Master IASD : Intelligence Artificielle, Systèmes, Données

Optimization for ML Project

Submitted by

Montassar Mhamdi

Supervised by

Gabriel Peyré
Mathieu Blondel
Clément Royer
Vincent Duval
Irene Waldspurger

15 January 2023

Contents

1 Context **1**

2 Exploratory Data Analysis **2**

2.1 About DataSet 2

2.2 Visualization and Observations 3

3 Gradient Descent **6**

3.1 Gradient Descent for Classification with Logistic Loss 6

3.2 Step Size 6

3.3 Convergence Rate 7

3.4 Performace on Test Set (change the RIDGE penalty) 8

4 Automatic Differentiation **9**

4.1 Elementary Functions 9

4.2 MLP with K Layers 9

4.3 Evalute and Check the Chain Calculus 10

4.4 Implement Training with Stochastic Gradient Algorithm . . . 10

5 Stochastic Batch Gradient **12**

5.1 SGD and Comparison with GD 12

5.2 Optimal Batch Size 13

5.3 Learning Rate Impact 13

5.4 Advanced Variant of SG : SVRG 13

6 Convexity and Constrained Optimization **15**

6.1 Conditional Gradient Descent 15

6.2 Projected Gradient Descent 16

7 Regularization **18**

7.1 L-1 Regularization : LASSO 18

7.2 L-2 Regularization : RIDGE 19

8 Large-Scale and Distributed Optimization 21

8.1 Randomized Block Coordinate Descent 21

8.2 Randomized Block Coordinate Descent combined with SG (Stochastic Sample) 22

9 Advanced Topics on Gradient Descent 24

9.1 Heavy Ball Algorithm 24

9.2 Add Non-Convex Penalty to the Loss (L-0.5 penalty) + GD . 24

List of Figures

2.1 PairPlot of Features 3
2.2 3D Embedding View after PCA dimensions reduction 4
2.3 Correlation Matrix between Features 5

3.2 GD with different constant τ 7
3.3 Sublinear Convergence Rate 7
3.4 Cost Values on Train/Test data $\lambda_{test} = 0.2$ 8

4.1 Elementary Functions 9
4.2 MLP with 4 layers 10
4.3 Chain Evaluation 10
4.4 Check the Gradient computation 11
4.5 MLP layers Impact 11

5.1 SG and GD Comparison with $\tau = \frac{1}{L}$ 12
5.2 Batch Size on SG 13
5.3 Effect of Learning Rate on SG 14
5.4 Stochastic Variance Reduced Gradient 14

8.1 RBCD on LASSO Problem 21
8.2 RBCD+SG on LASSO Problem 22
8.3 RBCD+SG 23

9.1 Heavy Ball + Different Momentum Parameters 25
9.2 Heavy Ball + Different Learning Rates 25
9.3 Non convex penalty + GD 25

Chapter 1

Context

Classification

Recall the Optimization problem for Logistic regression and also the formulas of ∇f :

$$\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{1.1}$$

where every y_i is a binary label in $\{-1, 1\}$ and $\lambda \geq 0$ is a regularization parameter. The gradients are as follows regarding the configuration of optimization we will confront in this work (without, LASSO, RIDGE regularizations) :

RIDGE Problem

$$f_i(w) = \log\left(1 + \exp\left(-y_i x_i^T w\right)\right) + \frac{\lambda}{2}\|w\|_2^2 \tag{1.2}$$

$$\nabla f_i(w) = -\frac{y_i}{1 + \exp\left(y_i x_i^T w\right)} x_i + \lambda w \tag{1.3}$$

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} -\frac{y_i}{1 + \exp\left(y_i x_i^T w\right)} x_i + \lambda w \tag{1.4}$$

LASSO Problem

$$f_i(w) = \log\left(1 + \exp\left(-y_i x_i^T w\right)\right) + \lambda\|w\|_1^2 \tag{1.5}$$

$$\nabla f_i(w) = -\frac{y_i}{1 + \exp\left(y_i x_i^T w\right)} x_i + \lambda \tag{1.6}$$

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} -\frac{y_i}{1 + \exp\left(y_i x_i^T w\right)} x_i + \lambda I \tag{1.7}$$

Chapter 2

Exploratory Data Analysis

2.1 About DataSet

This project consists in implementing optimization algorithms and ML techniques in the context of supervised learning. We will use the Advertising dataset provided on **https://www.kaggle.com/diabetes-dataset**. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Here is a description of the variables of the dataset :

- Pregnancies: Number of times pregnant.
- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
- BloodPressure: Diastolic blood pressure (mm Hg).
- SkinThickness: Triceps skin fold thickness (mm).
- Insulin: 2-Hour serum insulin (mu U/ml).
- BMI: Body mass index (weight in kg/(height in m)^2).
- DiabetesPedigreeFunction: Diabetes pedigree function.
- Age: Age (years).
- Outcome: Class variable (0 or 1).

The data contains $n = 768$ examples and 8 variables other than the Outcome (predicted variable). So the input space is $\mathbb{R}^d$, with $d = 8$. Given train data A and train labels y, y being labels of examples in A ,$y \in \{0, 1\}$ (Outcome variable), the task try to put into practice many optimization algorithms and techniques to minimize the cost function of our learning problem.

2.2 Visualization and Observations

First of all, we checked that there are no missing values and that the two classes contain the same number of individuals. Secondly, a pairplot is provided so as to gain some insight on the data distribution :

