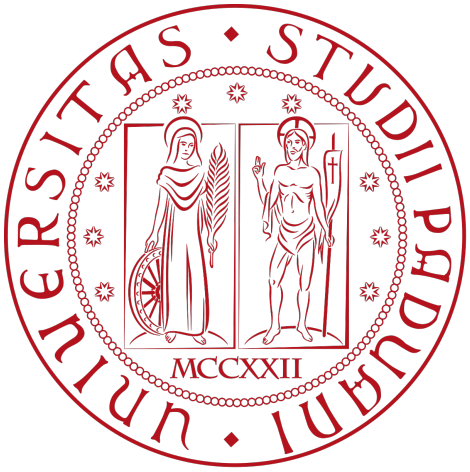


Analysis of Gradient Descent and BCGD methods

Optimization for Data Science
(2021/2022)

Sinigaglia Alberto
ID: 2044712

May 10, 2022

Contents

1. Introduction

On this essay will be presented a problem of semisupervised/transductive learning for binary classification, where we are given very few samples of labeled data, and many samples of unlabeled data, and our goal is to find those missing labels.
To do so, we will use gradient descent methods and BCGD methods to decrease a function, that is going to describe a potential loss given the current configuration.

2. Dataset

2.1. Data

The analysis will be performed on the following datasets:

- A synthetic dataset, generated given 2 or more functions, in order to generate pairs of $(x, f(x))$, on which will be applied some Gaussian noise in order to add some variances
- Deng, L., 2012. The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), pp.141–142.
- Ramachandran, Anil, 2018, "diabetes.txt", Binary dataset, `https://doi.org/10.7910/DVN/LRNLZV/ZIOATY`, Harvard Dataverse, V1
- KOKLU, M. and OZKAN, I.A., (2020), Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques. Computers and Electronics in Agriculture, 174, 105507, `https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset`.

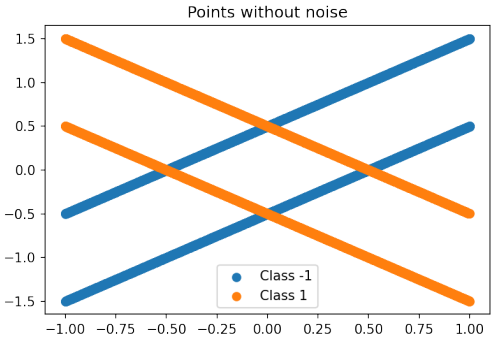
In addition to this, there is a *spiral-like* dataset used to generate the animations, which will be not analyzed.
For this analysis, all the images shown will come from a sample of 10.000 elements of the synthetic dataset, where only 1% of them will be labeled, the rest will be initialized to 0, from the following functions:

- Class -1:

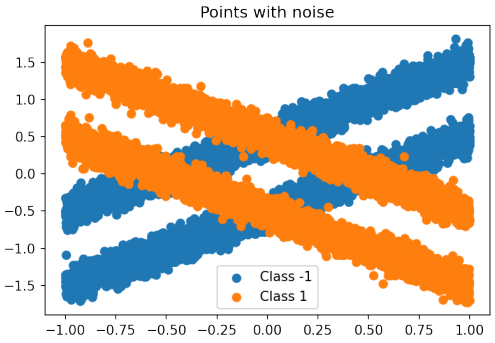
$$\{(x_i, f(x_i)) : f(x) = x + 0.5, i \in [0, 2500]\}$$
$$\cup$$
$$\{(x_i, f(x_i)) : f(x) = x - 0.5, i \in [0, 2500]\}$$

- Class 1:

$$\{(x_i, f(x_i)) : f(x) = -x + 0.5, i \in [0, 2500]\}$$
$$\cup$$
$$\{(x_i, f(x_i)) : f(x) = -x - 0.5, i \in [0, 2500]\}$$



On those points, will then be applied a noise sampled from $N(0, 0.1)$



2.2. Points generation

In order to have more control over the results of the analysis to have more precise results, the dataset is a synthentic one generated algorithmically.
To do so, it's been used the following algorithm:

1. create 2 or more ranges on which the clusters will be generated (for example $[0, 1]$)
2. create a function for each range, which will define the second coordinate (for example $f(x) = x^2$)
3. now that we have the 2 clusters, in order to have some variance over that, generate noise from a Gaussian distribution (for example $N(0, 1)$)
4. apply the generated noise either on the x or on the $f(x)$ of the points

2.3. Preprocessing

As reported later, the loss function will depend on some weights, which should be a sort of similarity weight, depending on the distance of two points; for this reason, this model is very sensitive to the *curse of dimensionality* problem, in particular when dealing with high dimensional data, with not many samples.
For this reason, except for the synthetic one, the datasets dimensionality will be reduced using Principal Component Analysis, in particular:

- Diabetes dataset passes from 9 dimensions to 2
- Dry Beans dataset passes from 16 dimensions to 5
- MNIST Digits passes from 784 dimensions to 5

Futhermore, since the weights will be based on the distance, the data from every dataset is also been min-max-scaled and standardized, in order to have a nice distribution near $\vec{0}$.

3. Formalization

3.1. Loss

The loss function were given, and it goes as follows:

$$f(y) = \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}(y^j - \bar{y}^i)^2 + \frac{1}{2}\sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(y^j - y^i)^2$$

Following are the meaning of each term:

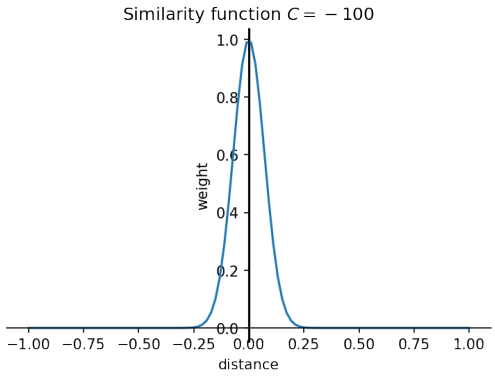
- y^j is the current label assigned to the j-th unlabeled element
- $\bar{y}^j$ is the label of the j-th labeled element
- w_{ij} is the weight that associates the i-th labeled element with the j-th unlabeled element
- $\bar{w}_{ij}$ is the weight that associates the i-th unlabeled element with the j-th unlabeled element

The meaning of this loss function comes entirely from the weights; they are supposed to be inversely proportional to the distance, and therefore the algorithms will try to make the label from each unlabeled point to any other point the more similar the closer they are, in order to try to make $y^i - y^j = 0$ and therefore avoiding to make the w_{ij} weight counts.

