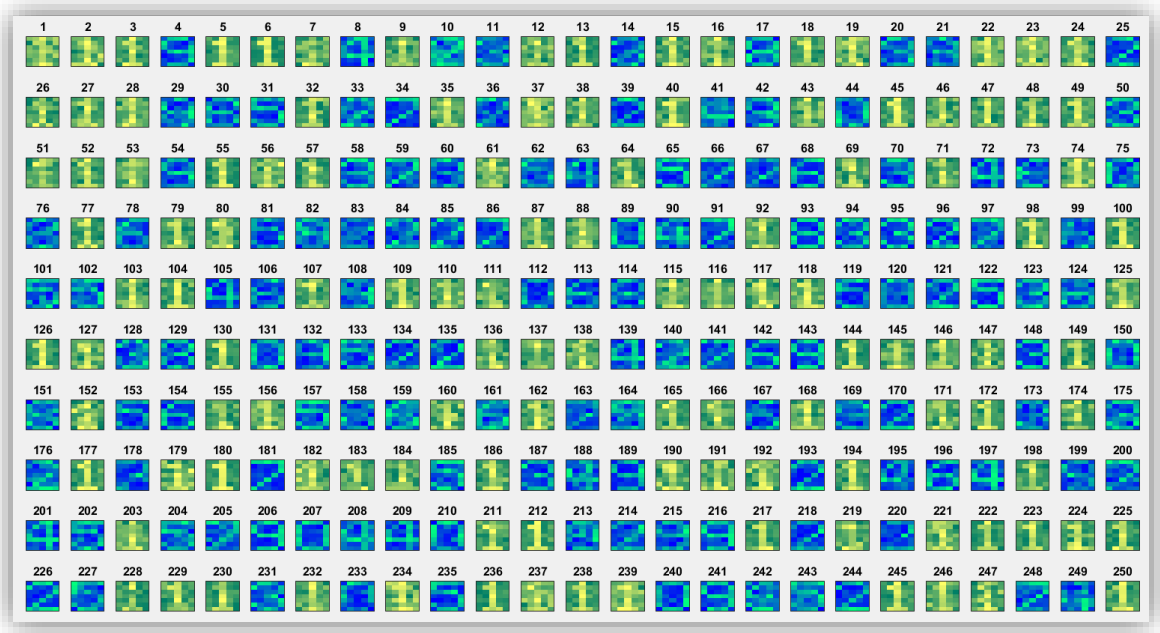
PATTERN RECOGNITION WITH SINGLE LAYER NEURAL NETWORK

SERGIO CÁRDENAS & ARMAND DE ASÍS



tr_seed = 47593233

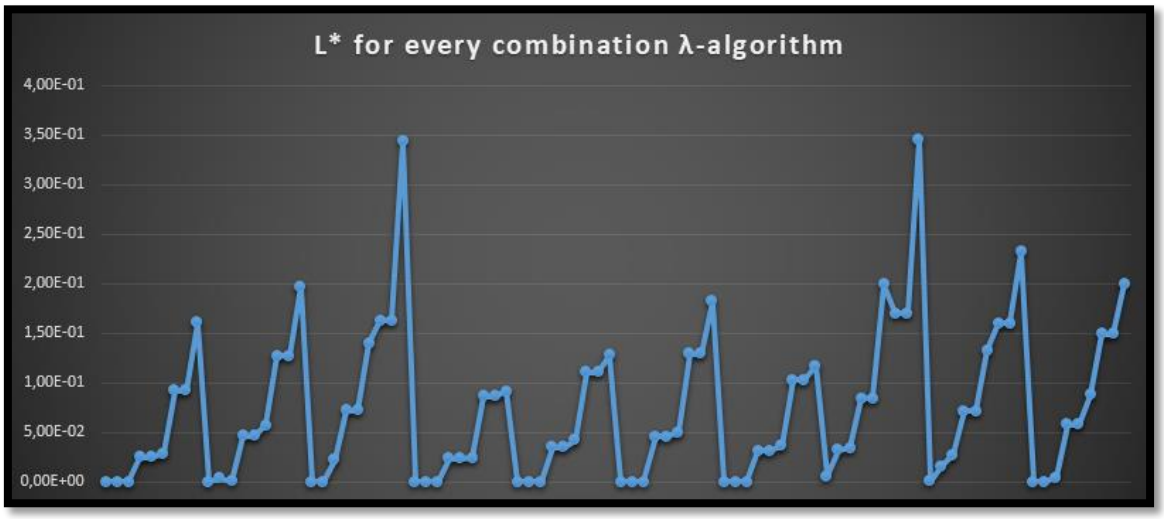
te_seed = 54175682

tg_seed = 47595417

STUDY OF THE CONVERGENCE

Global convergence

After running all instances and obtaining the results, we compare the value of L* for every combination λ-algorithm using the following graphic.

