Universitat Politècnica de Catalunya

Barcelona School of Informatics

Master in Innovation and Research in Informatics

Optimization Techniques for Data Mining

- Unconstrained Optimization -

Author
Marc Mendez
Ricard Meyerhofer

Lecturer
F.-Javier Heredia

January 14, 2020

Index

1 Statement

The aim of this project is to build a Single Layer Neural Network using first derivative optimization methods, compare and evaluate them. In this particular case, we are going to build a SLNN which will be able to recognize numbers from 0 to 9. Each number, consists of a matrix of 7x5 pixels which have value 10 if white and -10 if black, then this content is vectorized and blurred with a Gaussian noise.

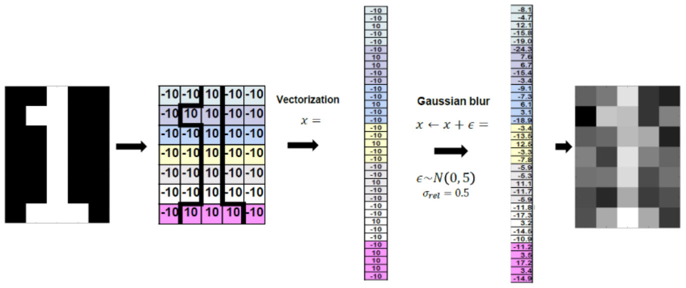

Figure 1

As we have a single layer, we have 35 neurons where each one, represents a pixel. In order to be able to classify correctly numbers, we need to find the associated weights to each neuron connect to the output layer. This output layer, provides a probability value which indicates if the digit is in the target set or not.

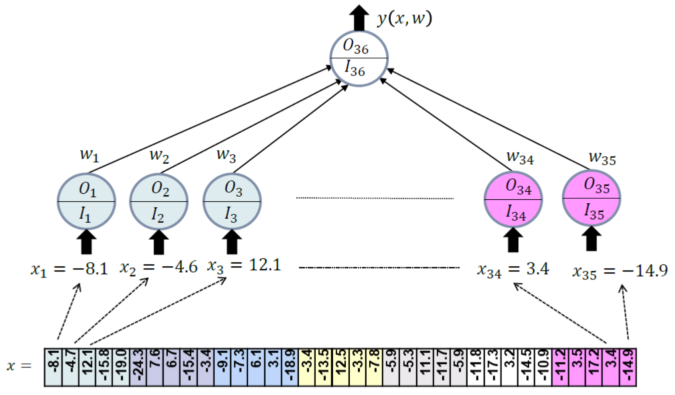

Figure 2

Finally, our objectives in this project are the following:

- Find which is the value of the regularization parameter λ that gives us the best results for the overall set of digits and optimization methods.
- For the previous λ, evaluate which is the best optimization algorithm, GM, QM-BFGS based on the analysis of the variables $Accuracy^{TE}$, $niter$ and tex. Describe how the execution-time per iteration (tex/niter)behaves for the three different methods and find an explanation.
- Describe how the accuracy of the pattern recognition $Accuracy^{TE}$ depends on the digit for the value of λ and optimization method determined in previous section. For the digit with the worst value of $Accuracy^{TE}$, display the results and provide arguments why the recognition rate is bad.
- Lastly, make use of the trained SLNN to develop a function that can identify series of 5 digits.

2 Best regularization parameter and optimization algorithm

In this section first we are going to evaluate which is the best regularization parameter in our executions and once we have decided which α is the best one, we are going to decide from that α, which optimization algorithm is the best.

2.1 Which is the best regularization parameter?

In order to find which is the best regularization parameter, we need to complete the code from the *uo_nn_batch.m* file. What the *uo_nn_batch.m* script does is to execute the whole process from adding noise to the data as we previously explained to optimizing with the GM or the QM-BFGS algorithms. Once executed, we have as result csv file which contains the number we are testing on (*num_target*), the regularization parameter used (*la*), the optimization algorithm used (*isd*) the number of iterations until it found convergence (*niter*), the elapsed time (*tex*) and finally the accuracy (*te_acc*).