3.2. Weights

As said before, the weights are a similarity coefficient, and therefore the higher a weight is, the more the two points it represent should be similar.
To do so, the following similarity function is been chosen:

$$dist(\vec{a}, \vec{b}) = e^{-C||\vec{a}-\vec{b}||_2^2}$$



This similarity function has a hyperparameter, which should be chosen based on how far a point should be considered relevant for the evaluation of a current one.
For the entire analysis, C will be 100

3.3. Gradient

Following is the gradient of the loss function in respect of a generic y^j:

$$\begin{aligned}\nabla_{y^j} f(y) &= 2[\sum_{i=0}^{l} w_{ij}(y^j - \bar{y}^i) + \sum_{i=0}^{u} \bar{w}_{ij}(y^j - y^i)] \\ &= 2[(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \bar{w}_{ij})y^j - \sum_{i=0}^{l} w_{ij}\bar{y}^i - \sum_{i=0}^{u} \bar{w}_{ij}y^i]\end{aligned}$$

The second formulation will be used in the associated Jupyter Notebook to calculate the gradient, in order to make it as effient as possible; in fact, the first coefficient is a constant independent from the y^j we are considering, and can be calculated just once, the second term is also a coefficient independent from the y^j we are considering, and so it can also be calculated just once, and the third element is the same for every element of the gradient, and so can be calculated only once at each calculation of the entire gradient. This new formulation brings the complexity for the calculation of the entire gradient from $O(u * (l + u))$ to just $O(u)$.

3.4. Accuracy

For the evaluation of the algorithms, it's necessary to define a measure of accuracy, however no optimal formula was found for this, but the following were tested:

- Rounding: considering every $y^j >= 0$ as 1 and the remaining as -1, however this had the problem that after even a small step in the correct direction, would consider that label as correctly classified, and so the tracking of the accuracy becomes almost constant after very few iterations of any of the methods
- Loss ratio: using $1 - \frac{\text{current loss}}{\text{max loss}}$ estimating a priori the maximum of the loss, however has the problem that the loss might have a minimum different from 0, and so even if the algorithm classifies everything correctly, the accuracy would not be 100%
- Norm ratio: calculated as $1 - \frac{||\text{target Y} - \text{current Y}||_1}{\text{initial norm}}$, however this also has the problem of not taking in account that the minimum of the loss might have y^j very small, and therefore the methods will almost never converges to 100% accuracy

For the convergence of the algorithms, will be used the third one, in order to have a better understanding the speed of convergence, which should be also observed on the loss; however, at the end the first one will be used to evaluate the final classification performance.

3.5. Hessian

with: $k \neq j \rightarrow \nabla_{y^j y^k} f(j) = -2\bar{w}_{kj}$

with: $k = j \rightarrow \nabla_{y^j y^j} f(j) = 2[(\sum_{i=0}^{l} w_{ij}) + (\sum_{i=0}^{u} \bar{w}_{ij}) - \bar{w}_{jj}]$

$$Hessian_{n\times n} =$$

$$2 \cdot \begin{bmatrix} (\sum_{i=0}^{l} w_{i1}) + (\sum_{i=0}^{u} \bar{w}_{i1}) - \bar{w}_{11} & -\bar{w}_{12} & \cdots & -\bar{w}_{1n} \\ -\bar{w}_{21} & (\sum_{i=0}^{l} w_{i2}) + (\sum_{i=0}^{u} \bar{w}_{i2}) - \bar{w}_{22} & \cdots & -\bar{w}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\bar{w}_{n1} & -\bar{w}_{n2} & \cdots & (\sum_{i=0}^{l} w_{in}) + (\sum_{i=0}^{u} \bar{w}_{in}) - \bar{w}_{nn} \end{bmatrix}$$

For the analysis, to evaluate the step-size to take, will be used the Lipschitz constant. To calculate it, we will use the maximum eigenvalue of the Hessian matrix of the loss function, since it's constant.
Futhermore, the Hessian will be used also to estimate σ to check if the problem is strongly convex or not.

$$\exists \sigma > 0 \text{ s.t. } f(x) - \frac{\sigma}{2}||x||^2 \text{ is convex}$$

3.6. Step-size

For the entire analysis, the fixed step size will be used. In order to have a safe step size, at the beginning will be calculated the Lipschitz constant L, which, since our Hessian is positive definite and constant, it's independent from the data and therefore is the same for the whole process, and it's equal to the highest eigenvalue of the Hessian matrix.
However, at the end of the analysis, other 3 alternatives will be considered:

- an optimized version of the fixed step size using a better estimate of the Lipschitz constant for the BCGD methods
- exact line search for the BCGD methods
- exact line search for gradient descent

4. Algorithms

The analysis will cover the following methods:

- Gradient Descent
- Gradient Descent with improved rate (in case of strong convexity)
- Heavy Ball
- Accelerated Gradient
- BCGM Random
- BCGM Cyclic
- BCGM Gauss-Southwell

Each of them will be run for 100 iterations (for BCGM, one iteration will be considered $|unlabeled\ points|$ updates), with a possible early stopping if $||\nabla f(x)|| < 10^{-6}$.
As mentioned before, the step-size for each of them will be calculated using the Lipschitz constant, and for Heavy ball and Accelerated gradient, the β parameter will be estimated as follows:

$$\lambda_0 = 1$$
$$x_0 = x_1$$
$$\lambda_k = \frac{1 + \sqrt{1 + 4\lambda_{k-1}^2}}{2}$$
$$\beta_k = \frac{\lambda_{k-1} - 1}{\lambda_k}$$

For the BCGM the block size is fixed at 1.
Since the loss function is convex, there is no much sense to consider the solution founded by all the algorithms, since they will all converge to the same one.
However, for each of them, will be compared CPU time and iterations against accuracy and loss.