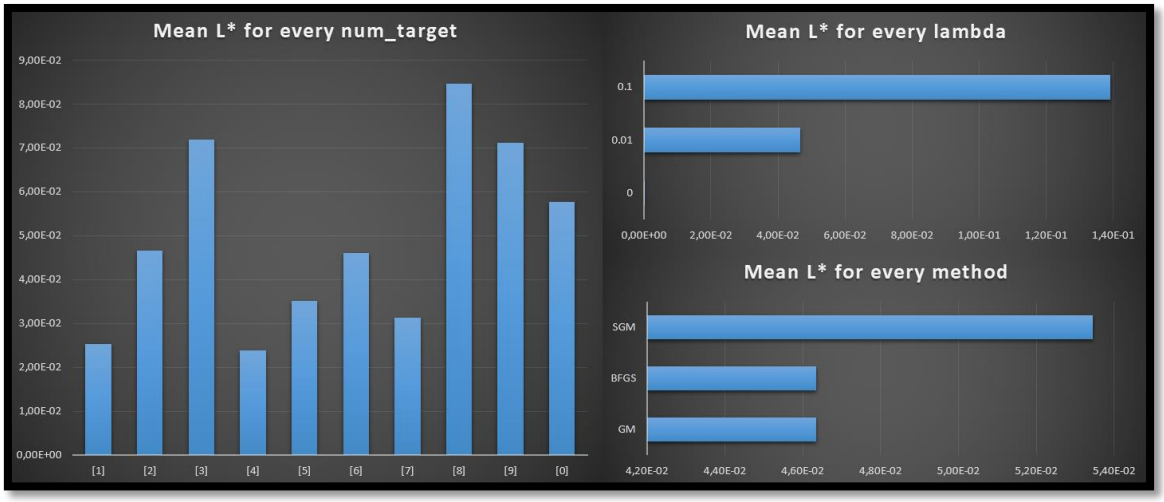


Graphic 1: L for every combination λ-algorithm*

The points in *Graphic 1* are ordered by num_target, λ and algorithm, so the first point represents [1] recognition for λ = 0 using GM, the second one [1] for λ = 0 using BFGS, the third [1] for λ = 0 using SGM, then [1] for λ = 0.01 using GM… and it follows, until the last point, which represents [0] recognition for λ = 0.1 using SGM.

At first sight, we can see that some numbers appear to be easier to recognise than the others. We can also clearly see that an increase of λ implies an increase on the L* value, due to the fact that λ makes the function more convex by adding an "error", making it easier to reach a solution, but losing confidence about the optimality of it. Finally, we can see from this graphic that the SGM algorithm seems to obtain worse results than the other ones. This can be caused by the random behaviour of the SGM due to its stochastic properties, because we find the *w* value for the next iteration using random permutations and subdivisions of the gradient, not necessarily a descent direction, which can leads us to a not optimal solution that, due to the incapability to improve the previous *w* values, makes us stop the algorithm. Therefore, we can't ensure SGM global convergence.

We can better observe all these results with the following graphics.



Graphic 2: Mean L for every num_target, λ and method*

As we said, it can clearly be seen from *Graphic 2* that not all numbers are recognised with the same precision ([1] and [4] are the easiest ones and [8] the most difficult), we obtain better results for L* when λ tends to 0, and SGM is not as precise as BFGS and GM.

According to theory, all three algorithms should converge to a stationary solution but, if we take a look to the information in the log file, we can see that some cases ([1], [8] and [9] using GM and [1], [4] and [7] using SGM, all with λ = 0) reach the maximum number of iterations for the GM and SGM algorithms. This doesn't have to mean they don't converge, because maybe they need a higher number of iterations to do it, but if they do converge, it will be on a very high number of iterations.