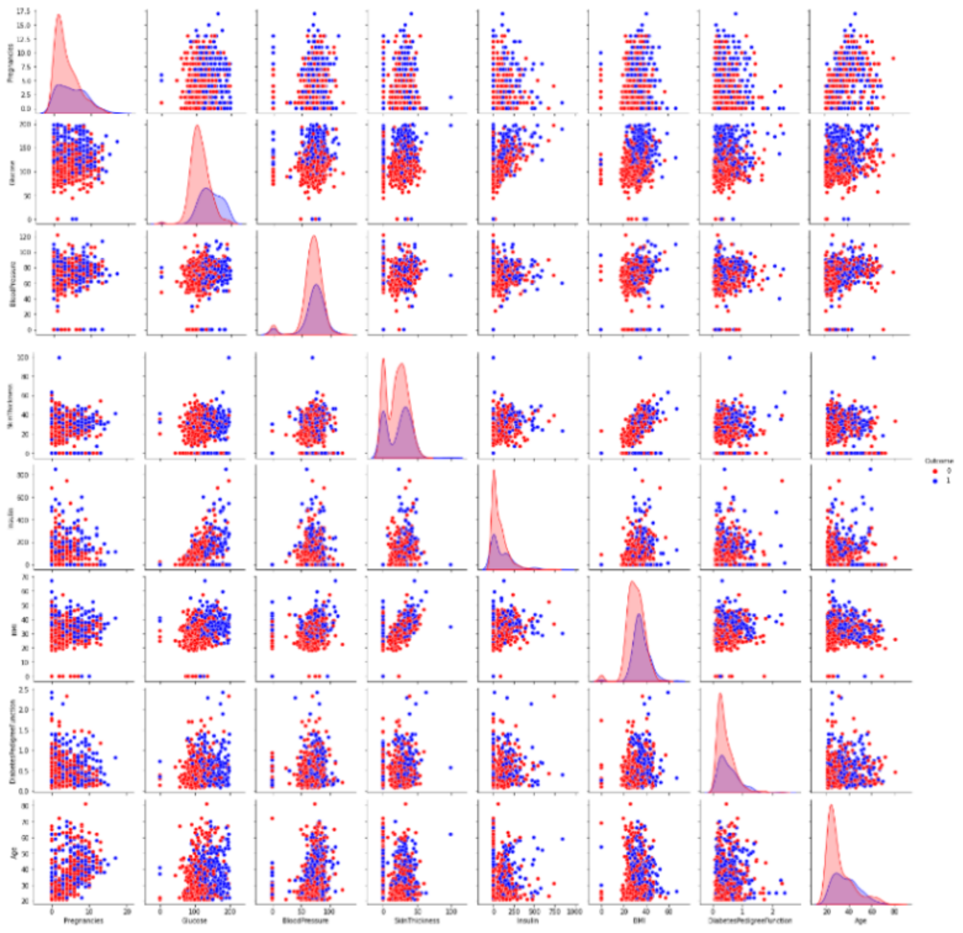


Figure 2.1: PairPlot of Features

Hence, we can observe :

- From scatter plots, to me only BMI, SkinThickness, Pregnancies ,Age seem to have positive linear relationships. Another likely suspect is Glucose and Insulin.

- There are no non-linear relationships.

We split the data into train and test sets . We normalized it and standardized it. In addition, we computed attributes correlation and 2D and 3D Projection with PCA. The figure 2.3 shows a correlation plot.

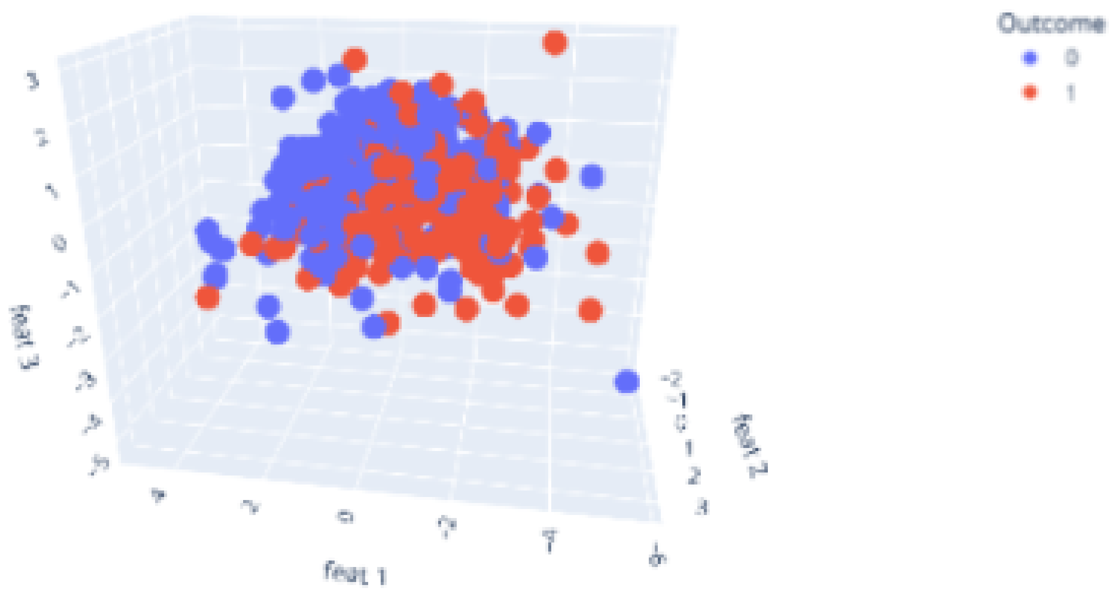


Figure 2.2: 3D Embedding View after PCA dimensions reduction

As we can see in the 3D projection retrieved by PCA in 2.2, the data is quite simple. It seems that the two classes are even almost linearly separable.. The explained variance conserved by PCA is of 61.67% which is not bad. That means our previous remark can be valid. According to the matrix of correlation 2.3, attributes are not highly correlated between them in global; but Pregnancies is a bit strongly and positively correlated with Age, then secondly comes the SkinThickness which is positively correlated with Insulin as well as BMI, which is already confirmed by the pairplot above.

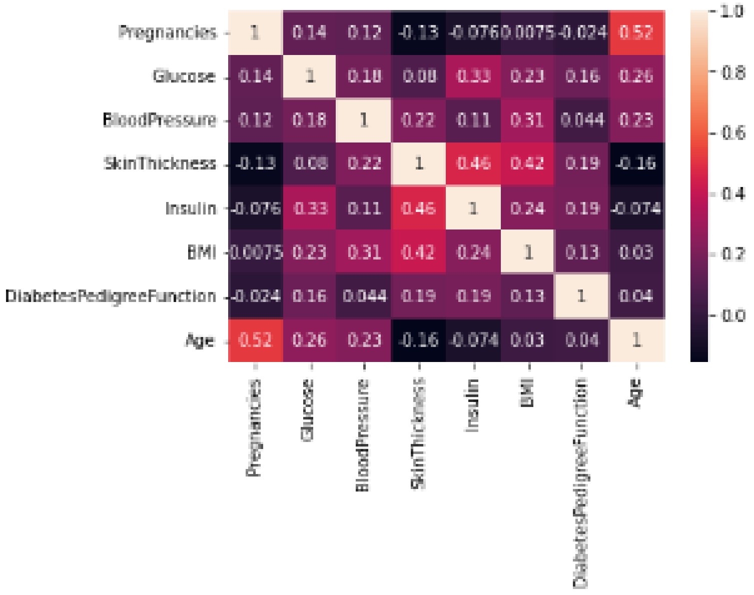


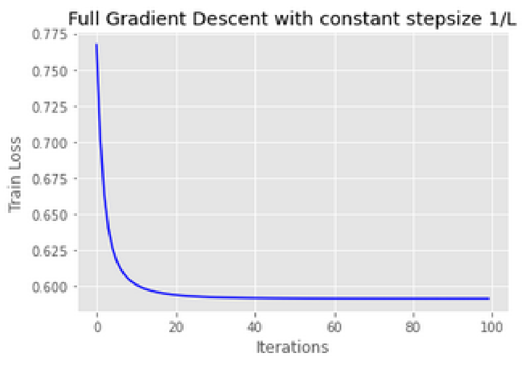
Figure 2.3: Correlation Matrix between Features

Chapter 3

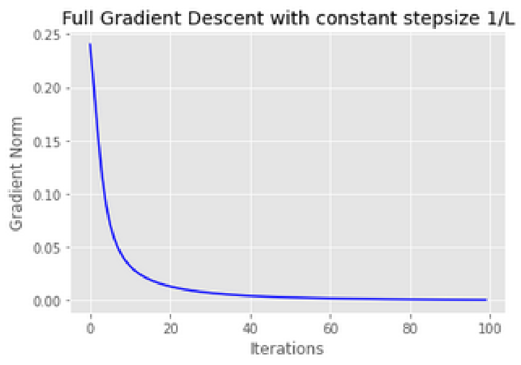
Gradient Descent

3.1 Gradient Descent for Classification with Logistic Loss

The plot 3.1a demonstrates that full gradient descent algorithm converges after around 30 iterations. As highlighted in 3.1b, the gradient's norm is approaching zero.