4.1. Gradient Descent

Gradient descent is been implemented in the most basic version, in order to have a benchmark for all the subsequent algorithms:

$$y_k = y_{k-1} - \frac{1}{L}\nabla f(y_{k-1})$$

4.2. Gradient Descent with improved rate

In case of strong convexity, we are assured convergence using $\alpha_k = \frac{2}{\sigma + L}$ instead of just $\alpha_k = \frac{1}{L}$, which depending

on σ might have a step-size up to two time than the previous one:

$$y_k = y_{k-1} - \frac{2}{\sigma + L}\nabla f(y_{k-1})$$

4.3. Heavy Ball

This algorithms uses a momentum term based on the previous iteration, in order to tends to follow the directions that don't changes often:

$$y_k = y_{k-1} - \frac{1}{L}\nabla f(y_{k-1}) + \beta_k(y_{k-1} - y_{k-2})$$

4.4. Accelerated Gradient

In this algorithm we make use of an additional step forward to predict sudden changes of the gradient:

$$p_k = y_{k-1} + \beta_k(y_{k-1} - y_{k-2})$$
$$y_k = p_k - \frac{1}{L}\nabla f(p_k)$$

4.5. BCGM Random

For this algorithm, to calculate the update, is been used the gradient formula, evaluated on just one single y^j:

$$J \sim \mathcal{U}\{0, |\text{unlabeled points}|\}$$
$$y_k^J = y_{k-1}^J - \frac{1}{L}\nabla_{y^J} f(y_{k-1}^J)$$

4.6. BCGM Cyclic

Also for this BCGM the block size is 1, therefore the updates is performed for each variable, given an arbitrary order, one at a time:

$$y_k^j = y_{k-1}^j - \frac{1}{L}\nabla_{y^j} f(y_{k-1}^j)$$
$$\forall j \in \{0, ..., |\text{unlabeled points}|\}$$

4.7. BCGM Gauss-Southwell

For this algorithm, the index is chosen on the block on which the gradient is higher. However, unlike for the previous BCGM algorithms, this requires to estimate the full gradient to evaluate the decision rule, therefore it's mandatory to find an efficient way to update the gradient at each iteration.
To do so, following are the calculations done to perform it, using the following constant in order to keep the calculations clearer.

$$c_j = (\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \bar{w}_{ij})$$
$$b = \sum_{i=0}^{l} w_{ij}\bar{y}^i$$

Supposing update on y^j:

$$\nabla_{y_{k+1}^j} f(y_{k+1}) = 2[c^j y_{k+1}^j - b - \sum_{i=0}^{u} \bar{w}_{ij} y_{k+1}^i]$$

Expanding the last summation:

$$\nabla_{y_{k+1}^j} f(y_{k+1}) = 2[c^j y_{k+1}^j - b - (\bar{w}_{0j} y_{k+1}^0 + ... + \bar{w}_{jj} y_{k+1}^j + ... + \bar{w}_{uj} y_{k+1}^u)]$$

However we have that:

$$y_{k+1}^i = y_k^i \forall i \in \{1, ..., |unlabeledpoints|\}\backslash\{j\}$$
$$y_{k+1}^j = y_k^j - \frac{1}{L}\nabla_{y^j} f(y_k^j)$$

Therefore we can rewrite the previous formula in the following way:

$$\nabla_{y_{k+1}^j} f(y_{k+1}) = 2[c^j(y_k^j - \frac{1}{L}\nabla_{y^j} f(y_k^j)) - b - (\bar{w}_{0j} y_k^0 + ... + \bar{w}_{jj}(y_k^j - \frac{1}{L}\nabla_{y^j} f(y_k^j)) + ... + \bar{w}_{uj} y_k^u)]$$

Multiplying and rearranging the formula we get:

$$\nabla_{y_{k+1}^j} f(y_{k+1}) = 2[c^j y_k^j - b - \sum_{i=0}^{u} \bar{w}_{ij} y_k^i] + 2[-c^j \frac{1}{L}\nabla_{y^j} f(y_k^j) + \bar{w}_{jj}\frac{1}{L}\nabla_{y^j} f(y_k^j)]$$

Which is equivalent to:

$$\nabla_{y_{k+1}^j} f(y_{k+1}) = \nabla_{y_k^j} f(y_k) + 2[(\bar{w}_{jj} - c^j)(\frac{1}{L}\nabla_{y^j} f(y_k^j))]$$

For all the other variable, the update is even simpler (considering $i \in \{1, ..., |\text{unlabeled points}|\}\backslash\{j\}$):

$$\nabla_{y_{k+1}^i} f(y_{k+1}) = 2[c^i y_{k+1}^i - b] + 2[-(\bar{w}_{0j} y_{k+1}^0 + ... + \bar{w}_{jj} y_{k+1}^j + ... + \bar{w}_{uj} y_{k+1}^u)]$$

However, as said before, all the elements except y^j has not changed, and so it's equivalent to:

$$\nabla_{y_{k+1}^i} f(y_{k+1}) = 2[c^i y_k^i - b - (\bar{w}_{0i} y_k^0 + ... + \bar{w}_{ji}(y_k^j - \frac{1}{L}\nabla_{y^j} f(y_k^j)) + ... + \bar{w}_{ui} y_k^u)]$$

Which multiplying and rearranging the terms, can be rewritten as:

$$\nabla_{y^i_{k+1}} f(y_{k+1}) = 2[c^i y^i_k - b - \sum_{i=0}^{u} \bar{w}_{ij} y^i_k] + 2[\bar{w}_{ji} \frac{1}{L} \nabla_{y^j} f(y^j_k)]$$