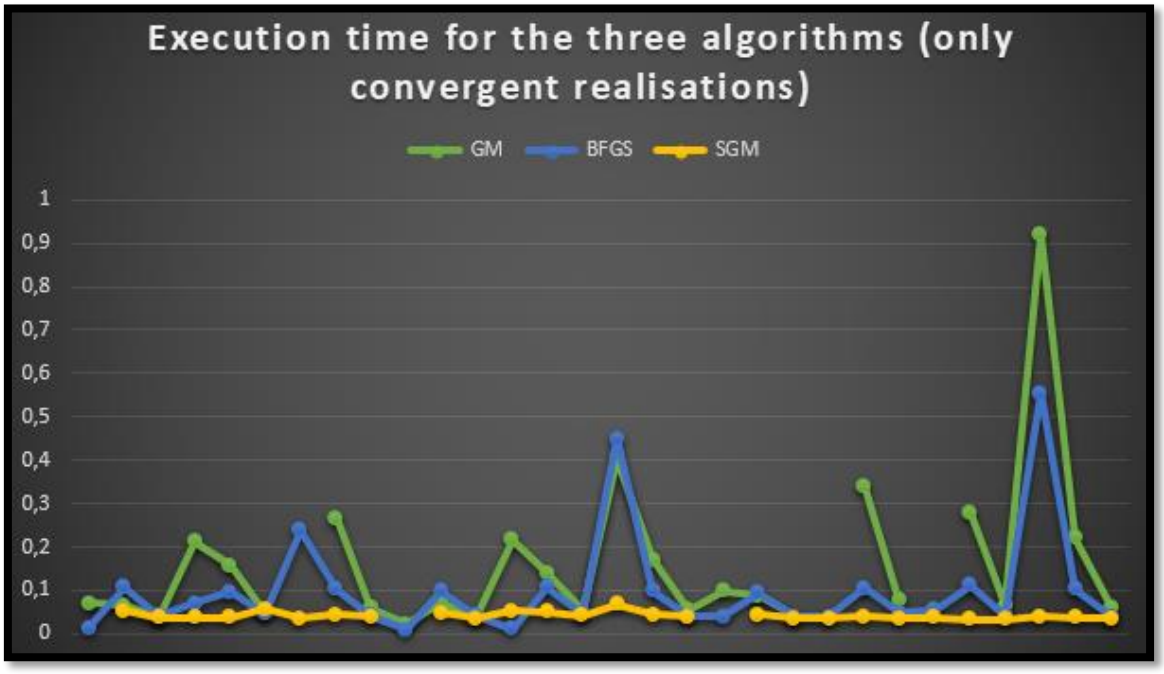
Local convergence

Let's compare the speed of convergence for the tree algorithms in terms of the execution time and number of iterations, using the following graphics.

At first, we have *Graphic 3*, which shows the execution time for the three algorithms, but if we take into account that, as we checked before, not all realisations for GM and SGM converge in the max number of iterations, we will notice that these ones make our graphic not really accurate, so we think it is better to plot just the convergent realisations.

Graphic 3: Execution time for the three algorithms

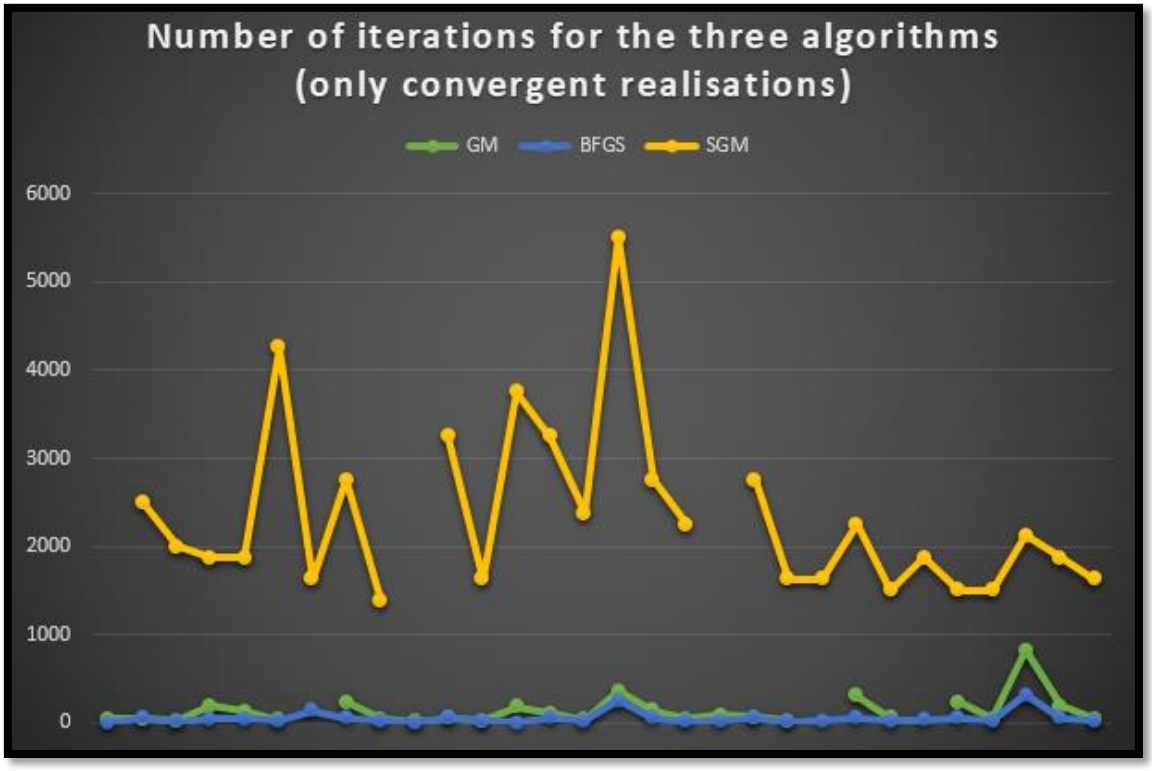
If we observe *Graphic 4*, we will see that SGM outperforms the other methods in terms of execution time, being faster and more constant. GM and BFGS have quite the same behaviour, but BFGS seems to be slightly better.



