we simply approximated them by $\sqrt{d}$, similarly to what can be done in Adagrad. This gave us the following bounds with unit roundings :

- $\approx -8029$ for SREG
- $\approx -10$ for SBEG

The bounds established in (4) are thus verified, but we're not very satisfied of them because we cannot interprete them.

# 7 Adaptive Subgradient Methods for Online Learning and Stochastic Optimization

## 7.1 Introduction

Adaptive subgradient methods are a class of optimization algorithms that adjust the learning rate for each model parameter based on the past gradients that have been computed for that parameter. This allows the model to automatically adjust the learning rate for each parameter. They are frequently used to train machine learning models. Adaptive subgradient methods are often used in situations where the data is sparse or where the objective function has a highly irregular structure. They are also well suited for online convec optimisation problems. Some popular adaptive subgradient methods include Adagrad, Adadelta, and RMSprop. These algorithms differ in the way they update the learning rate for each parameter. AdaGrad and its variants, which update the stepsize in stochastic gradient descent on the fly in accordance with the gradients received along the way, are adaptive gradient methods that have become very popular in large-scale optimization because they have a strong tendency to converge. The theoretical assurances for AdaGrad, however, focus on online and convex optimization.

## 7.2 Algorithms

INPUT: $\eta > 0, \delta \geq 0$
VARIABLES: $s \in \mathbb{R}^d, H \in \mathbb{R}^{d\times d}, g_{1:t,i} \in \mathbb{R}^t$ for $i \in \{1,\dots,d\}$
INITIALIZE $x_1 = 0, g_{1:0} = []$
FOR $t = 1$ to $T$
Suffer loss $f_t(x_t)$
Receive subgradient $g_t \in \partial f_t(x_t)$ of $f_t$ at $x_t$
UPDATE $g_{1:t} = [g_{1:t-1}\ g_t]$, $s_{t,i} = \|g_{1:t,i}\|_2$
SET $H_t = \delta I + \mathrm{diag}(s_t)$, $\psi_t(x) = \frac{1}{2}\langle x, H_t x\rangle$

Primal-Dual Subgradient Update (3):

$$x_{t+1} = \operatorname*{argmin}_{x\in\mathcal{X}} \left\{ \eta \left\langle \frac{1}{t}\sum_{\tau=1}^{t} g_\tau, x \right\rangle + \eta\varphi(x) + \frac{1}{t}\psi_t(x) \right\}.$$

Composite Mirror Descent Update (4):

$$x_{t+1} = \operatorname*{argmin}_{x\in\mathcal{X}} \left\{ \eta\langle g_t, x\rangle + \eta\varphi(x) + B_{\psi_t}(x, x_t) \right\}.$$

INPUT: $\eta > 0, \delta \geq 0$
VARIABLES: $S_t \in \mathbb{R}^{d\times d}, H_t \in \mathbb{R}^{d\times d}, G_t \in \mathbb{R}^{d\times d}$
INITIALIZE $x_1 = 0, S_0 = 0, H_0 = 0, G_0 = 0$
FOR $t = 1$ to $T$
Suffer loss $f_t(x_t)$
Receive subgradient $g_t \in \partial f_t(x_t)$ of $f_t$ at $x_t$
UPDATE $G_t = G_{t-1} + g_t g_t^\top$, $S_t = G_t^{\frac{1}{2}}$
SET $H_t = \delta I + S_t$, $\psi_t(x) = \frac{1}{2}\langle x, H_t x\rangle$

Primal-Dual Subgradient Update ((3)):

$$x_{t+1} = \operatorname*{argmin}_{x\in\mathcal{X}} \left\{ \eta \left\langle \frac{1}{t}\sum_{\tau=1}^{t} g_\tau, x \right\rangle + \eta\varphi(x) + \frac{1}{t}\psi_t(x) \right\}.$$

Composite Mirror Descent Update ((4)):

$$x_{t+1} = \operatorname*{argmin}_{x\in\mathcal{X}} \left\{ \eta\langle g_t, x\rangle + \eta\varphi(x) + B_{\psi_t}(x, x_t) \right\}.$$

Figure 10: ADAGRAD with diagonal matrices (left algorithm) and full matrices (right algorithm)