And therefore we have:

$$\nabla_{y^i_{k+1}} f(y_{k+1}) = \nabla_{y^i_k} f(y_k) + 2[\bar{w}_{ji} \frac{1}{L} \nabla_{y^j} f(y^j_k)]$$

Summarizing, supposing $\nabla_{y_k} f(y)$ given, we can update it to get $\nabla_{y_{k+1}} f(y)$ in the following way:

$$\nabla_{y^j_{k+1}} f(y_{k+1}) = \nabla_{y^j_k} f(y_k) + 2[(\bar{w}_{jj} - c^j)(\frac{1}{L} \nabla_{y^j} f(y^j_k))]$$
$$\nabla_{y^i_{k+1}} f(y_{k+1}) = \nabla_{y^i_k} f(y_k) + 2[\bar{w}_{ji} \frac{1}{L} \nabla_{y^j} f(y^j_k)]x$$

Since we have an efficient way to update the gradient with block of size 1, the algorithm follows:

$$j = \underset{i \in [1,u]}{\text{Argmax}} ||\nabla_i f(y)||$$
$$y^j_k = y^j_{k-1} - \frac{1}{L} \nabla_{y^j} f(y^j_{k-1})$$

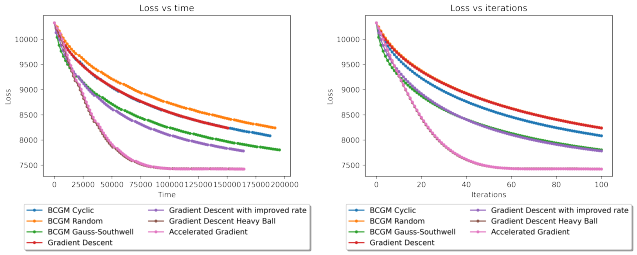
5. Comparison

None of the algorithm hit the early sopping before reaching the maximum number of iterations allowed, so all the following images reports the same number of iterations.

5.1. Loss

5.1.1 General comparison

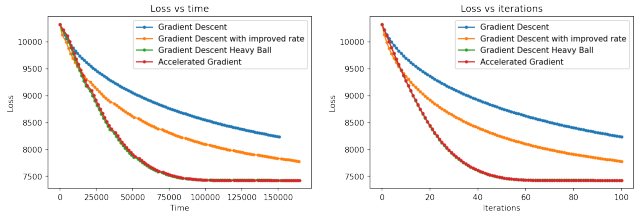
Following is the graph of the loss of all the algorithms:



Clearly, BCGD Random had the same convergence rate as gradient descent, however a full iteration of random cost more than one iteration of gradient descent.
Same holds for all the BCGM, and it's due to the fact that the full gradient costs only $O(u)$, therefore it's pretty efficient to be calculated.

5.1.2 Full gradient algorithms comparison

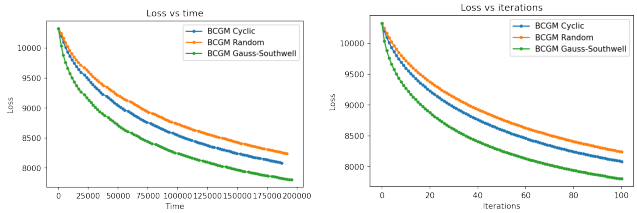
Following is the graph of the loss of the algorithms that use the full gradient as update:



The two methods that uses the momentum clearly outrun the others, and the improved step-size helped a lot the naive gradient descent to converge.

5.1.3 BCGM algorithms comparison

Following is the graph of the loss of the BCGD methods:

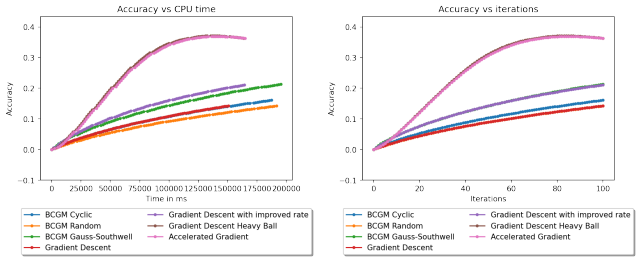


The Gauss-Southweel approach beats all the others, since it uses first order information to perform the step, where the other two no.

5.2. Accuracy

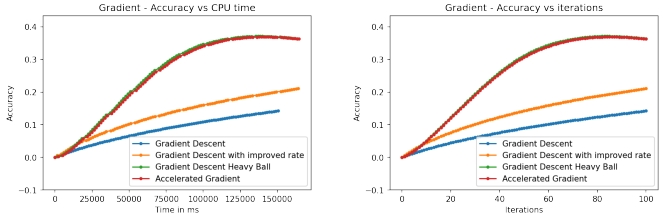
5.2.1 General comparison

Following is the graph of the accuracy of all the algorithms:



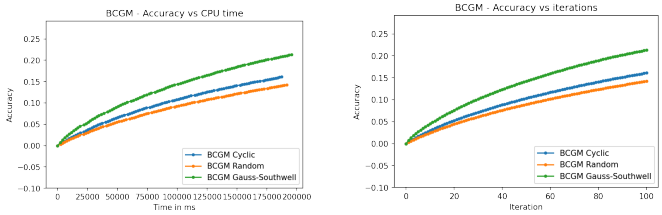
5.2.2 Full gradient algorithms comparison

Following is the graph of the accuracy of the algorithms that use the full gradient as update:



5.2.3 BCGM algorithms comparison

Following is the graph of the accuracy of the BCGD methods:

