Pattern recognition with Single Layer Neural Network (SLNN)

Marc Vernet Sancho
Jordi Puig Rabat
tr_seed = 425924225
te_seed = 223314308

Table of contents

Introduction

The main goal of this project is to create a Single-Layer Neural Network to identify digits using different first derivative optimization methods. Every digit will consist of a 35 pixel matrix (7x5). In the matrix each pixel is assigned a 10 or -10 value and then blurred with Gaussian noise. The neural network will be able to identify all digits between 0 and 9.

A SLNN consists of an only layer of neurons, in this case there are 35 neurons in the layer. The objective of the optimization methods is to minimize the *Loss function* (with L2 regularization with parameter λ) that will be used to obtain the weights between each neuron and the output neuron. The optimization methods that will be used are Gradient Method (GM), Quasi-Newton Method (QN-BFGS) and Stochastic Gradient Method (SGM).

The main objective for the computational experiments that will be developed are to study how the regularization parameter (λ) affects the results and the relative efficiency of the different optimization algorithms.

In the code appendix we can find all the code requested, there is a script for each routine: **solve**, **descent direction**, **SGM** but the implementintation of the GM and QNM algorithms is in one file (*optimize.m*). Moreover there is the data generator in *main_batch.m* and the corresponding data (with the seeds used) in the *.csv* file. You can run the main program with a single configuration or run all the configurations running *main_batch.m*.

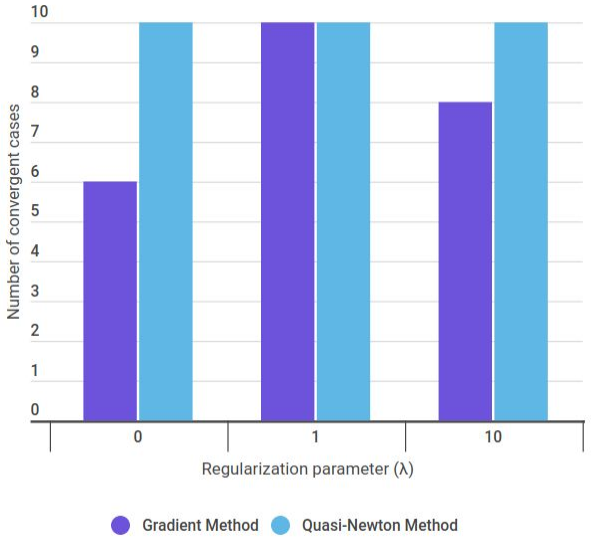
1. Convergence of the algorithms

In this section we are going to check for every optimization algorithm if it has global convergence and how is it's local convergence. An algorithm has global convergence if it's able to find a stationary solution, local convergence it's how fast the algorithm finds the solution.