The equations (3) and (4) in the figure 10 can be written as :

$$x_{t+1} = \arg\min_{x\in\chi} \quad \{\langle u\ ,\ x\rangle + \varphi(x) + \frac{1}{2}\langle x\ ,\ H_t x\rangle\} \tag{5}$$

where $u = \eta t\bar{g}_t = \eta t \sum_{\tau=1}^{t} g_\tau$ for the equation (3) and $u = \eta t g_t - H_t x_t$ for the equation (4).
We consider the setting in which $\varphi \equiv 0$ and $\chi = \left\{x : \|x\|_1 \leq c\right\}$
In the article, it has been demonstrated that (5) for the primal-dual update (3) is equivalent to :

$$\begin{aligned} \underset{z}{\text{minimize}} \quad & \left\| z + H^{\frac{-1}{2}} u \right\|_2 \\ \text{subject to} \quad & \|Az\|_1 \leq c, z = H^{\frac{1}{2}} x, A = H^{\frac{-1}{2}} \end{aligned}$$

$\Longleftrightarrow$

$$\begin{aligned} \min_{x} \quad & \left\| H^{\frac{1}{2}} x + \eta t H^{\frac{-1}{2}} g_t \right\|^2 \\ \text{s.t.} \quad & \|x\|_1 \leq c \end{aligned}$$

$\Longleftrightarrow$

$$\begin{aligned} \min_x \quad & \left\| H^{\frac{1}{2}} x + \eta t H^{\frac{-1}{2}} g_t \right\|^2 \\ \text{s.t.} \quad & \|x\|_1 \leq c \end{aligned}$$

$\Longleftrightarrow$

$$\begin{aligned} \min_x \quad & \left\| H^{\frac{1}{2}} (x - y_t) \right\|^2 \\ \text{s.t.} \quad & \|x\|_1 \leq c, y_t = \eta t H^{-1} g_t \end{aligned}$$

$\Longleftrightarrow$

$$\begin{aligned} \min_x \quad & \left\| x - y_t \right\|_H^2 \\ \text{s.t.} \quad & \|x\|_1 \leq c, y_t = \eta t H^{-1} g_t \end{aligned}$$

The equation (5) for the composite mirror-descent update (4) has a similar form equivalent to :

$$\begin{aligned} \min_x \quad & \left\| H_t^{\frac{1}{2}} x + H^{\frac{-1}{2}} (\eta g_t - H_t x_t) \right\|^2 \\ \text{s.t.} \quad & \|x\|_1 \leq c \end{aligned}$$

$\Longleftrightarrow$

$$\begin{aligned} \min_x \quad & \left\| H_t^{\frac{1}{2}} (x - (x_t - \eta H_t^{-1} g_t)) \right\|^2 \\ \text{s.t.} \quad & \|x\|_1 \leq c \end{aligned}$$

$\Longleftrightarrow$

$$\begin{aligned} \min_x \quad & \left\| x - y_t \right\|_{H_t}^2 \\ \text{s.t.} \quad & \|x\|_1 \leq c, y_t = x_t - \eta H_t^{-1} g_t \end{aligned}$$

which corresponds to the projected Adagrad algorithm seen in the course.

## 7.3 Theoretical results

In online learning, the learner predicts $x_t$ and then the loss function $f_t$ is revealed. Then the algorithm incurs the loss $f_t(x_t)$ and the learner's goal is to achieve low regret at any horizon T :

$$R(T) = \sum_{t=1}^{T} f_t(x_t) - \inf_{x \in \chi} \sum_{t=1}^{T} f_t(x)$$

In the paper, they considered online learning with a sequence of composite functions $\phi(x) = f_t(x) + \varphi(x)$ where $f_t$ and $\varphi$ are (closed) convex functions. In this setting, $f_t$ is is either an instantaneous loss or a stochastic estimate of the objective function in an optimization task. The function $\varphi$ is a fixed regularization function. So, the regret with respect to the fixed (optimal) predictor $x^*$ is defined as :

$$R_\phi(T) \triangleq \sum_{t=1}^{T} [\phi_t(x_t) - \phi_t(x^*)] = \sum_{t=1}^{T} [f_t(x_t) + \varphi(x_t) - f_t(x^*) - \varphi(x^*)]$$