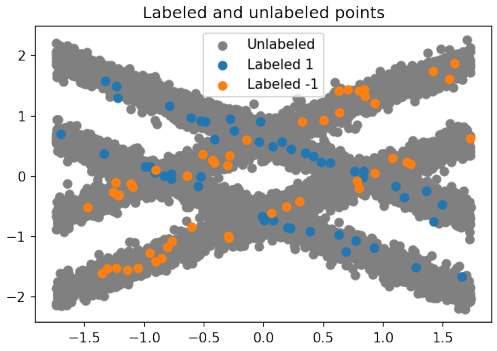


5.3. Discussion

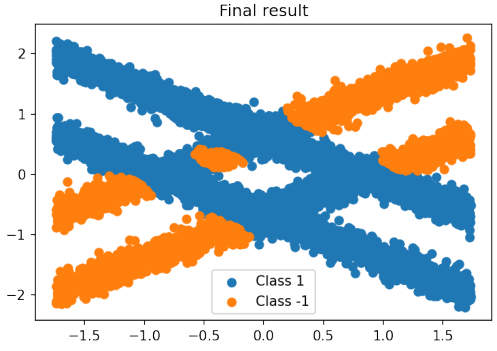
Clearly, the two methods based on momentum outperformed all the others. However, worth noting how the convergence of Gradient descent changes in case of strong convexity, using the improved step.
On the other hands, both the BCGM cyclic version and the Gauss Southwell version, outperformed the Gradient Descent in accuracy and loss, using the same step-size.

6. Results

All the results shown, as previously said, comes from a synthetic dataset composed by 10.000 points, of which only 1% is labeled. Those points can be seen in the following image:



All the algorithms, if we round the labels in $-1, 1$, converged to the same following configuration:

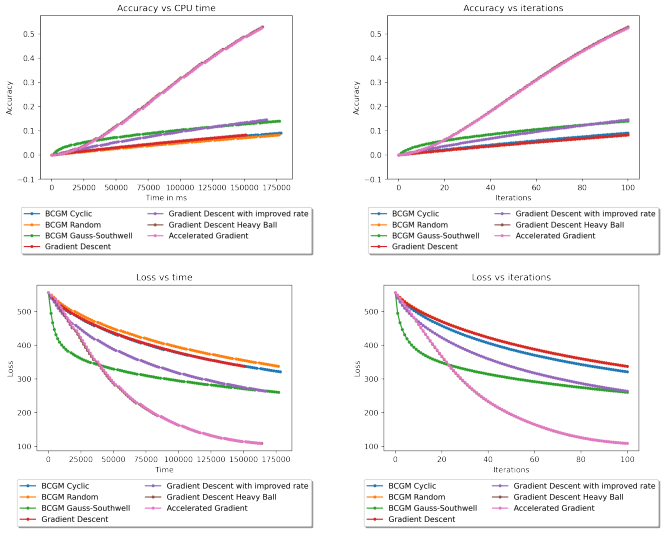


7. Dataset results

As mentioned before, the model is being tested also on publicly available datasets, and following are the results.

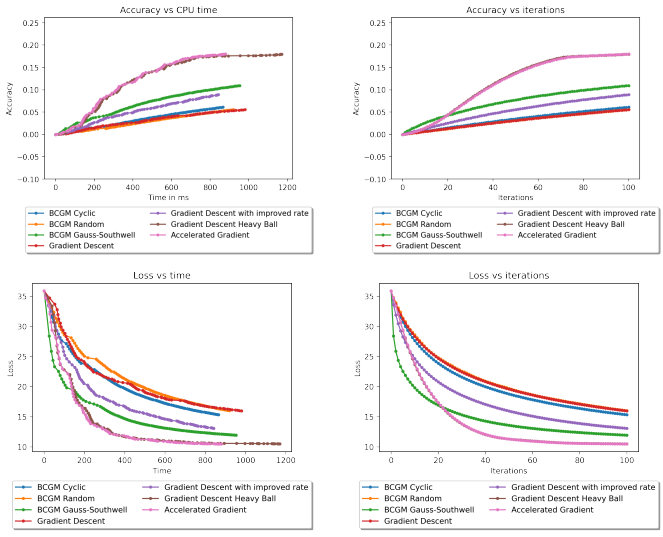
7.1. MNIST Digits

With this dataset, in order to make it a binary OVR classification problem, it's been tested the ability to discriminate 0 against all the other digits.
This dataset has the peculiarity to be very ill-formed, in fact the estimated Lipschitz constant is 54.3, but σ is $7.56 \cdot 10^{-7}$.
However this won't effect the momentum based algorithm, and in fact the difference is noticeable:



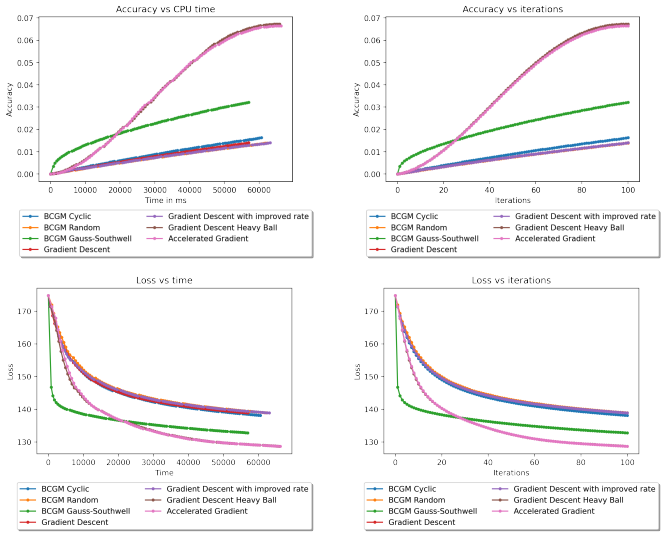
7.2. Diabetes

This dataset is relatively small, and there are only 268 samples per class, therefore the ratio of labeled data is been increased to 10%, and has no problem of ill-formation.