(a) Cost Function Values on training data



(b) Gradient Norm

3.2 Step Size

The impact of step size aka learning rate on gradient descent was investigated in 3.2 and shows different curves for different values of stepsize. As seen in the course, to ensure theoretically the convergence of FGD, a step size should be picked such as :

$$\tau < \frac{2}{L} \tag{3.1}$$

where $L = \frac{\|A^T A\|}{4n}$ is a Lipschitz constant for ∇f when there is no regularization and n is the number of samples provided in the dataset. The optimal choice for the step size at which the figures 3.1a and 3.1b were generated is $\tau = \frac{1}{L} \approx 1.86$

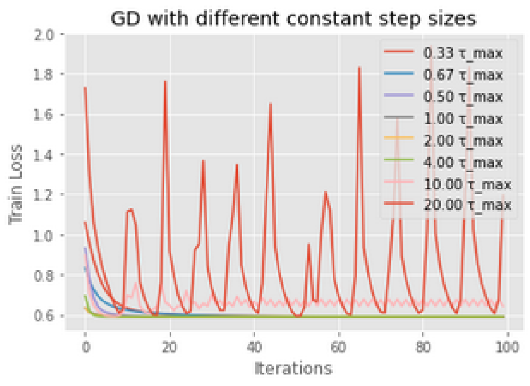


Figure 3.2: GD with different constant τ

3.3 Convergence Rate

As already seen in case of smooth convex functions, we expect a sublinear convergence rate$\mathcal{O}(\frac{1}{k})$. In 3.3, the $\log\left(f\left(x_k\right) - \min\left(f\left(x_0\right), f\left(x_1\right) \dots f\left(x_n\right)\right)\right)$ drawn for different values of τ. The logistic loss is a smooth convex function (it's not strongly convex unless we add an l^2 term and obtain a quadratic convergence rate $\mathcal{O}(\frac{1}{k^2})$). However, in our case, the curves for $\tau \leq \tau_{max}$ are quite parallel to the line of the convergence rate.

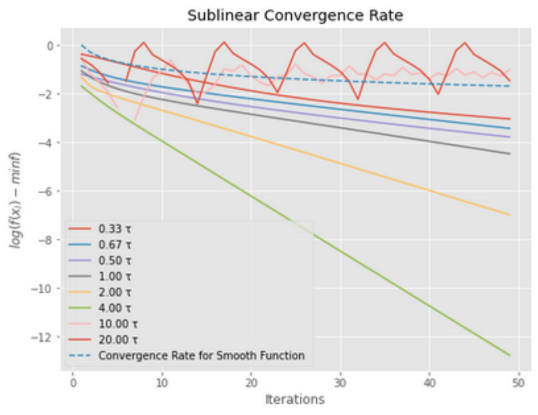


Figure 3.3: Sublinear Convergence Rate

3.4 Performace on Test Set (change the RIDGE penalty)

The dataset use case does not suffer from multicollinearity and regularization is not needed that much in this problem. 3.4 indicates that there is no overfitting from our model and the test cost function decreases.

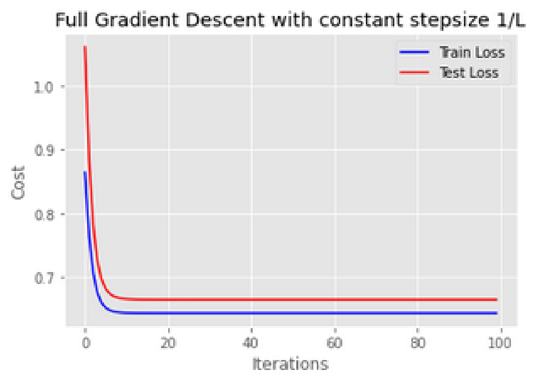


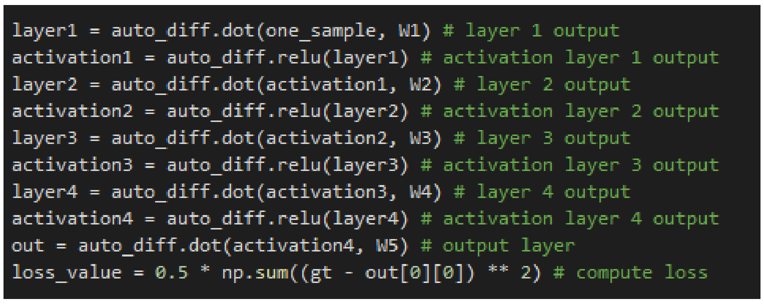
Figure 3.4: Cost Values on Train/Test data $\lambda_{test} = 0.2$

Chapter 4

Automatic Differentiation

4.1 Elementary Functions

In this section, we implemented the elementary necessary functions for the MLP which is elaborated by the 4.1 and include the dot product, the relu activation and the squared loss (the complete squared loss with the its vjp are used in the automatic chain in the next section but here it's mostly for checking purposes). It's worth mentioning that squared loss is not the appropriate loss for the diabetes dataset with classification task, but it's not that bad either.

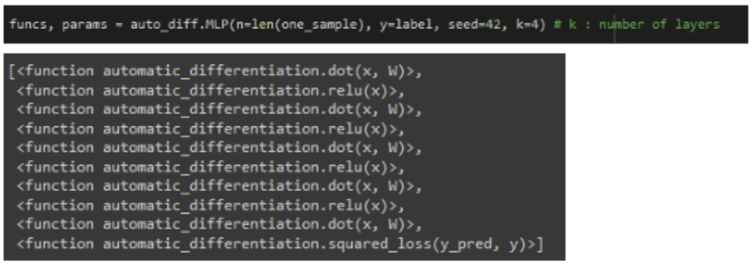


```
layer1 = auto_diff.dot(one_sample, W1) # layer 1 output
activation1 = auto_diff.relu(layer1) # activation layer 1 output
layer2 = auto_diff.dot(activation1, W2) # layer 2 output
activation2 = auto_diff.relu(layer2) # activation layer 2 output
layer3 = auto_diff.dot(activation2, W3) # layer 3 output
activation3 = auto_diff.relu(layer3) # activation layer 3 output
layer4 = auto_diff.dot(activation3, W4) # layer 4 output
activation4 = auto_diff.relu(layer4) # activation layer 4 output
out = auto_diff.dot(activation4, W5) # output layer
loss_value = 0.5 * np.sum((gt - out[0][0]) ** 2) # compute loss
```

Figure 4.1: Elementary Functions

4.2 MLP with K Layers

The MLP in the figure 4.2 has four layers of neurons (K=4), each with a random number of neurons ranging from 3 to 7. It is critical to follow the dimension convention for weight matrices (dimension in the form (n, n-1) where n is the number of neurons in the current layer and n-1 is the number of neurons in the previous layer).