We denote $g_{1:t}$ as the matrix of concatenated subgradients and $G_t = \sum_{\tau=1}^{T} g_\tau g_\tau^\top$ as the outer product matrix. According to the article, the ADAGRAD algorithm with full matrix divergences has bounds of the form :

$$R_\phi(T) = \mathcal{O}(\|x^*\|_2 \mathrm{tr}(G_T^{1/2}))$$

and

$$R_\phi(T) = \mathcal{O}(max_{t\leq T}\|x_t - x^*\| \mathrm{tr}(G_T^{1/2}))$$

with

$$\mathrm{tr}(G_T^{1/2})) = d^{1/2}\sqrt{\inf_S(\sum_{t=1}^{T}\langle g_t\ ,\ S^{-1}g_t\rangle : S \geq 0, \mathrm{tr}(S) \leq d)}$$

When the proximal function takes the form of $\Psi_t(x) = \langle x, diag(G_t)^{\frac{1}{2}}x\rangle$ the algorithm (diagonal version) has bounds attainable in time at most linear in the dimension $d$ of our problems of the form :

$$R_\phi(T) = \mathcal{O}(\|x^*\|_\infty \sum_{i=1}^{d}\|g_{1:T,i}\|_2))$$

and

$$R_\phi(T) = \mathcal{O}(max_{t\leq T}\|x_t - x^*\| \sum_{i=1}^{d}\|g_{1:T,i}\|_2))$$

with

$$\mathrm{tr}(G_T^{1/2})) = d^{1/2}\sqrt{\inf_S(\sum_{t=1}^{T}\langle g_t\ ,\ \mathrm{diag}(s)g_t\rangle : S \geq 0, \langle 1, s\rangle \leq d)}$$

**Some additional results:**
Concerning the algorithm with diagonal matrices, for $x_t$ generated using the primal-dual subgradient update (3) with $\delta \geq \max_t \|g_t\|_\infty$ , assuming $\chi$ is compact ans setting $D_\infty = \sup_{x\in\chi}\|x - x^*\|_\infty$, we have for any $x^* \in \chi$ :

$$R_\phi(T) \leq 2\|x^*\|_\infty \gamma_T + \delta\frac{\|x^*\|_2^2}{\|x^*\|_\infty} \leq 2\|x^*\|_\infty\gamma_T + \delta\|x^*\|_1$$

where

$$\gamma_t \triangleq \sum_{i=1}^{d}\|g_{1:T,i}\|_2 = \inf_s\left\{\sum_{t=1}^{T}\langle g_t, \mathrm{diag}(-s)^{-1}g_t\rangle : \langle 1, s\rangle \leq \sum_{i=1}^{d}\|g_{1:T,i}\|_2, 0 \leq s\right\}$$

For $x_t$ generated using the composite mirror-descent update (4) and setting $\eta = \|x^*\|_\infty$, we have :

$$R_\phi(T) \leq \sqrt{2}D_\infty \sum_{i=1}^{d}\|g_{1:T,i}\|_2 = \sqrt{2}D_\infty\gamma_T$$

For a sequence generated by the update (4), we assume that $\max_t \|x^* - x_t\|_\infty \leq D_\infty$. Using $\eta = \frac{D_\infty}{\sqrt{2}}$, the following bound holds:

$$R_\phi(T) \leq \sqrt{2d}D_\infty\sqrt{\inf_{s\ ge0, \langle 1\ ,\ s\rangle \leq d}\sum_{t=1}^{T}\|g_t\|^2_{\mathrm{diag}(s)^{-1}}} = \sqrt{2}D_\infty\sum_{i=1}^{d}\|g_{1:T,i}\|_2$$

## 7.4 Results

Now we compare all these different versions of the algorithms with SVM on the MNIST dataset. We had to implement the projected adagrad with the primal dual method and the online mirror descent method on $\ell 1$ ball. We have also implemented these methods with $\ell 1$ regularization. Following this comparison, we set $z = 100$ and $\lambda = 0.01$ (the regularization/penalization setting).