If we take a look at the execution data that we obtain, we have a total of 60 instances. Note that this execution had as input parameters the ones that were stated in the assignment:

- **Optimization parameters:** $epsG = 10^{-06}, kmax = 5000, almax = 10, c1 = 0.01, c2 = 0.45, kBLSmax = 30$ and $epsal = 10^{-03}$.
- **Parameters to generate training and test sets:** $tr\_p = tr\_q = 250, tr\_freq = 0.5, te\_freq = 0.0$. Our seeds are our ID's.

We are going to consider only which of the three λ has the biggest te_acc. In this particular case we can see that the best regularization parameter is $\lambda = 1$ with an 98.68% of accuracy.

In the barplots below, we can see each of the results with $\lambda = 0, \lambda = 1, \lambda = 10$:

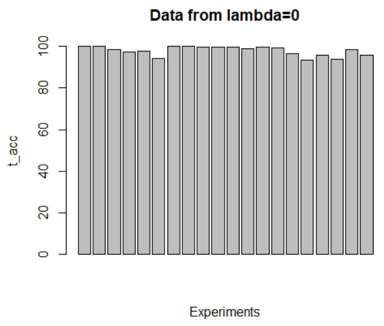

Figure 3: Accuracy with lambda=0

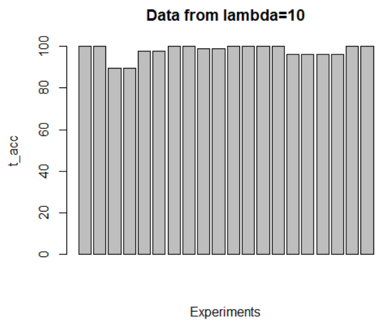

Figure 4: Accuracy with lambda=1

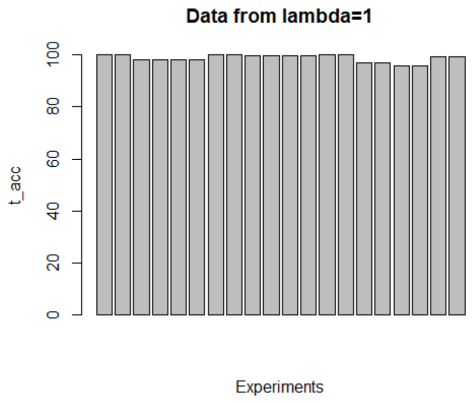

Figure 5: Accuracy with lambda=10

Note that if our classifier had to deal more with some number with others, we would weight the accuracy by the importance of each of the digits in its real application but, since there is not a statement implying that there is different frequency of numbers to classify, we assume that there is the same distribution for each.

2.2 Which is the best optimization algorithm?

In this section we are going to evaluate which is the best optimization algorithm given a $\lambda = 1$ which has been determined in the previous section. We are going to base our analysis on the variables *te_acc*, *niter* and *tex*.

To do so we are going to split our data according to one method or to the other, and we are going to visualize it according to each parameter.

In the case of *te_acc*, we can see that the values according to one optimization parameter or another are completely the same. Which means that both converge and reach the same optima.

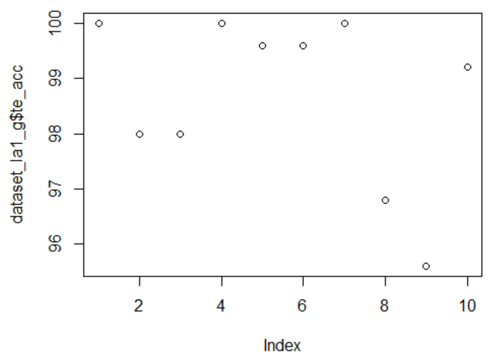

Figure 6: Accuracy with BFGS method

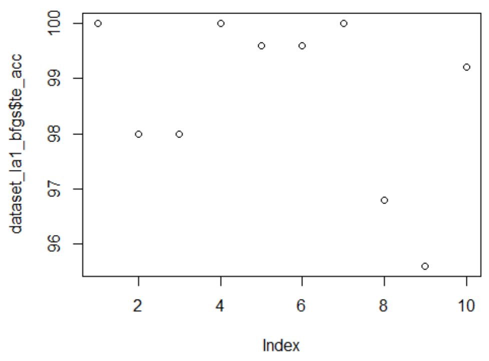

Figure 7: Accuracy with Gradient method

However, this does not mean that the method is the same. We can see that the number of iterations (*niter*) is lower in case of the BFGS method which makes it preferable because it also takes less time.