7.3. Beans

For this dataset, the two classes chosen to be classified using OVO are "SIRA" and "DERMASON"



8. Model accuracy

As conclusion it's been tested the accuracy of the model, even though it's not relevant for the comparison of the algorithms, since they all converges to the same solution.
In particular, to get those results comes from the following setting:

- using the MNIST dataset
- using Accelerated Gradient as algorithm
- Accuracy evaluated on OVR experiments
- 5000 elements of the current digit, 5000 elements of other digits
- accuracy calculated using the rounding method described before

Following are the results:

Digit	Accuracy 1% labeled	Accuracy 10% labeled
0	95.57%	95.98%
1	87.26%	95.77%
2	86.63%	88.96%
3	85.53%	87.92%
4	81.89%	88.24%
5	82.90%	88.77%
6	86.08%	90.06%
7	84.28%	88.68%
8	83.17%	87.16%
9	82.34%	87.07%

Table 1. Results of OVR on MNIST digits

9. Variants

9.1. Nesterov's distribution for BCGM random

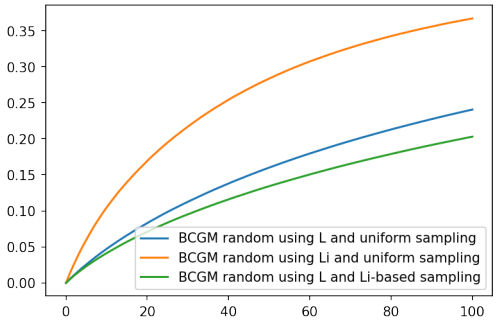
In this case, in the BCGM random algorithm, we change how we sample the index, where previously was:

$$J \sim \mathcal{U}\{0, |\text{unlabeled points}|\}$$

And so each index has the same probability of being drown, to a new one based on the L_i:

$$P(i_k = i) = \frac{L_i}{\sum_{j=0}^{b} Lj}$$

However, in this case, the change did not helped:



9.2. Optimized Lipschitz constant for BCGM

For all the previous results, all the methods had the same step-size $\frac{1}{L}$, since that constant guarantees convergence even when $\frac{L}{\sigma}$ is very big.
In other words, it's a "safe" step-size, since when we use the full gradient to update the y, we have to make the same "step size" in all the directions, indipendently on the steepness of the directions, which might vary a lot.
However, when dealing with the single block of size 1, this is no more true, and so we might try to do better; in particular, considering our problem, it's a quadratic function, and dealing with blocks of size 1, makes it univariate, of which the Hessian is just the diagonal of the Hessian of the original problem. As seen before, that Hessian is constant, and therefore also it's diagonal is constant, in fact, supposing to update a generic y^j:

$$f(y^j)'' = 2[(\sum_{i=0}^{l} w_{ij}) + (\sum_{i=0}^{u} \bar{w}_{ij}) - \bar{w}_{jj}]$$

This means that when we are dealing with just one block, the Lipschitz constant of that block is the maximum eigenvalue of a 1×1 matrix, which is the element itself, and furthermore, also σ is that eigenvalue, and therefore (noting

L_j as the Lipschitz constant for the variable j):

$$L_j = 2[(\sum_{i=0}^{l} w_{ij}) + (\sum_{i=0}^{u} \bar{w}_{ij}) - \bar{w}_{jj}]$$

$$sigma = 2[(\sum_{i=0}^{l} w_{ij}) + (\sum_{i=0}^{u} \bar{w}_{ij}) - \bar{w}_{jj}]$$

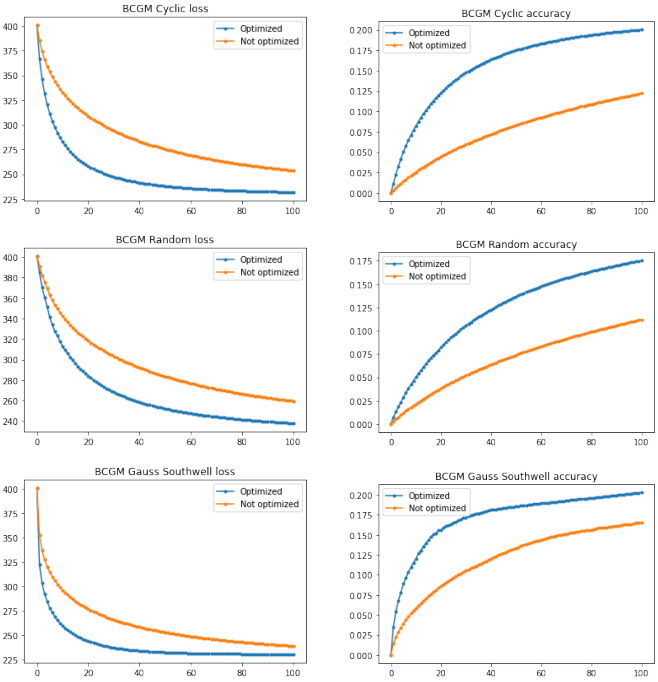
Therefore L and σ are the same, supposing $(\sum_{i=0}^{l} w_{ij}) + (\sum_{i=0}^{u} \bar{w}_{ij}) - \bar{w}_{jj} > 0$, we are dealing with a strongly convex univariate function, and by using the improved rate for strongly convex functions we have:

$$\begin{aligned}\alpha_j &= \frac{2}{L_j + \sigma_j} \\ &= \frac{2}{2L_j} \\ &= \frac{1}{L_j} \\ &= \frac{1}{2[(\sum_{i=0}^{l} w_{ij}) + (\sum_{i=0}^{u} \bar{w}_{ij}) - \bar{w}_{jj}]}\end{aligned}$$

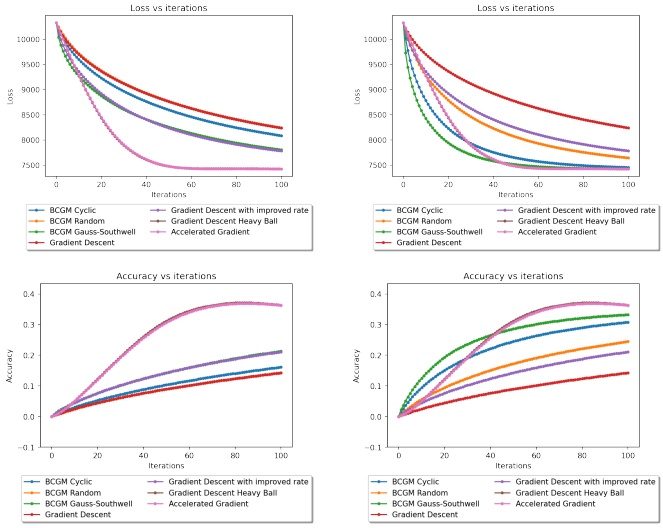
Now we have a better estimate of a safe step-size for our BCGM problem.