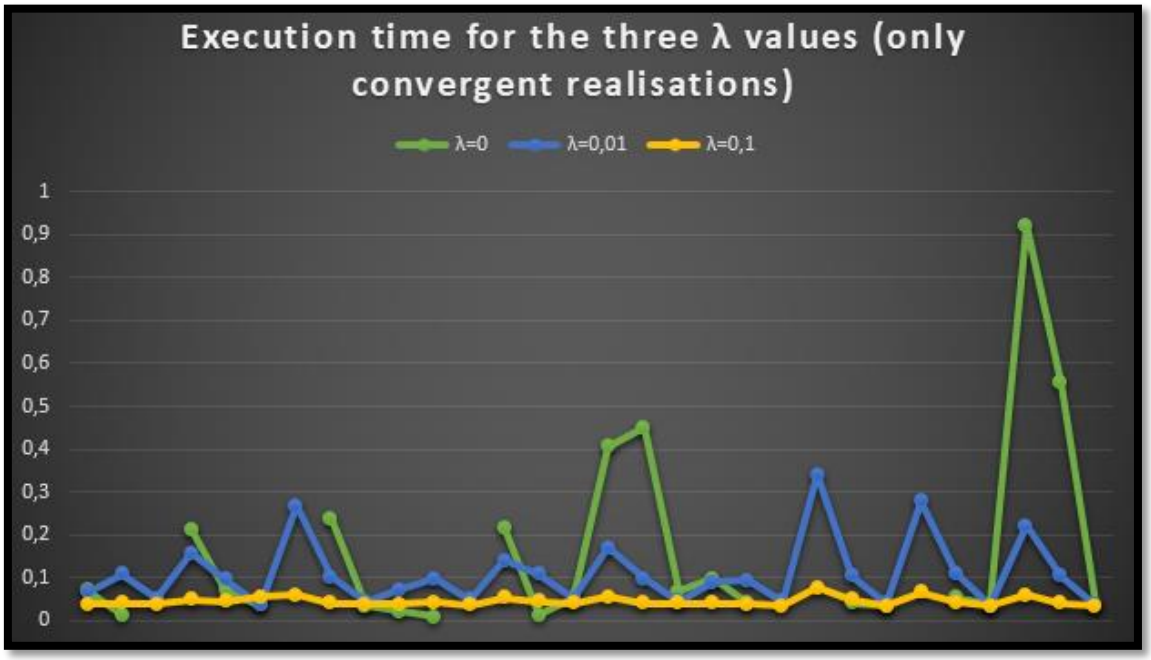
Graphic 4: Execution time for the three algorithms (only convergent realisations)

Now, we are going to compare the number of iterations for every algorithm, taking into account that we don't need the realisations that not converge.

From *Graphic 5*, it can be clearly seen that SGM is the one with the higher number of iterations, due to its epochs and minibatches structure, and also the one with a most variety of number of iterations, given by its random behaviour. GM and BFGS have quite similar number of iterations but, as in the execution time case, BFGS seems to have less.