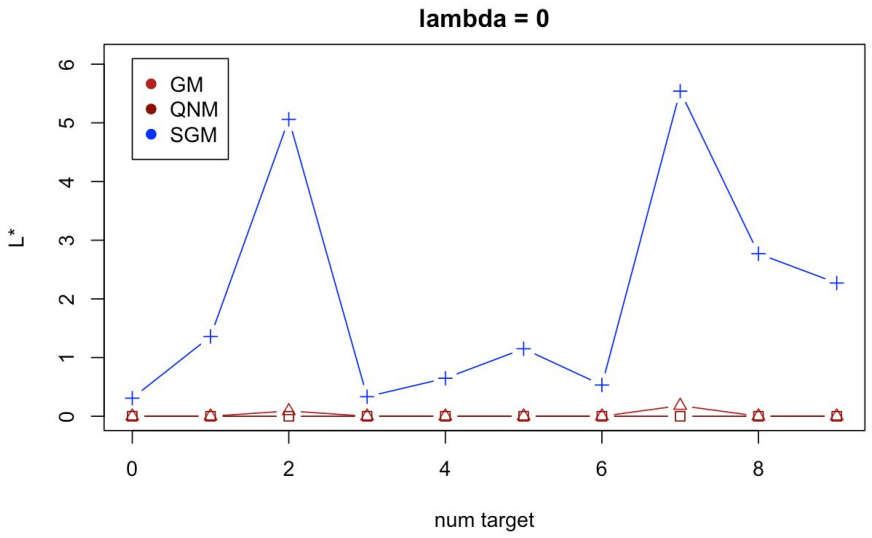
1.1 Global convergence

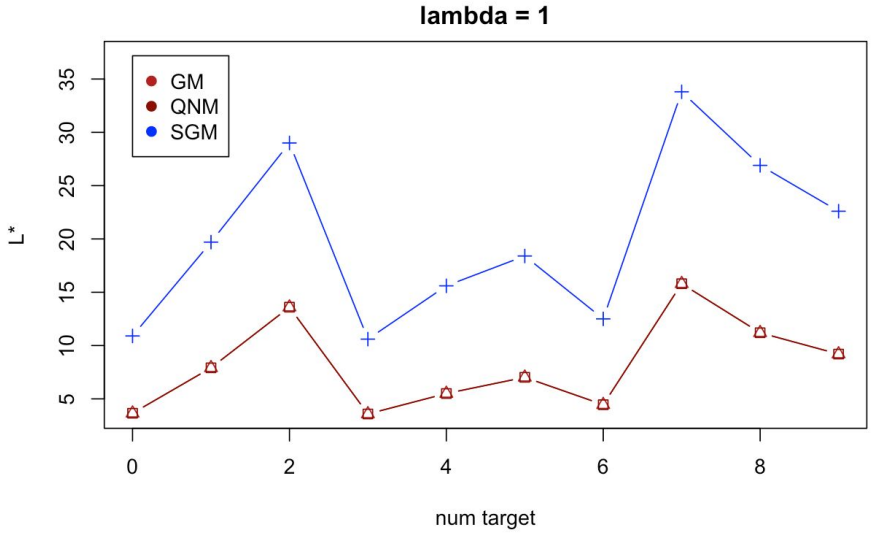
First of all, we will analyze the global convergence of GM and QN-BFGS methods. Theoretically it's known that both algorithms can achieve global convergence if certain criteria are meet. We will be looking at the number of iterations (niter) that each instance needs to find an optimal solution. It will be considered non convergent the cases that reach the iteration limit set at the code (1000 iterations).

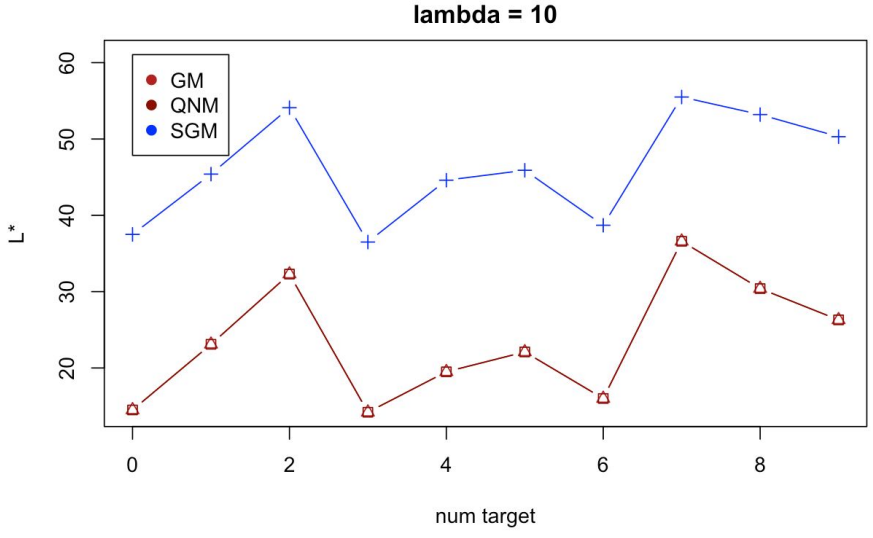
In the plot below it can be seen how many cases of each of both algorithms for every λ value is convergent. The QN-BFGS algorithm is always convergent, GM is convergent in most of the cases but has some non convergent cases, especially for λ = 0.



If we check the objective function, we can see that for the convergent cases both GM and QN-BFGS have very similar or the same result. Both converge to the same stationary point. For the non convergent places it can be checked is it's objective function value is similar to the value of the convergent with same characteristics to know how near it was to converge. Here we have plotted the loss function value over the different target digits; we have 3 graphs, one for each λ value. As the legend indicates, we have different colors for each algorithm used. The plots are in the next page.