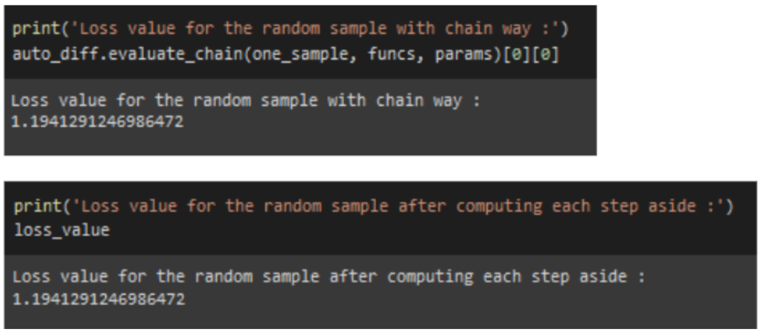
```
funcs, params = auto_diff.MLP(n=len(one_sample), y=label, seed=42, k=4) # k : number of layers
```

```
[<function automatic_differentiation.dot(x, W)>,
 <function automatic_differentiation.relu(x)>,
 <function automatic_differentiation.dot(x, W)>,
 <function automatic_differentiation.relu(x)>,
 <function automatic_differentiation.dot(x, W)>,
 <function automatic_differentiation.relu(x)>,
 <function automatic_differentiation.dot(x, W)>,
 <function automatic_differentiation.relu(x)>,
 <function automatic_differentiation.dot(x, W)>,
 <function automatic_differentiation.squared_loss(y_pred, y)>]
```

Figure 4.2: MLP with 4 layers

4.3 Evalute and Check the Chain Calculus

To check the correctness of the automatic chain calculus, two examples are provided in 4.3 with both automatic and manual to verify the common result. Furthermore, the check of gradient computation is ensured in 4.4



```
print('Loss value for the random sample with chain way :')
auto_diff.evaluate_chain(one_sample, funcs, params)[0][0]
```

```
Loss value for the random sample with chain way :
1.1941291246986472
```

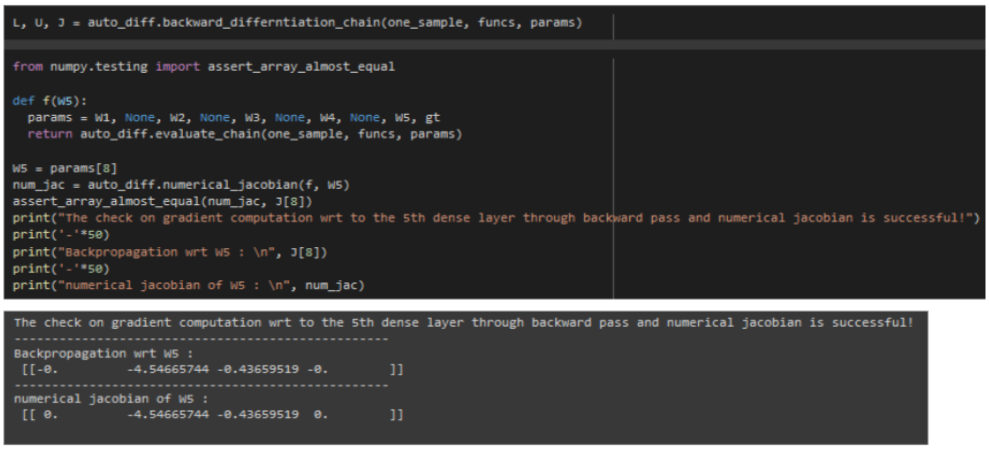
```
print('Loss value for the random sample after computing each step aside :')
loss_value
```

```
Loss value for the random sample after computing each step aside :
1.1941291246986472
```

Figure 4.3: Chain Evaluation

4.4 Implement Training with Stochastic Gradient Algorithm

As mentioned before, the number of hidden units is picked randomly inside the MLP, and a graph illustrating the impact of layer number is provided in 4.5



```
L, U, J = auto_diff.backward_differntiation_chain(one_sample, funcs, params)
```

```
from numpy.testing import assert_array_almost_equal

def f(W5):
  params = W1, None, W2, None, W3, None, W4, None, W5, gt
  return auto_diff.evaluate_chain(one_sample, funcs, params)

W5 = params[8]
num_jac = auto_diff.numerical_jacobian(f, W5)
assert_array_almost_equal(num_jac, J[8])
print("The check on gradient computation wrt to the 5th dense layer through backward pass and numerical jacobian is successful!")
print('-'*50)
print("Backpropagation wrt W5 : \n", J[8])
print('-'*50)
print("numerical jacobian of W5 : \n", num_jac)
```

```
The check on gradient computation wrt to the 5th dense layer through backward pass and numerical jacobian is successful!
--------------------------------------------------
Backpropagation wrt W5 :
 [[-0.         -4.54665744 -0.43659519 -0.        ]]
--------------------------------------------------
numerical jacobian of W5 :
 [[ 0.         -4.54665744 -0.43659519  0.        ]]
```

Figure 4.4: Check the Gradient computation

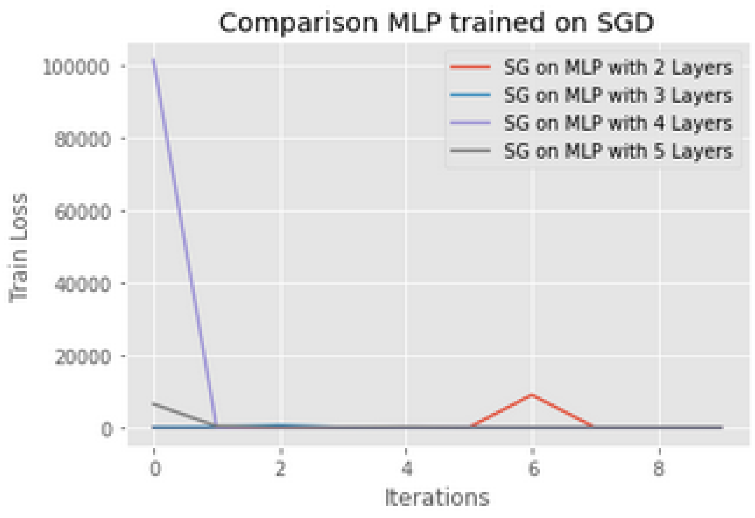


Figure 4.5: MLP layers Impact

Chapter 5

Stochastic Batch Gradient

In SGD (a variant of SBG with number of samples equal to 1), we can think of it as trying to get an approximated gradient instead of exact gradient. The approximation is coming from one data point (or several data points called mini batch). Therefore, in SGD, we can update the parameters very quickly. In addition, if we loop over all data (one epoch), we actually have 600 updates. The trick is that, in SGD we do not need to have 600 iterations (updates), but much less iterations (updates) and we will have "good enough" model to use. It's more suitable in case of large dataset (not the case here = 600 data point).