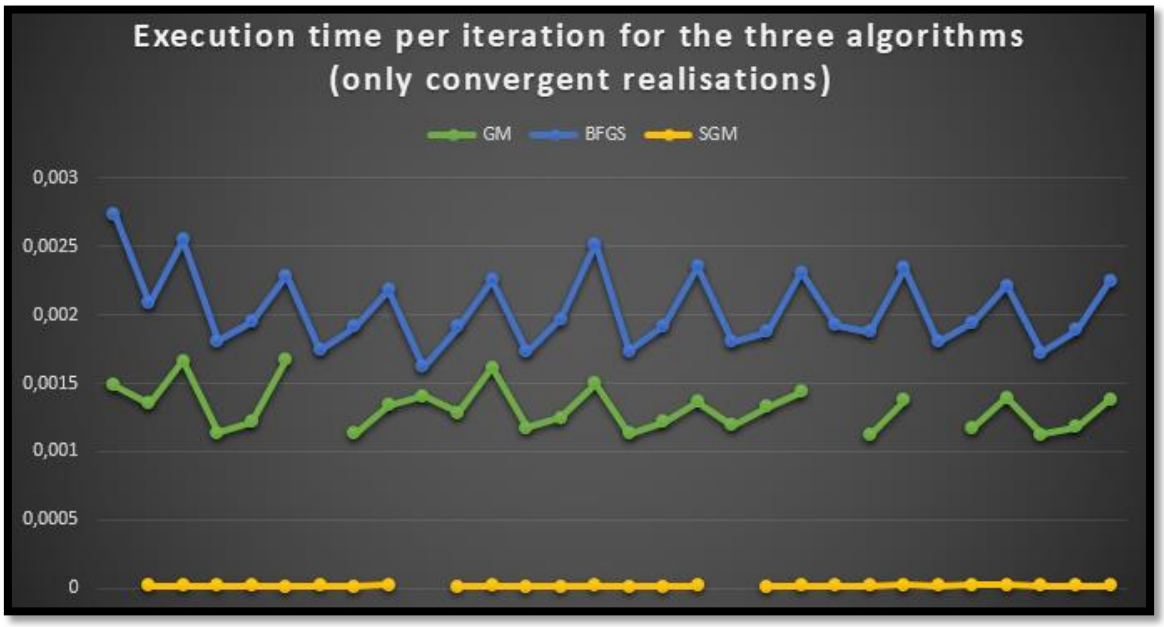
Graphic 5: Number of iterations for the three algorithms (only convergent realisations)



Graphic 6: Execution time for the three λ values (only convergent realisations)

With *Graphic 6* we can compare the execution time for the three λ values, observing that the fastest one is λ = 0.1. The other two cases are difficult to order, because λ = 0 makes it tricky, due to the fact that some of its realisations don't converge. Despite that, we could just say that λ = 0 and λ = 0.01, for convergent realisations, give us similar results. These results make sense because λ makes the function to minimize more convex, so it is easier to find a descend direction. It is logical, then, that the higher the λ value is, the lower the execution time of the algorithm is.

Using *Graphic 7*, we can analyse the running time per iteration for the three algorithms. At first, we can clearly see that SGM is the method with the lower running time per iteration, due to the fact that each SGM iteration gets *α* and *d* instantly. Then, it follows GM, with a much higher running time per iteration than SGM, caused by the cost of using line search to get *α* at each iteration, despite calculating *d* instantly too (this time using the whole vectors X^{TR} and y^{TR} to calculate the gradient). The slowest method in terms of running time per iteration is BFGS, which, as GM does, uses line search to calculate *α*, but, additionally, does not get *d* instantly, but using an auxiliary *H* matrix, which has to be calculated at each iteration, increasing the running time.