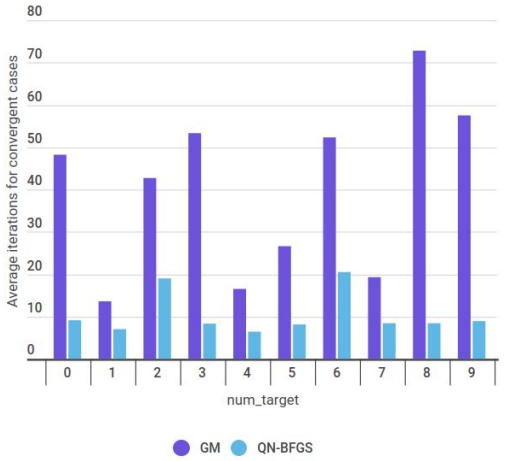

lambda = 0
GM
QNM
SGM
L*
num target


lambda = 1
GM
QNM
SGM
L*
num target

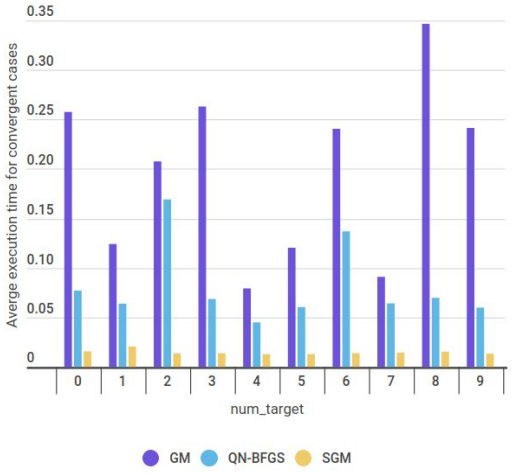

lambda = 10
GM
QNM
SGM
L*
num target

1.2 Local convergence

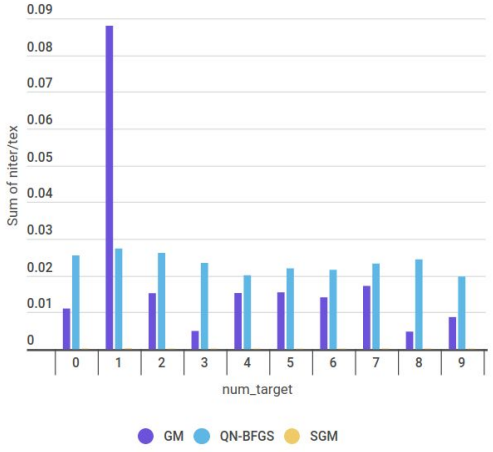
The plots for this section have been generate without the non convergent cases, because we are interested in studying the algorithms behaviour in the cases where they work well.
In the first plot we can see the average number of iterations (niter) for GN and QN-BFGS algorithms in function of the target number (num_target). SGM method hasn't been plotted here because it always use 1000 iterations. As it can be seen, GM needs much more iterations than QN-BFGS.



In the time plot (tex), we can see that execution time for GM is always higher than the time needed by QN-BFGS. This is probably due to the fact that GM needs more iterations. Here we can also see that the time used by SGM is much lower than the time used by both the other algorithms, although it always uses much more iterations than the other algorithms.



The third plot represents the fraction between number of iterations and execution time needed (niter/tex). As it can be supposed by the information of the last plot, SGM has a really low fraction of time used for each iteration. Also the fraction of time used by QN-BFGS is higher that the one used by GM (with one exception).
The information obtained can be interpreted with the theoretical knowledge of how each algorithm works. In the GM algorithm the calculations needed at each iteration are very simple so the fraction of time is small, but this results in needing more iterations the reach a stationary point and therefore needing more time. The QN-BFGS needs more time for each iteration, because it has to calculate the hessian approximation, but this allows it to reach a solution with less iterations and less time.
Theoretically, it's known that GM has linear local convergence, and QM-BFGS can have superlinear convergence if certain criteria are met. So in general, QM-BFGS will reach a solution faster than GM.