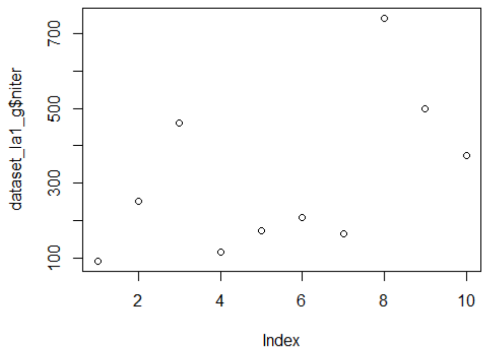

Figure 8: Niter with Gradient method

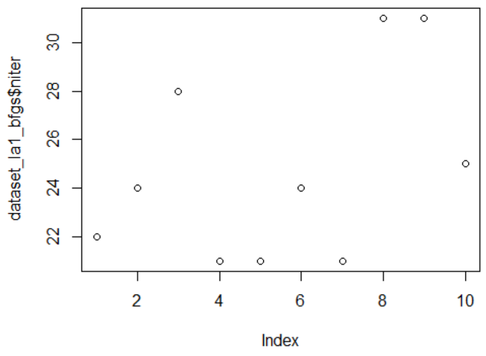

Figure 9: Niter with BFGS method

Finally we can see that *tex* and *niter* are correlated as we can see in this plot. So the more iterations the more will take (which makes sense).

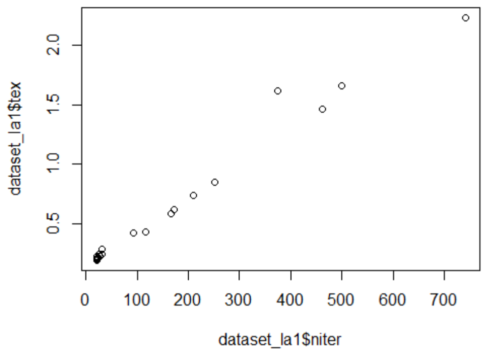

Figure 10: Correlation between number of iterations and time of execution

Therefore, we can say that the best optimization problem for us is the Broyden-Fletcher-Goldfarb-Shanno(BFGS) algorithm. Which takes less to converge in comparison to Gradient Method.

As we can see the BFGS has a lower execution time for iteration. As we

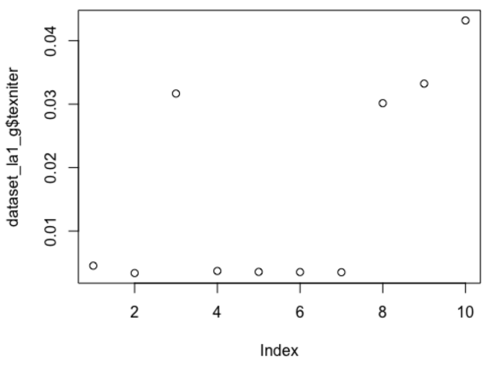

Figure 11: Tex/niter Gradient.

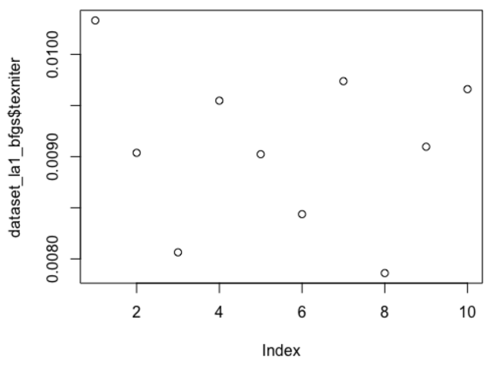

Figure 12: Tex/niter BFGS.

saw on the last part that the correlation between tex and niter is almost maximum we can only say the BFGS is faster in overall.