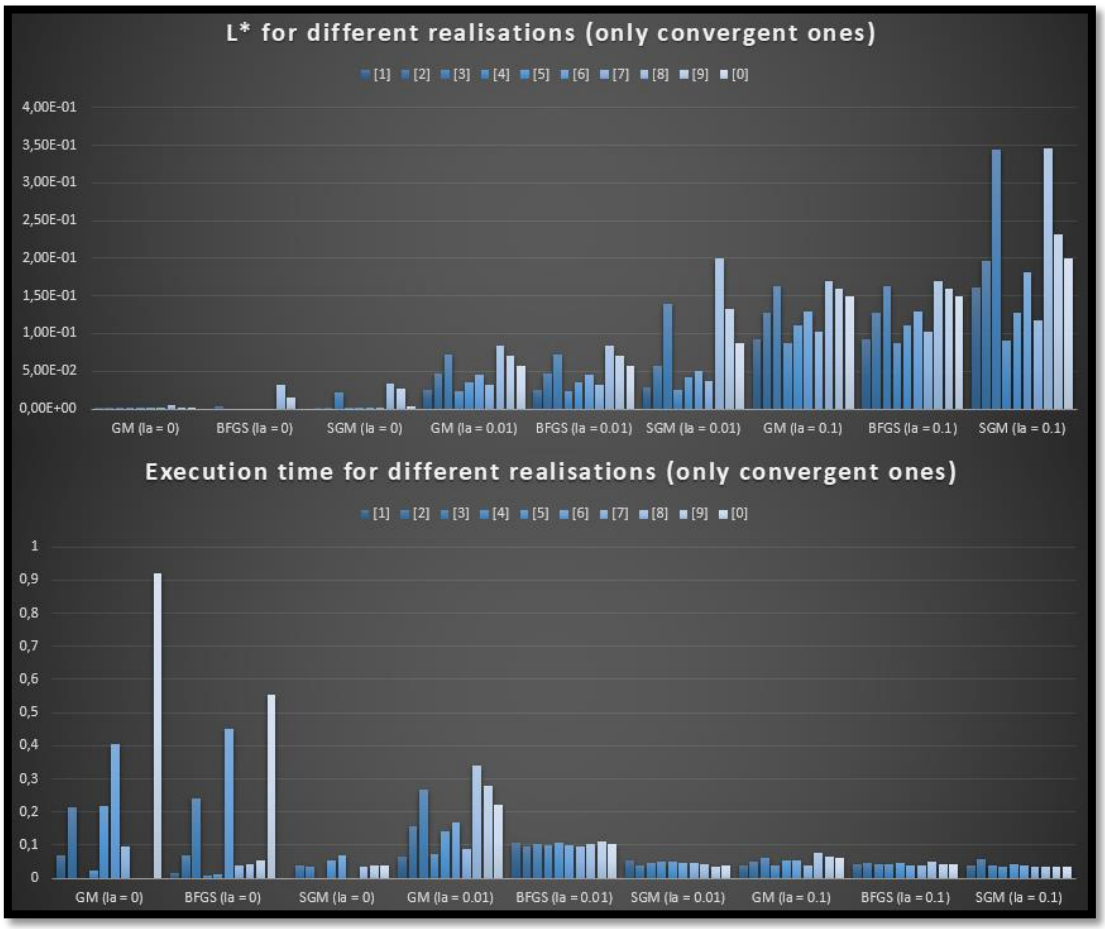


Graphic 7: Execution time per iteration for the three algorithms (only convergent realisations)

Discussion

For GM, we have seen that global convergence is not always ensured, especially for λ = 0, and local convergence is usually the worst one. BFGS has good global convergence properties for all λ value, and similar local convergence to GM. In terms of L*, both algorithms have the same behaviour, with near to 0 values. Finally, SGM clearly outperforms the other two algorithms in terms of local convergence and, despite not being as good as GM and BFGS for global convergence (having higher L* values), these difference is not quite significative and it still has decent properties.

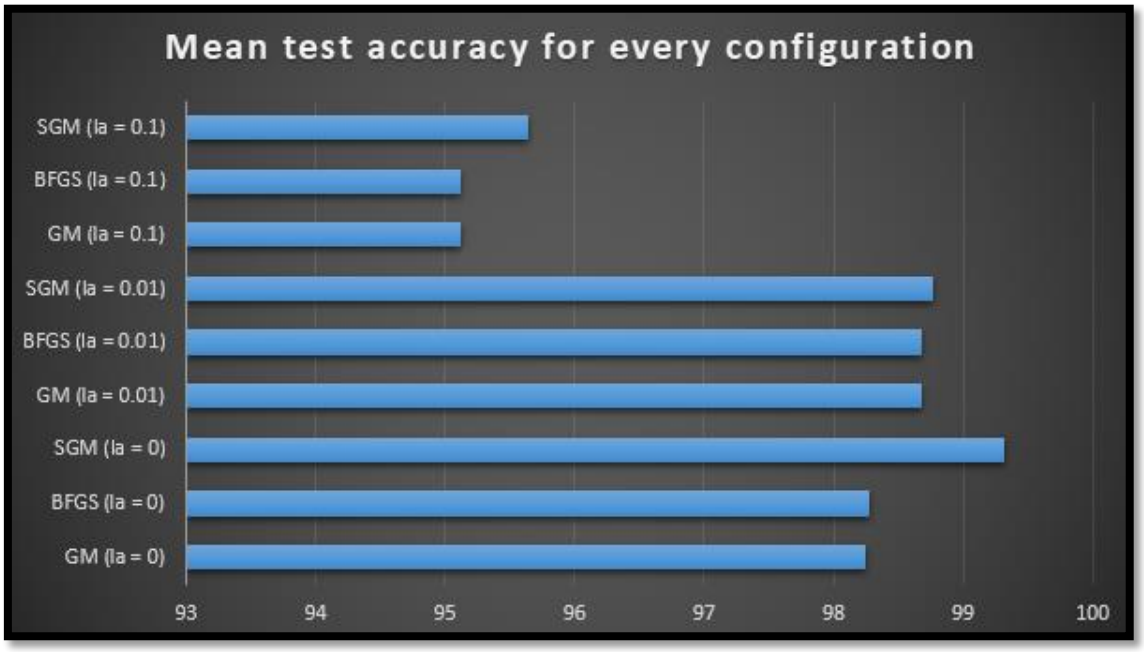
Having this information, we would choose λ = 0.01 for any algorithm, because it ensures the global convergence without adding as much error by shifting the optimal point as λ = 0.1 would do. Speaking of the algorithm, we would choose SGM, due to its local convergence outperforming, and because with λ = 0.01 the global convergence is ensured and the L* is not quite significant. So, the most efficient configuration for the minimization of L* would be SGM with λ = 0.01.