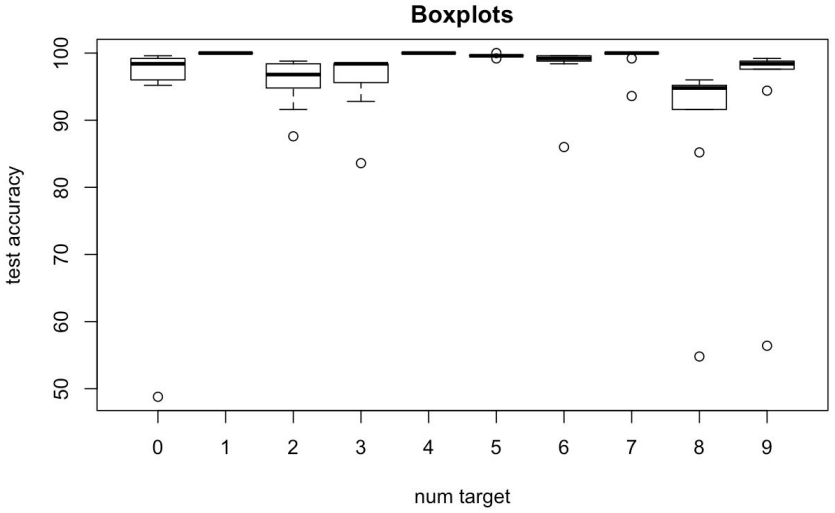


1.3 Performance of the algorithms

If the three algorithms are compared it's easy to see that SGM is the fastest algorithm with difference. The main problem is that having a fixed number of iterations causes it to usually not reach an stationary point, what means that the optimal values are not found. Although the difference between the SGM found values and the stationary points found by GM and QN-BFGD are usually small, it can possibly give the neural network less accuracy. It will be necessary to decide if this loss of accuracy is small enough to justify the speed gains of this algorithm.
The main difference between GM and QN-BFGS is easy to see observing their number of iterations. Although QN-BFGS iterations are more costous than GM's iterations, it's fastest in time and needs much less iterations.

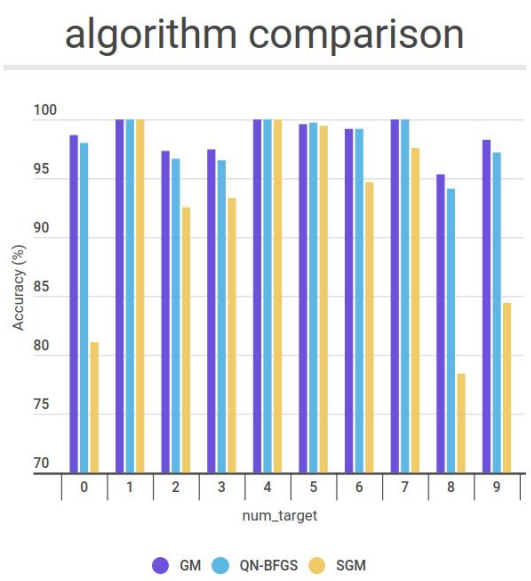
2. Recognition accuracy

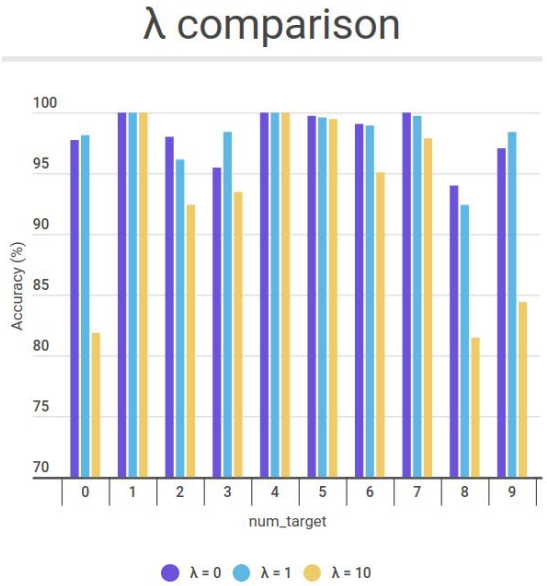
In this section it will be studied the digit identification accuracy of the neural network. The test accuracy will be analyzed in function of the regularization with parameter λ and the algorithm used. With this study it will be possible to identify digits that are specially difficult to identify. In order to have an intuition of the test accuracy for every target digit (num target) without considering the λ-algorithm combination this graph shows the boxplot for each different digit.