9.2.1 Comparison

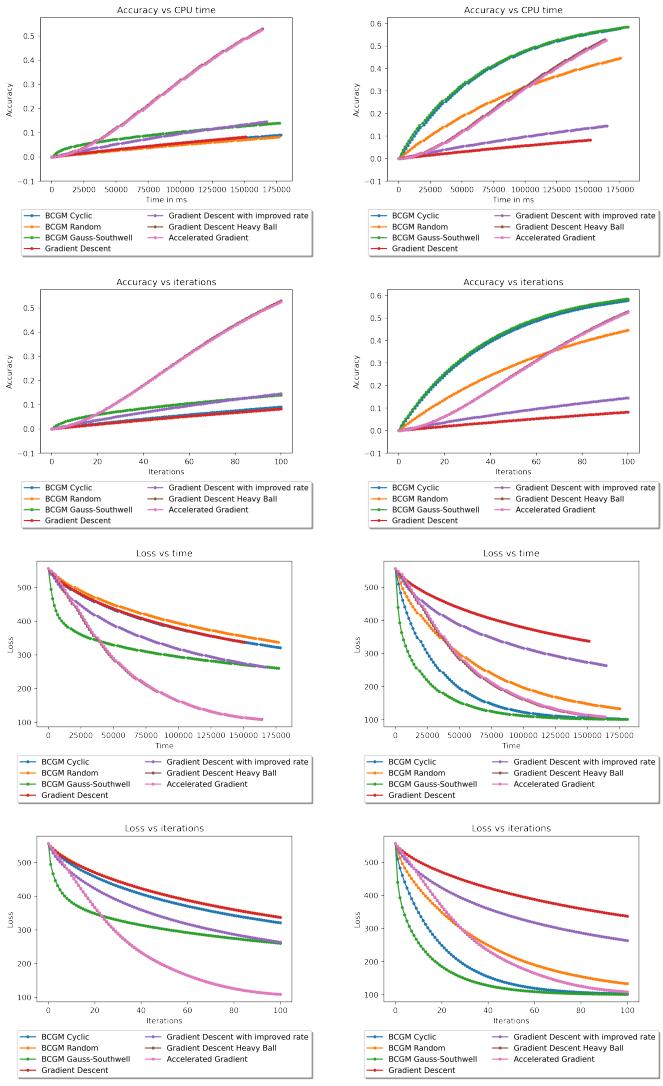
The three following images compare the loss and accuracy of the same method with the 2 step-sizes:



And following is the general comparison of loss and accuracy with and without the optimization:



However, it's even more noticeable in the MNIST digits dataset, which which is ill formed since $L >> \sigma$.



9.3. Exact line search for BCGM

The aim of this section is to exploit the structure of the gradient of the single variable in order to develop an exact line search for the BCGM.
Recalling the gradient:

$$\nabla_{y^j} f(y) = 2[\sum_{i=0}^{l} w_{ij}(y^j - \bar{y}^i) + \sum_{i=0}^{u} \bar{w}_{ij}(y^j - y^i)]$$
$$= 2[(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \bar{w}_{ij})y^j - \sum_{i=0}^{l} w_{ij}\bar{y}^i - \sum_{i=0}^{u} \bar{w}_{ij}y^i]$$

As seen before, the first coefficient and the second are constant, so we can rewrite them in the following way

$$c^j = (\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \bar{w}_{ij})$$
$$b = \sum_{i=0}^{l} w_{ij}\bar{y}^i$$

Resulting in the following formula:

$$\nabla_{y^j_{k+1}} f(y_{k+1}) = 2[c^j y^j_{k+1} - b - \sum_{i=0}^{u} \bar{w}_{ij} y^i_{k+1}]$$

The goal is now to find a step size that allows this gradient to be 0.
However, as previously said:

$$y^i_{k+1} = y^i_k \forall i \in \{1, ..., |unlabeledpoints|\}\backslash\{j\}$$
$$y^j_{k+1} = y^j_k - \alpha_k \nabla_{y^j} f(y^j_k)$$

Therefore we can rewrite the previous formula in the following way:

$$\nabla_{y^j_{k+1}} f(y_{k+1}) = 2[c^j(y^j_k - \alpha_k \nabla_{y^j} f(y^j_k)) - b$$
$$- (\bar{w}_{0j} y^0_k + ... + \bar{w}_{jj}(y^j_k - \alpha_k \nabla_{y^j} f(y^j_k)) + ... + \bar{w}_{uj} y^u_k)]$$

Multiplying and rearranging the formula we get:

$$\nabla_{y^j_{k+1}} f(y_{k+1}) = 2[c^j y^j_k - b - \sum_{i=0}^{u} \bar{w}_{ij} y^i_k]$$
$$+ 2[-c^j \alpha_k \nabla_{y^j} f(y^j_k) + \bar{w}_{jj} \alpha_k \nabla_{y^j} f(y^j_k)]$$

$$\nabla_{y^j_{k+1}} f(y_{k+1}) = \nabla_{y^j_k} f(y_k) + 2[(\bar{w}_{jj} - c^j)(\alpha_k \nabla_{y^j} f(y^j_k))]$$