Graphic 8: L and execution time for different realisations (only convergent ones)*

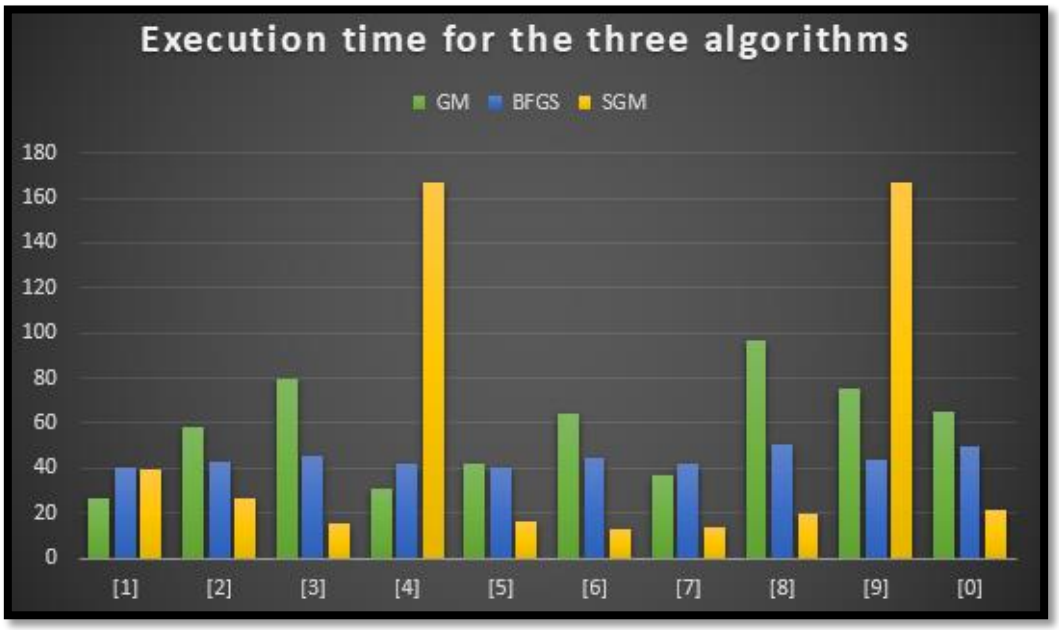
STUDY OF THE RECOGNITION ACCURACY

If we take a look at *Graphic 9*, we can see that the value of λ showing the best test accuracy in the results of the previous section is 0.01 for GM and BFGS and 0 for SGM, so we will focus on these three configurations for this part.