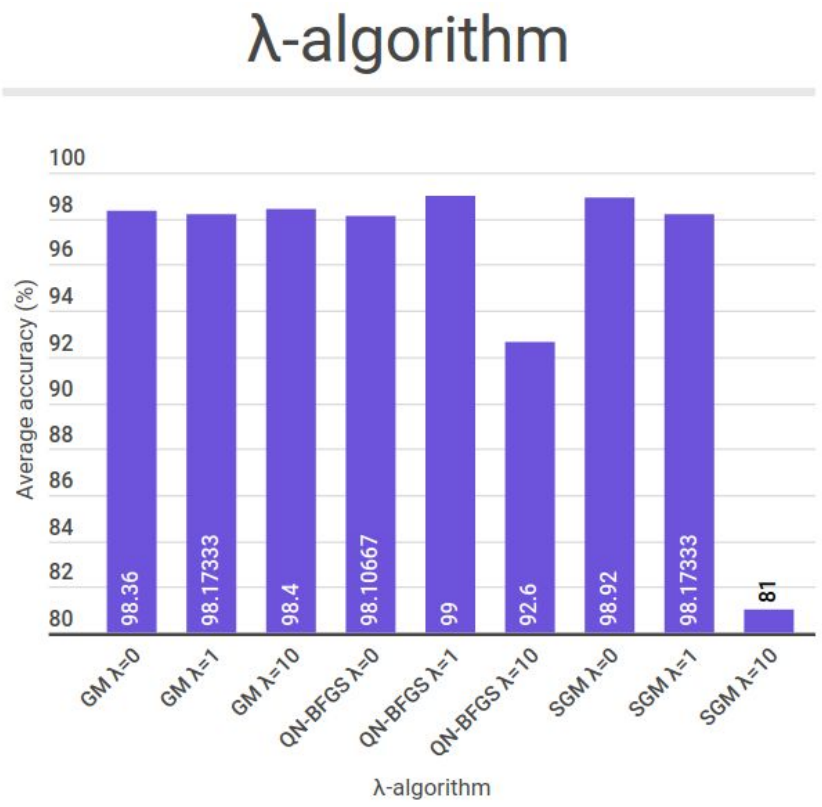
2.1 Best accuracy λ-algorithm

In the following plots, it's displayed the average test accuracy for every target number, grouped by algorithm used and by λ value. In the algorithm comparison plot, it's easy to see that SGM has worst accuracy in general. This is due to the algorithm not having a global convergence and the objective function not reaching a stationary, optimal value. On the other side, GM and QM-BFGS present very similar average accuracy in most of the cases, with very good performance. In the λ values plot, the accuracy for λ = 10 is by far the worst., with some really bad results in comparison of what can be achieved by the other values. λ = 1 and λ = 0 both have similar good performance, not being easy to decide which is better.





If we plot the average accuracy for every λ-algorithm it's obvious that SGM with λ = 10 has the worst performance with difference (having a 81% average test accuracy).
The rest of the algorithms, with exception of QN-BFGS with λ = 10 (92.6% accuracy), have similar good performance. By a small difference the best configuration is QN-BFGS with λ = 1 (99% accuracy).
By comparing this data, it can be observed that the bad performance of SGM in the algorithm comparison plot is due to it being really bad when used with λ = 10, but it has a good enough performance when used with the other λ values. In fact, when used with λ = 0 it has a value nearly as good as the best combination (QN-BFGS with λ = 1). Considering that SGM is much faster that QN-BFGS and the loss of performance is very small, it would be the best choice for the neural network.

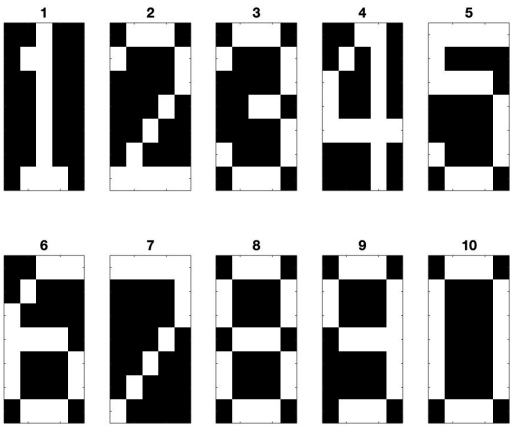


2.2 Difficult digit identification

We get the worst test accuracy when the target digit is 8 and we use the SGM and the lambda parameter is set to 10. We can check in the next plot that most numbers are false positives. The reason this happens is because most digits –when represented this way– are in some sense a subset of digit 8. In the last plot we see that numbers like 3, 6, 9 and 0 have a lot in common with 8. The reason it happens this way (numbers are misrecognized as 8) is because 8 contains most of the white pixels while other numbers contain more black pixels in some parts.

The predictions for the test data:

The 10 digits represented with binary pixels:



3. Conclusions

After the analysis of the data obtained with this experiments, we can conclude that the best option is using the QNM-BFGS method with $\lambda = 1$. As seen, this combination has the highest test accuracy and the lower execution time compared to the convergent cases. Although the results lead us to this conclusion there is still room for further experiments and combinations, we are not including this in the report because it is not requested, but changing the maximum number of iterations for the SGM algorithm could lead us to better performances and maybe it could improve (or at least get much close) to the QNM results of test accuracy.