Since the aim is to find the minimum, then the new gradient should be zero:

$$\nabla_{y^j_{k+1}} f(y_{k+1}) = 0$$

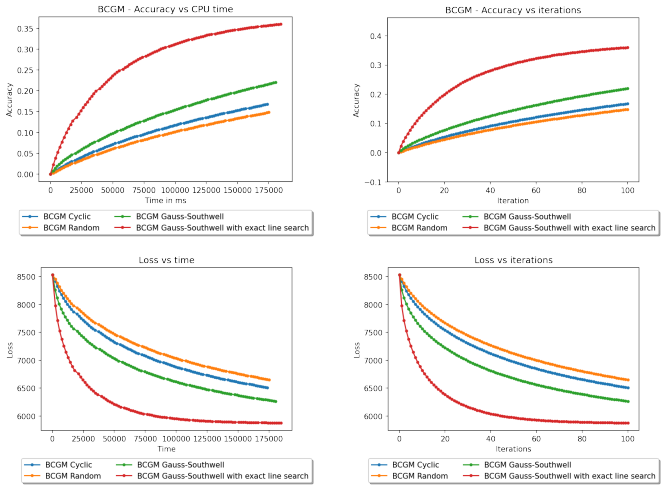
Therefore:

$$\nabla_{y^j_k} f(y_k) + 2[(\bar{w}_{jj} - c^j)(\alpha_k \nabla_{y^j} f(y^j_k))] = 0$$
$$2[(\bar{w}_{jj} - c^j)\alpha_k \nabla_{y^j} f(y^j_k)] = -\nabla_{y^j_k} f(y_k)$$
$$\alpha_k = \frac{-\nabla_{y^j_k} f(y_k)}{2\nabla_{y^j} f(y^j_k)(\bar{w}_{jj} - c^j)}$$
$$\alpha_k = \frac{-1}{2(\bar{w}_{jj} - c^j)}$$

However, if we now come back to the original value of the constants, the result is the following:

$$\alpha_k = \frac{-1}{2[\bar{w}_{jj} - (\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \bar{w}_{ij})]}$$
$$= \frac{1}{2(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \bar{w}_{ij} - \bar{w}_{jj})}$$

This is exactly the same result reported on the previous section, on the improved step-size of the Lipschitz constant for the BCGM, and therefore the result/graphs are the same, but can be easily summarized in the following ones:



9.4. Exact line search for Gradient Descent

Exact line search in case of Gradient descent means finding a direction on which we are going to optimize, and the best step size that can be performed.
Formally, denoting with d the gradient at iteration k, and α the best step size that we want to find at iteration k, we need to solve the following:

$$\alpha_k = \underset{\alpha}{argmin} f(y + \alpha d)$$

Following are the steps to find α_k:

Given that: $f(y) = \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}(y^j - \bar{y}^i)^2 + \frac{1}{2}\sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(y^j - y^i)^2 \Rightarrow$

$$f(y+\alpha d) = \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}(y^j + d^j\alpha - \bar{y}^i)^2 + \frac{1}{2}\sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(y^j + d^j\alpha - y^i - d^i\alpha)^2$$

$$f'(y+\alpha d) = \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}d^j 2(y^j + d^j\alpha - \bar{y}^i) + \frac{1}{2}\sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)2(y^j - y^i + (d^j - d^i)\alpha)$$

$$f'(y+\alpha d) = \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}d^j 2(y^j - \bar{y}^i) + \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}d^j 2d^j\alpha + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)(y^j - y^i) + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)(d^j - d^i)\alpha$$

$$f'(y+\alpha d) = \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}d^j 2(y^j - \bar{y}^i) + \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}(d^j)^2 2\alpha + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)(y^j - y^i) + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)^2\alpha$$

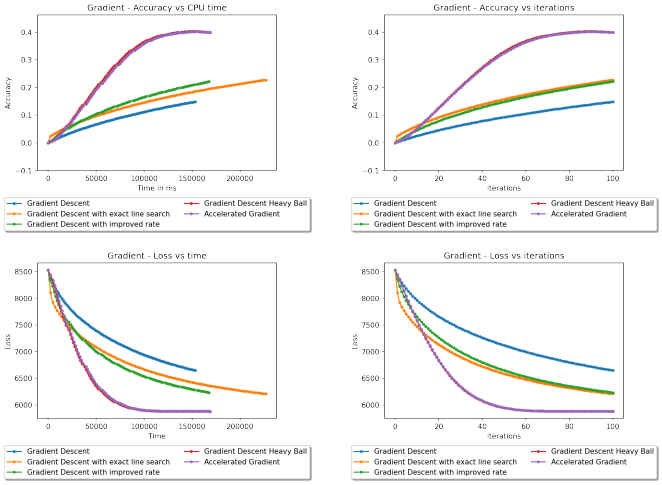
However we want: $\alpha_k = \underset{\alpha}{argmin} f(y+\alpha d) \leftrightarrow f'(y+\alpha d) = 0$

$$\sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}d^j 2(y^j - \bar{y}^i) + \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}(d^j)^2 2\alpha + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)(y^j - y^i) + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)^2\alpha = 0$$

$$(\sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}(d^j)^2 2 + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)^2)\alpha = -(\sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}d^j 2(y^j - \bar{y}^i) + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)(y^j - y^i))$$

$$\alpha = \frac{-(2\sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}d^j(y^j - \bar{y}^i) + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)(y^j - y^i))}{2\sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}(d^j)^2 + \sum_{i=0}^{u}\sum_{j=0}^{u} \bar{w}_{ij}(d^j - d^i)^2}$$

Using this step size, using the gradient as direction, we perform a step with length exactly long as is needed to get the minimum point of the loss function in that direction.
Following are the results on the synthetic dataset of the gradient descent with the exact step size, compared to all the other "full gradient" methods analyzed previously:



Clearly, the exact line search beats both the naive gradient descent using $\frac{1}{L}$ as step size and the improved one using $\frac{2}{L+\sigma}$ as step size, if we fix the number of iterations.
On the other hand however, the calculation of the exact step size is very costly compared to the other evaluation of the step size, so the improvement in precision, is wasted when we count the time it takes, in fact as we can see from the graphs, comparing the losses vs time, it's faster to use the improved step size instead of the exact line search, since it gives good performance, is way less time.