5.1 SGD and Comparison with GD

As mentioned in the course, there are three assumptions in order to prove theoretical convergence rates and ensuring the decreasing aspect of the cost function. In this section, the same step size $\tau = \frac{1}{L}$ is used.The figure 3.1b demonstrates that the stochastic gradient is so noisy and that it is not a descent method because the loss value can fluctuate over iterations.

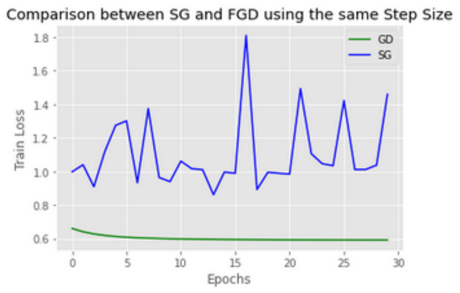


Figure 5.1: SG and GD Comparison with $\tau = \frac{1}{L}$

5.2 Optimal Batch Size

The impact of batch size is investigated in 3.2 to extract the optimal choice that realizes a good compromise between gradient descent and stochastic gradient. This result shows that stochastic gradient methods with a constant step size can only be guaranteed to converge to a neighborhood of the optimal value. It also shows that this neighborhood becomes tighter as batch size increase. However, one can observe that values of mini batch size $6, 30, 60, 100$ have lower cost error and faster convergence than batches with high size $300, 600$ or stochastic gradient (where batch size $= 1$).

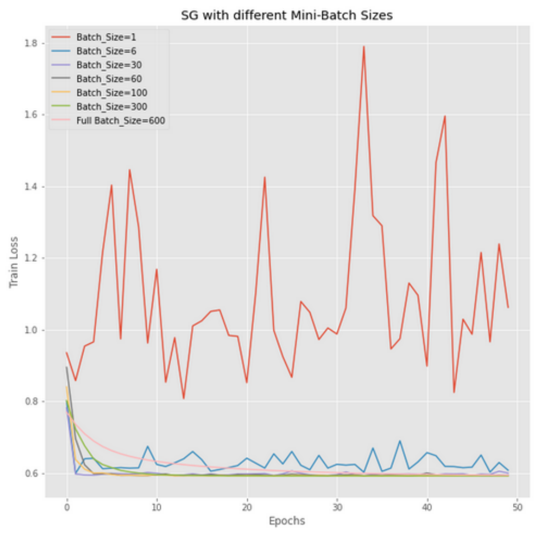


Figure 5.2: Batch Size on SG

5.3 Learning Rate Impact

As expected and similar to the GD case, large step size tends to diverge from the solution but in 5.3 it's more trivial and in that case SG ensure bad learning and worst the GD case.

5.4 Advanced Variant of SG : SVRG

SVRG is specifically designed to reduce the variance that is inherent in the latter method and this can be illustrated by 5.4 where fluctuations are suppressed. As expected from the algorithm where a reference full gradient is

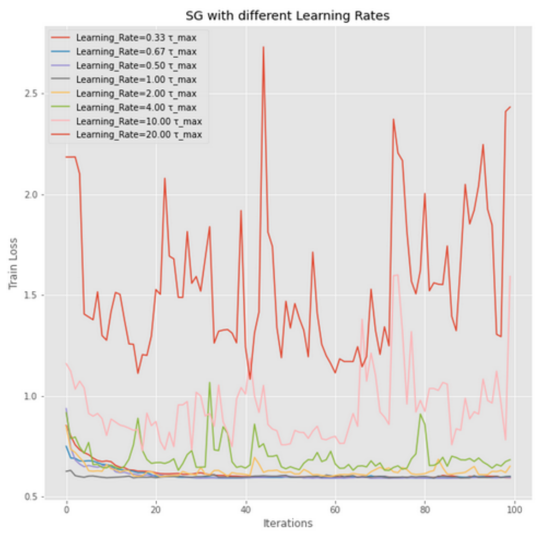


Figure 5.3: Effect of Learning Rate on SG

first evaluated in the outer loop (over 2 loops) and then used to yield a variance reduced estimate of the current gradient in the inner loop.

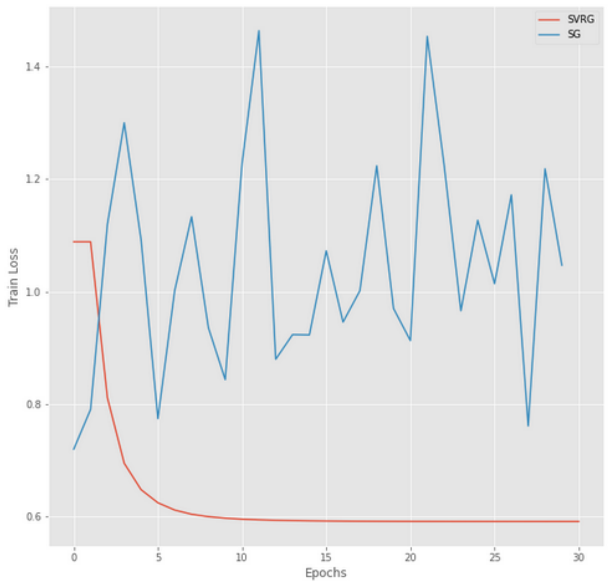


Figure 5.4: Stochastic Variance Reduced Gradient

Chapter 6

Convexity and Constrained Optimization

Now, we consider optimization algorithms for constrained problems with a convex objective function and a convex feasible set. There are some details not covered in this report about the following algorithms theory which are well explained in research papers.

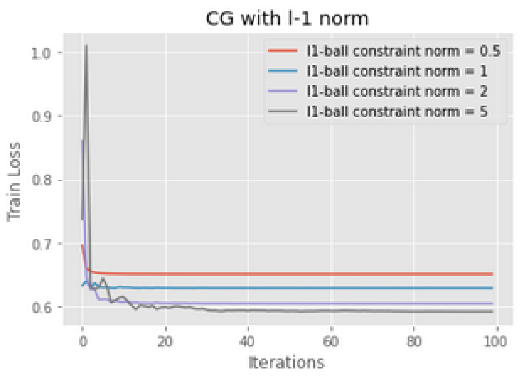
6.1 Conditional Gradient Descent

For Conditional Gradient Descent, we choose an initial x and then at each iteration we update such that :