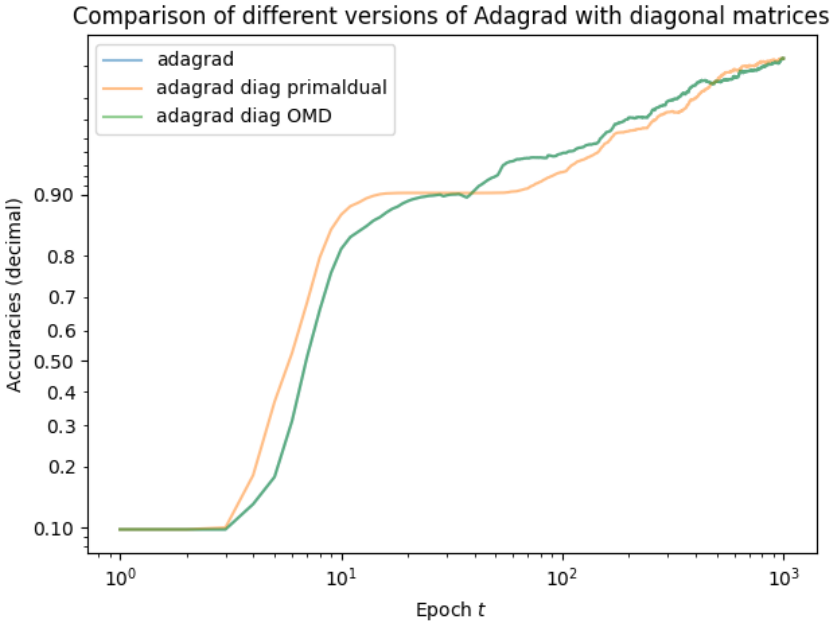


Figure 11: Projected adagrad algorithms with the primal dual and online mirror descent method

As we have shown previously, the curve of the projected adagrad algorithm seen in class overlaps with the online mirror descent method in the case of diagonal version. Adagrad with the primal dual and online mirror descent method has similar performance even though adagrad with primal dual method is slightly better than algorithm with online mirror descent.

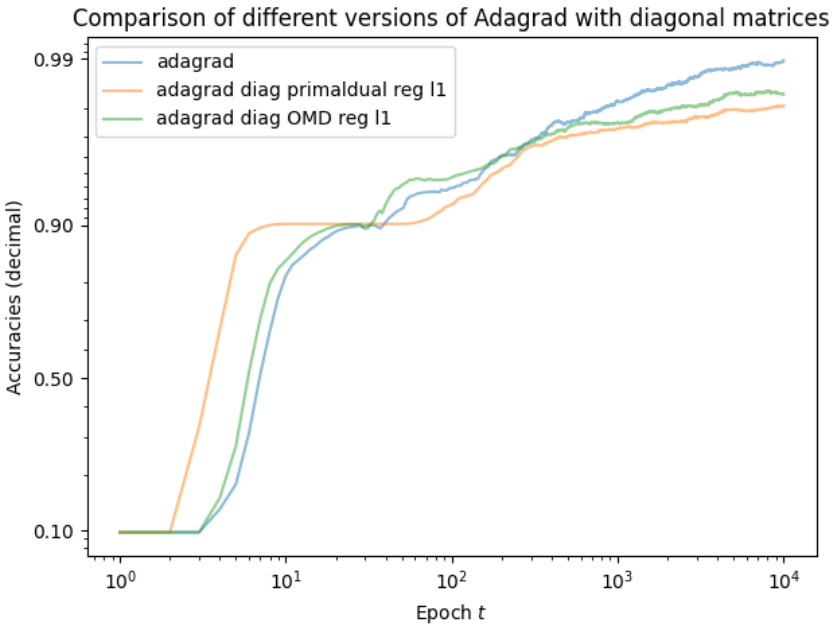


Figure 12: Adagrad algorithms with the primal dual and online mirror descent method with $\ell 1$-regularization

The projected adagrad algorithm and the adagrad with the online mirror descent method and the $\ell 1$ regularization have similar performance. In the first iterations, the adagrad algorithm with the primal dual method and the $\ell 1$ regularization reaches more quickly an accuracy of 0.9 compared to the two other algorithms, but the projected adagrad algorithm seen in class gives better results at the end of

iterations.