That's due to the fact that BFGS is an implementation of Quasi-Newton Methods(QM) which use a second derivative matrix that helps when choosing the direction of the next iteration. This is why the iterations in time take longer but at the end the number of iterations is much lower because of having a better direction.

3 Accuracy dependency on regularization parameter and optimization method

On the last section we mentioned the best possible regularization parameter and algorithm for our case. $\lambda = 1$ and BFGS. As we can see on the figure below, this particular case works perfectly(100 accuracy) for digits 1,4,7, almost perfect(less than 100, more than 98) for digits 2,3,5,6,0, and the worst values which are 8 and 9 (accuracy 97 and 95.5 respectively).

So we can see that our worst result is with number 9 for this particular algorithm and regularization parameter. We will try to analyze the results through the *uo_nn_Xyplot* function and see what produces this mistakes.

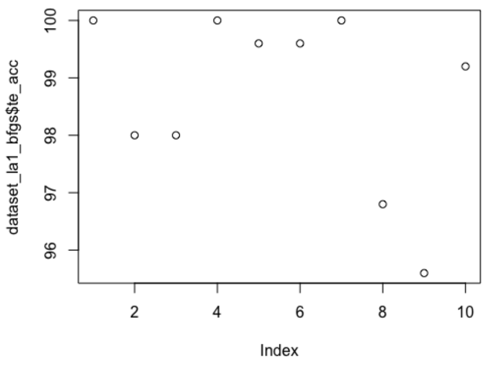

Figure 13: Values of Accuracy for $\lambda = 1$ and BFGS for every digit from 1 to 0(10 on the index row).

3.1 Results of the worst value of $Accuracy^{TE}$

This figure will be attached to the project($num9la1bfgs.png$) to help the mistakes detection as we probably need to zoom it a little bit. As we can see

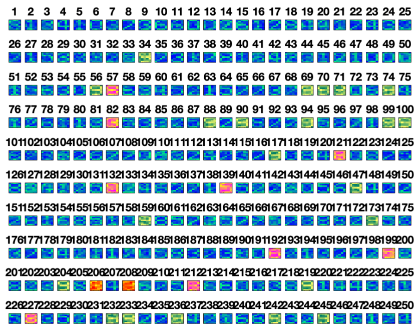

Figure 14: Plots for numtarget=9, $\lambda = 1$ and BFGS.

on the Figure 14, the algorithm makes 11 mistakes that we can classify into 9 false positives and 2 false negatives. If we analyze the number 57 the SLNN classified it as a 9 but apparently it is not it could be an 8 which is quite similar and with the blur it can confuse people. If we had to say a digit on 57 I will also say it's a 9, so the mistake here is quite understandable. For the others false positives most of them are quite confusing even a person could miss.

Talking about the false negatives, which are the ones the algorithm classified as 'no 9' but in fact they are. Number 206 and 208. Both of the images are quite blurred and is difficult to say what number it is.

4 Developing a SLNN to identify 5 digit series

The code related to this part is found under the name of *uo_nn_batch_c.m* and *uo_nn_solve_c.m*. This code generates the matrix of weights which is stored and can be read, and then uses this matrix to calculate the 5 digit num target generated randomly. It reads the digit and goes for the column of the matrix of weights optimized corresponding to the digit read. Then the accuracy is the mean of all the individial accuracies. On the following table will show some particular results but as the num_target is generated randomly we cannot reproduce the same example.

Numtarget	**tex**	**Accuracy**
20305	0.0711	71.4
05438	0.0637	68.6
41797	0.0639	68.9
13574	0.0636	57.4
44502	0.0684	78.2
60533	0.0684	74.8
39028	0.0696	65.7
79959	0.0977	76.2
63856	0.0700	69.6
91112	0.0643	79.0

Table 1: Results for the 5 digit series

Each one of the rows is a random generated number with its execution time, much lower than before because we don't need to calculate the weights matrix. As we can see the accuracy, calculated as the mean accuracy of each individual digit is much lower than before but most of them are over 70% which is a reasonable accuracy.