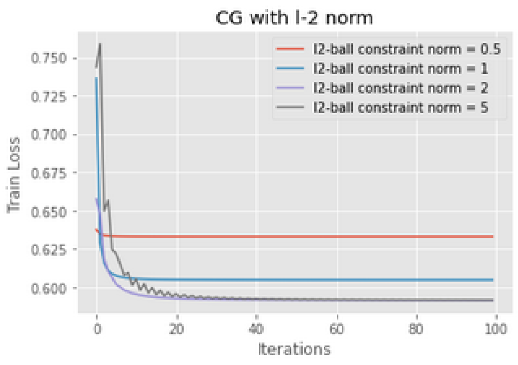
$$\begin{aligned} s^{k-1} &\in \text{argmin}_\text{s}\, \nabla f(x^{k-1})^T s \\ x^k &= (1-\gamma_k)x^{k-1} + \gamma_k s^{k-1} \end{aligned} \tag{6.1}$$

where $\gamma_k = \frac{2}{k+1}$

The results in 6.1a 6.1b reveal that there are oscillations in the loss value for both l-1 and l-2norm constraints of 2 and 5. This shows that CG is not a descent algorithm i.e the function value is not guaranteed to decrease at every iteration. We notice that CG reaches lower values of the objective function for higher values of L1 norm ball radius. This can be attributed to the radius of the ball being too small or sufficiently large (the same for l-2 norm). Also, if CG algorithm finds an optimum near one after which expanding the constraint region (increasing the radius of the ball) has no effect and the same optimum is discovered (case of 2 and 5). Finally, In cases where norm =0.5, 1, the behaviour of CG indicates that the chosen constraint is small and cannot make progress. On the other hand, cases of norms (2, 5) show that the problem can converge if we increase the radius of constraint.



(a) Cost Function Convergence with CG on l-1 norm.



(b) Cost Function Convergence with CG on l-2 norm.

6.2 Projected Gradient Descent

We consider the constrained problem such that :

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & x \in C \end{aligned} \tag{6.2}$$

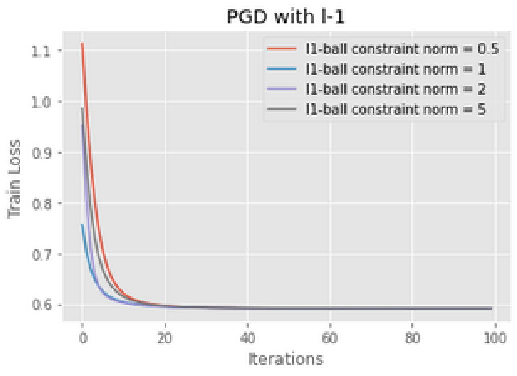
For Projected Gradient Descent, we choose an initial x and then at each iteration we update such that :

$$x^k = \mathrm{P_C}(x^{k-1} - \alpha_k \nabla f(x^{k-1})) \tag{6.3}$$

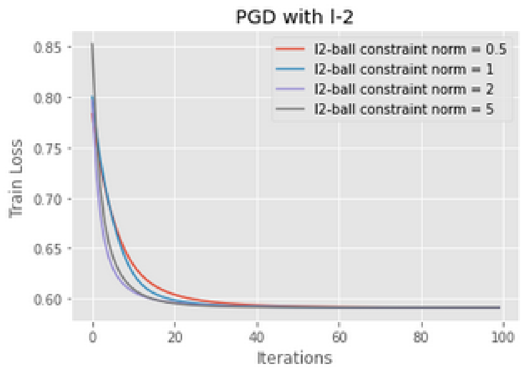
where $\mathrm{P_C}$ is the projection operator onto the set C.

In 6.2a and 6.2b, we hightight that in all constraints norm values, the behaviour of PGD indicates that the chosen constraint is convenient to progress and converge. However, there is a stagnation for each case : at every iteration, PGD is pushing out the vector x of the L2 norm ball which is then being projected back onto the same position leading to and being nearly the same for every iteration.

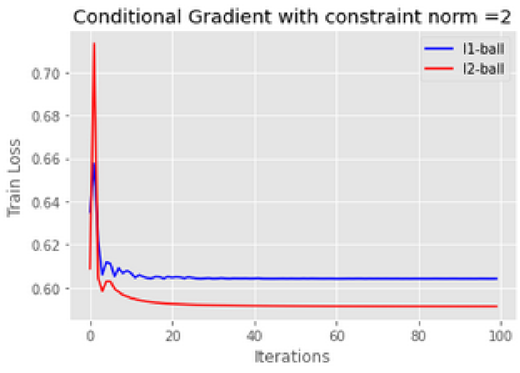
Finally, 6.3a The difference between the two curves with CD is due to the fact that solution with L2 norm can be comparable with L1 norm solution if we increase the ball radius for the latter norm. Nevertheless, PGD 6.3b shows no difference between the two norms curves.



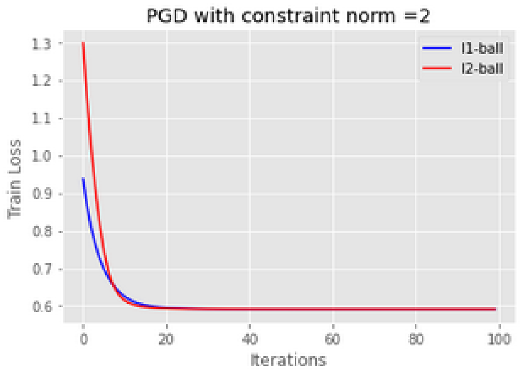
(a) Cost Function Convergence with PGD on l-1 norm.



(b) Cost Function Convergence with PGD on l-2 norm.



(a) CG : Norms Comparison.



(b) PGD : Norms Comparison .

Chapter 7

Regularization

Although regularization is unnecessary in this case where we have p = 8 ($\ll$ n = 600) extracted numerical features (Daily Time Spent on Site, Age, Area Income, Daily Internet Usage, Male), still we want to observe the impact of regularization on the solution.

7.1 L-1 Regularization : LASSO

The simplest iterative algorithm to perform the minimization is the so-called iterative soft thresholding (ISTA), aka proximal gradient. It performs first a gradient step (forward) of the smooth part of the loss function and then a proximal step (backward) step which account for the L-1 penalty and induce sparsity.

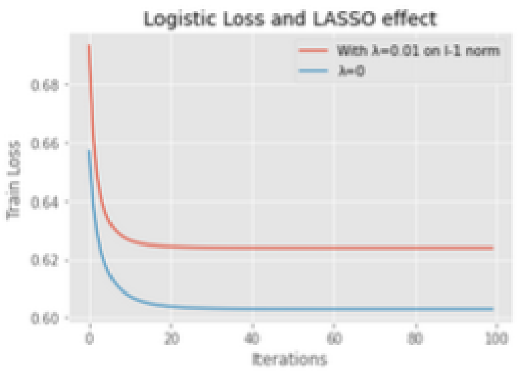
$$x_{k+1} = \operatorname{Prox}_{\tau \cdot g}\left(x_k - \tau \cdot \nabla\left(f\left(x_k\right)\right)\right) \tag{7.1}$$

And the proximal step is the soft-thresholding operator as follows :

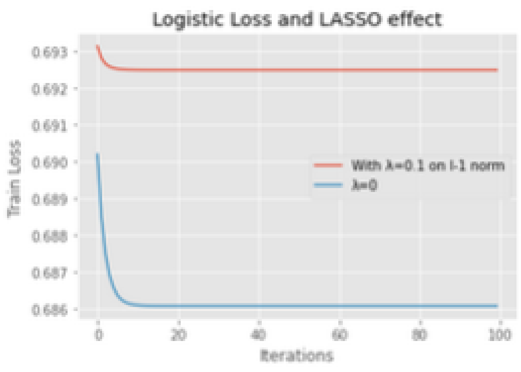
$$S(x) \equiv \max(|x| - \lambda, 0)\,\operatorname{sign}(x). \tag{7.2}$$

In figures 7.1a and 7.1b, we notice the effect of penalty amount on the cost function. At certain point, a large penalization can lead to a divergence from an appropriate learning, which is also the case for Ridge penalty.