Graphic 9: Mean test accuracy for every configuration

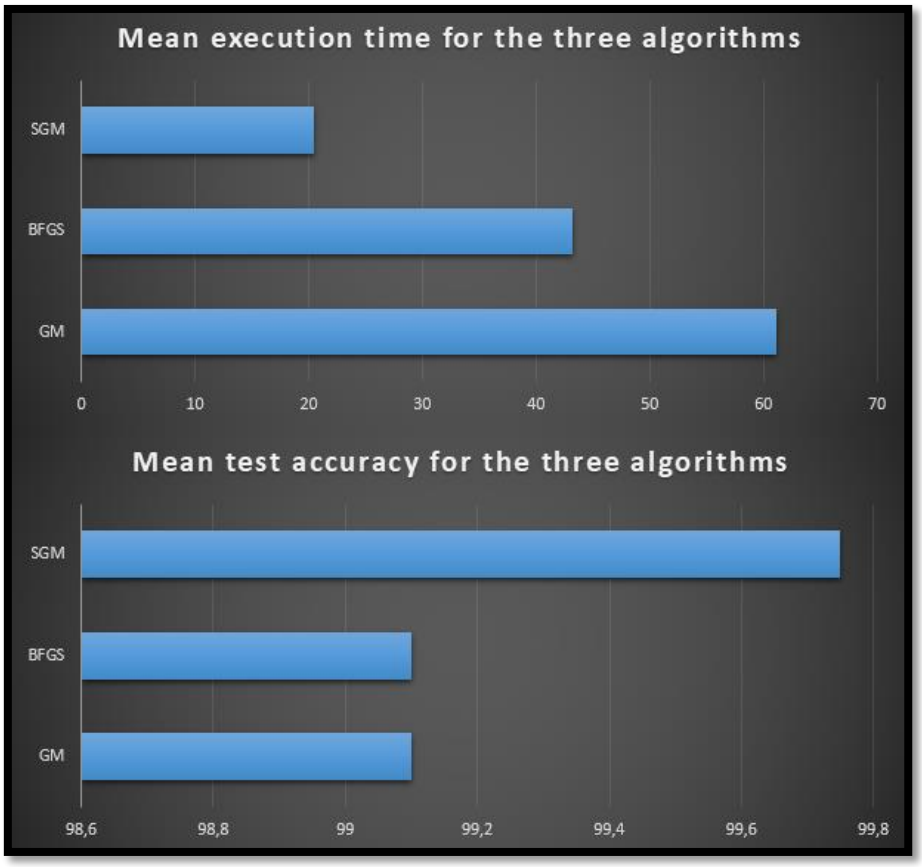
Best method for large problems



Graphic 10: Execution time for the three algorithms

After running the new realisations, we can see in *Graphic 10* that, in terms of execution time, GM and BFGS seem to have similar behaviour, but the second one performs a little better, and SGM outperforms the others, except in two cases, where the function does not converge. This is caused because the best λ value for SGM according to test accuracy was 0, which does not ensures global convergence.

To get a better idea of the behaviour of these executions, we are going to avoid divergence problems by working with mean values. So, if we take a look at *Graphic 11*, we can see that SGM clearly outperforms the others in terms of speed and recognition accuracy. Speaking of speed, it doubles and triples BFGS and GM, respectively, allowing us to increase the size of the problem (for instance, increasing the number of pixels) without having significant efficiency loses. This is caused by the scalability of the SGM algorithm, which works using minibatches (parts of the training data that obviously have less size) to estimate the gradient of the loss function, so it doesn't really need to work with the whole size of the training data. Because of that, SGM is an algorithm that can afford to work on complex and large size problems.



Graphic 11: Mean execution time and test accuracy for the tree algorithms

Discussion $Accuracy^{TE}$ – L*

The best combination λ-algorithm with respect to the maximization of test accuracy is SGM with λ = 0, and the best combination with respect to convergence and the minimization of L* is SGM with λ = 0.01, so both combinations coincide in algorithm but not in λ value.

To understand this discrepancy, we must remember the effect of λ in the loss function. This was the "convexification" of the loss function when we increase this hyperparametre, achieved by adding a penalization to the loss function when *w* value is high, which shifts the optimal solution. With this, we have achieved a better convexity of the function, which makes it easy to find a minimum, despite not being the original.

This is what causes test accuracy to fluctuate, because we are obtaining a close solution but not the same with respect to the original one, so it makes sense that, with this criteria, the chosen λ is 0.

From the other side, when we decided the best combination with respect to the minimization of L*, we took into account global and local convergence, and, in terms of local convergence, we saw that 0 was the worst value for λ. The best value for both convergences was 0.1, which ensured us to achieve a minimum in a fast way, but the error that caused was much higher than the ones that the other λ values did. So, the best option for us was λ = 0.01, that offered similar global and local convergence properties than λ = 0.1 but without obtaining a higher error value.

In general, no matter what the size of the problem is, if we want to obtain better test accuracy results, we would choose λ = 0, but if we want to ensure that all the realisations converge, despite losing some accuracy, we would choose λ = 0.01.

Another conclusion we can extract is that SGM is the best algorithm for this type of problems. For its structure, it can clearly be seen that it can afford large computational size problems on an efficient way, and its convergence properties are also quite good, so it is a wise option to choose SGM as the minimization algorithm for this problem.