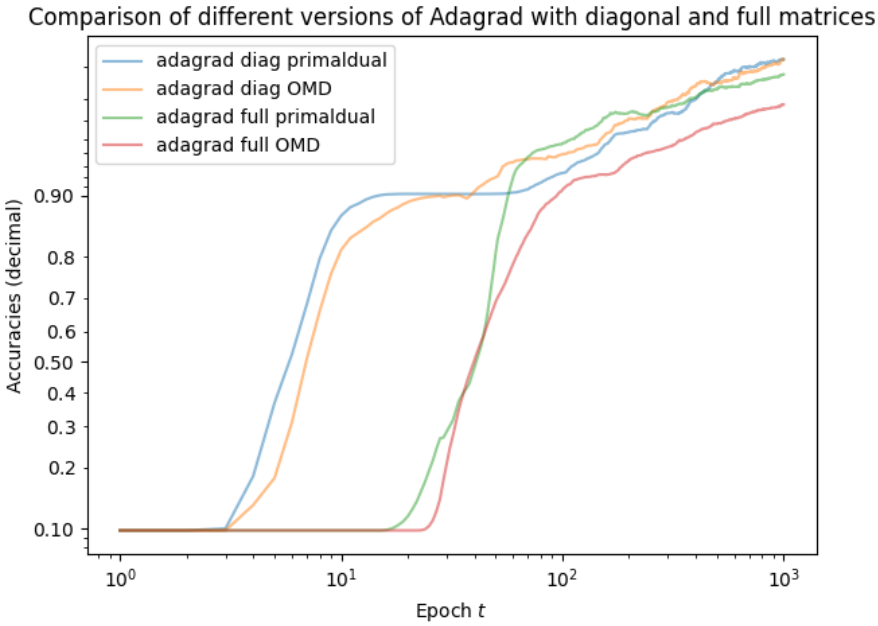


Figure 13: Adagrad algorithms with the primal dual and online mirror descent method with diagonal and full matrix

It is surprising that Adagrad with the diagonal matrix gives better results than the algorithm with full matrix. In our case, the matrix H is not always invertible. So in our version of the algorithm, we take the full matrix if H is invertible otherwise we just take diagonal terms and use the projected Adagrad.

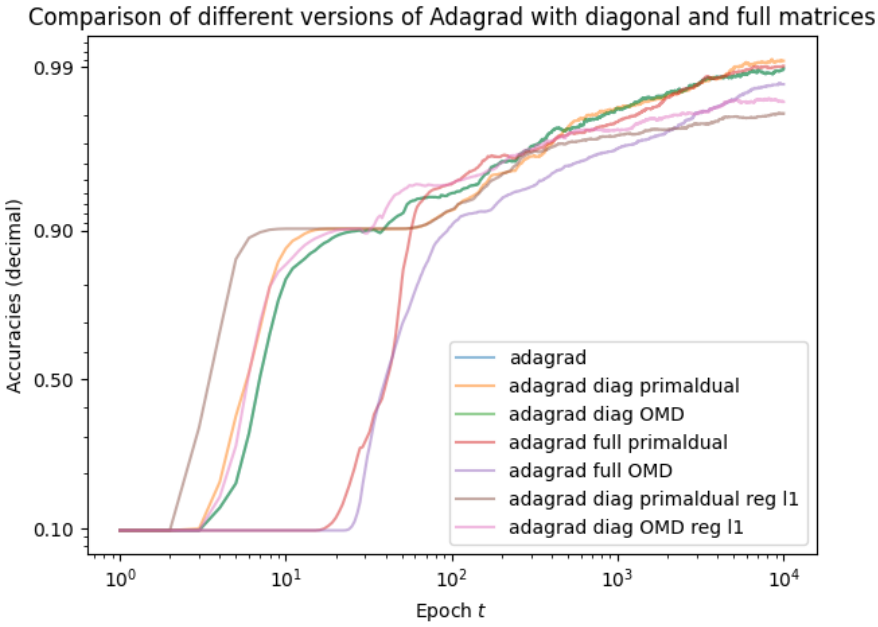


Figure 14: All versions of Adagrad algorithms

In the figure 14, we have plotted all the versions of the Adagrad algorithms. We see that projected Adagrad gives better results than the others algorithms. Primal dual methods and online mirror descent

methods have similar performances even if Adagrad with primal dual methods give slightly better results, but Adagrad with primal dual methods is computationally expensive. Adagrad with the primal dual method takes 12s and 9s with online mirror descent to only update the weights. As our dataset is unbalanced, we cannot come to an exact conclusion. The performance of these algorithms varies according to data. Here we have a matrix of size d = 785 which is larger, so applying Adagrad with full matrix is not a good idea so it is interesting to use diagonal version. Using the diagonal version of Adagrad can lead to better results in terms of the objective function being optimized. The performances are in accordance with the regret bounds established in the article.