In 7.3a, the regularization paths for Lasso is ivestigated in order to study the effect of regulatization coefficient λ on the sparsity of the final weight vector returned by the proximal gradient algorithm. It is, in fact, completely consistent with the main conclusion during the class. The more the parameter λ is increased, the Lasso penalty forces some coefficients to zero. We also observe that Insulin and BloodPressure features reach zero first among all the features. The λ value that yield a sparse solution here is $\lambda \approx 0.1$



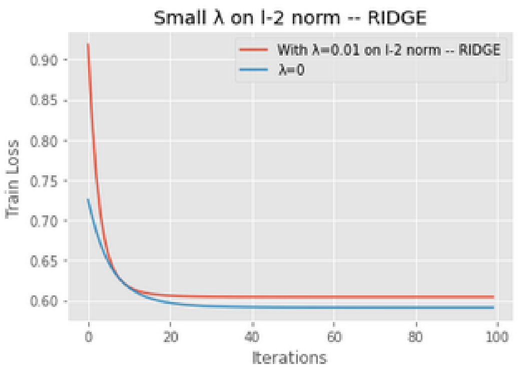
(a) Small L-1 Penalty



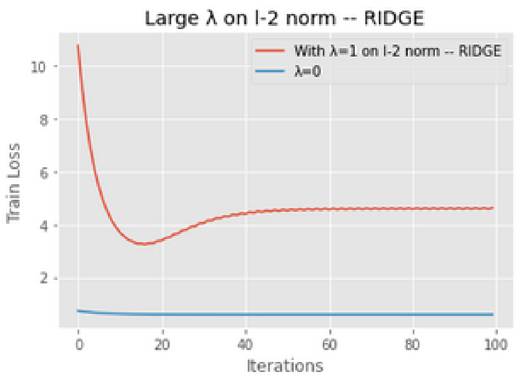
(b) Large L-1 Penalty

7.2 L-2 Regularization : RIDGE

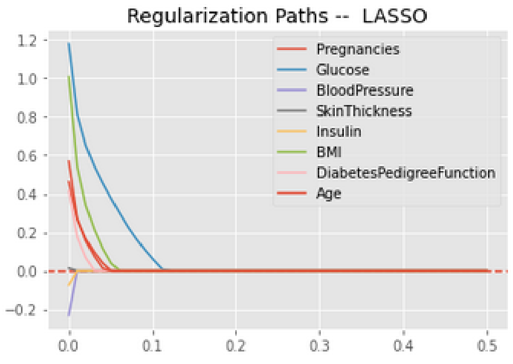
Ridge regularization is on the other hand simple to tackle. We modify the regularization parameter value and plug it into the gradient descent function. The evolution of the loss function in both figures 7.2a, 7.2b has the same behavior except the starting warm-up. So it's not that beneficial here to include regularization.



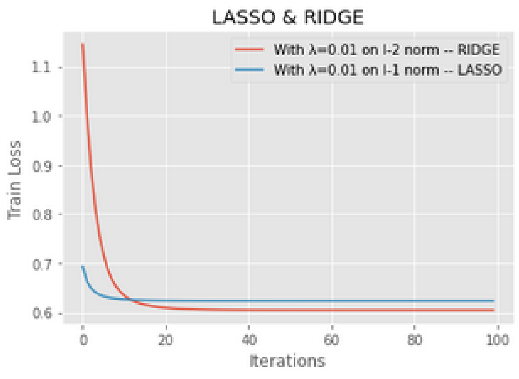
(a) Small L-2 Penalty



(b) Large L-2 Penalty



(a) Regularization Paths over Features



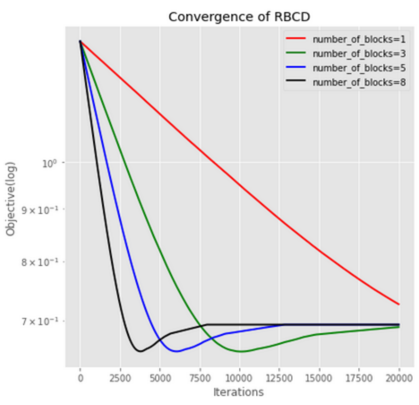
(b) RIDGE v LASSO

Chapter 8

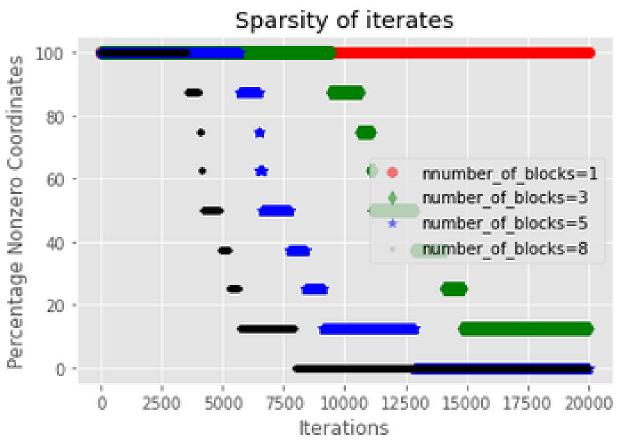
Large-Scale and Distributed Optimization

8.1 Randomized Block Coordinate Descent

The idea behind RBCD algorithm is the fact of successively minimizing along coordinate directions randomly picked to find the minimum of the cost function. An experience over 20000 iterations is launched within a regularized setting ($\lambda = 0.15$). Recall that we have 8 features for this work.



(a) Randomized Block Coordinate Descent

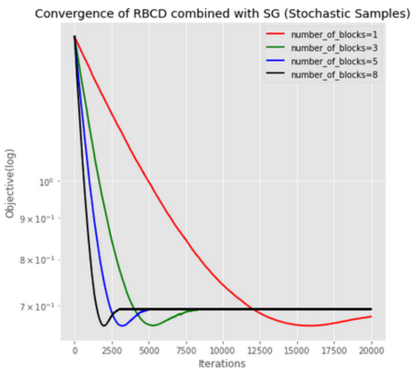


(b) The sparsity of Iterates

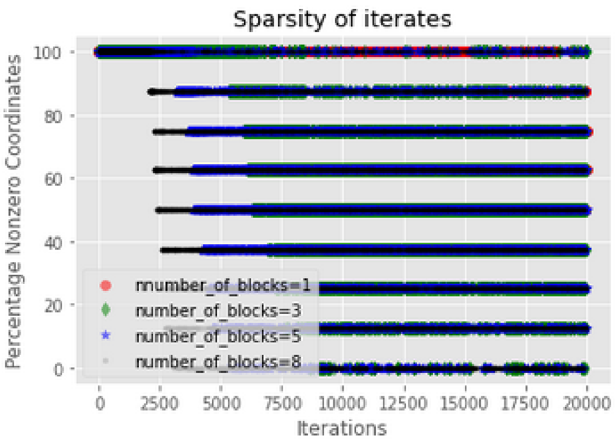
Figure 8.1: RBCD on LASSO Problem

8.2 Randomized Block Coordinate Descent combined with SG (Stochastic Sample)

This section treats the same logic of algorithm and adds the stochasticity of picking samples (batch size = 1). Thus, we can study the impact of regularization : first of all, we tested the same configuration through 8.2a as in the previous section ”with lasso regularization”. Secondly we deleted the regularization term and we notice there is no longer a smooth behavior (noisy) of curves for any number of coordinates blocks and we can recognize this effect in SG in 9.1. In 8.2b, the percentage of non zero coefficients are plotted through iterations. The more coordinates we pick to update, the less non zero coefficients due to regularization. Globally, there is no benefit in adding SG feature in this case. In fact, the convergence become slower than the previous version.



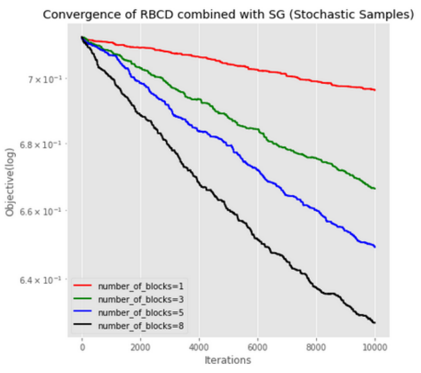
(a) Randomized Block Coordinate Descent combined with SG



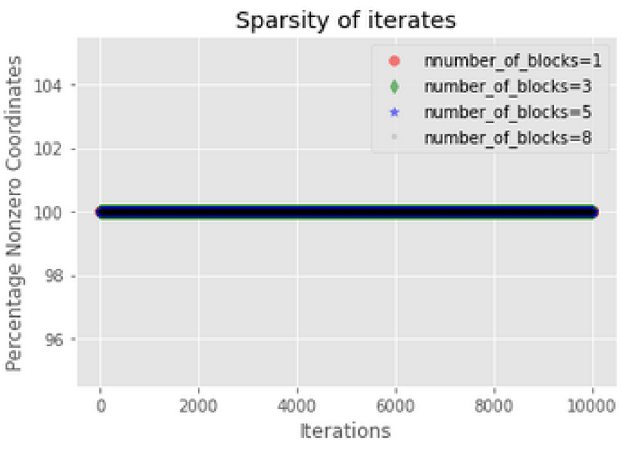
(b) The sparsity of Iterates

Figure 8.2: RBCD+SG on LASSO Problem

In case of non regularization, we don’t expect a sparse solution : the solution vector have 100% as percentage of non zero coefficients as in 9.2.



(a) RBCD + SG with no Regularization



(b) The sparsity of Iterates

Figure 8.3: RBCD+SG

Chapter 9

Advanced Topics on Gradient Descent

9.1 Heavy Ball Algorithm

In figure 9.1 and 9.2, we can see respectively that the experience conducted to try different different momentum parameters to speed up the convergence and the effect of the step size on that. In this dataset case, there is no much difference between heavy ball and GD methods. However not all parameters are quite good for this task. In fact lower γ seems to not work well on this dataset and neither high values such as the case of 1 and 0.9. The optimal momentum value that can make HB and FD comparable is $\gamma = 0.8$.
The impact of step size is not surprising. The algorithm gives good result with the range of step size values as seen before in previous sections (here we use $\tau = \frac{1}{L}$.

9.2 Add Non-Convex Penalty to the Loss (L-0.5 penalty) + GD

Convex optimization tools like gradient descent could be used to find a local minimum for 0¡p¡1, but there's no guarantee that it would be globally optimal. As illustrated in 9.3, with small regularization term, we opt to have a local minimum and the model diverges again.

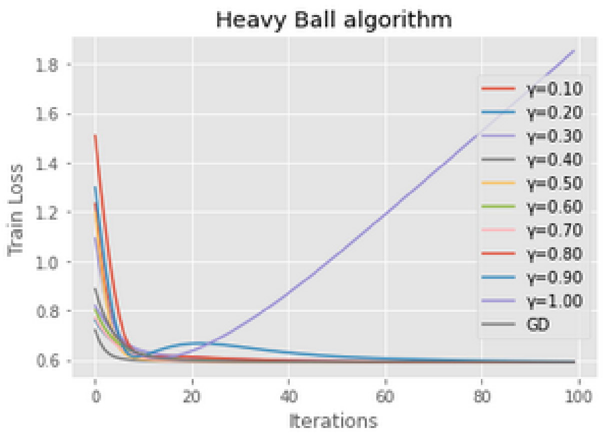


Figure 9.1: Heavy Ball + Different Momentum Parameters

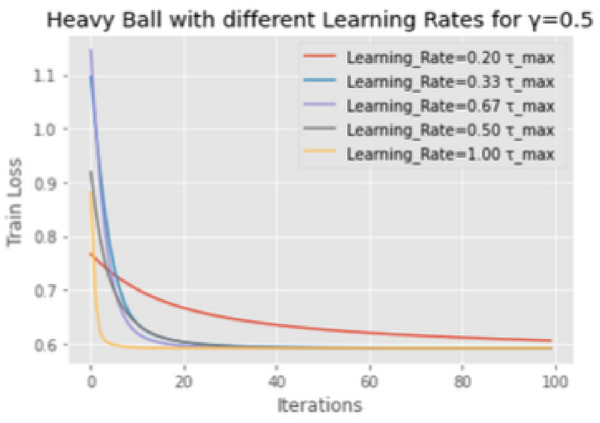


Figure 9.2: Heavy Ball + Different Learning Rates

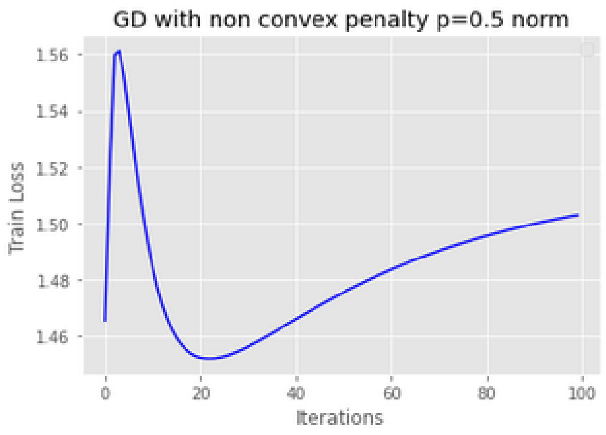


Figure 9.3: Non convex penalty + GD