Pattern recognition with Single Layer Neural Network. Results Report

GCED - Optimització Matemàtica
Maria Ribot Vilà & Àlex Martí Guiu

Train Seed = 1712
Test Seed = 2883

1. Introduction

The project consists on the creation of a single layer neural network and the use unconstrained optimisation techniques in order to recognize the numbers in a sequence of blurred grey-level digits. The procedure to achieve that will be the creation of a Single Layer Neural Network that will be trained to recognize the different numbers.

To do so, we will use first derivative optimization methods to minimize the loss function defined for a random training set, to afterwards test the results for a test set. This is the loss function (objective function) that we will minimize $(;_{tr},_{tr},) = \|_{tr},-y_{tr}\|^2 + \frac{\|w\|^2}{2}$ with its gradient $(;_{tr},_{tr},) = 2(_{tr})((((_{tr},)-_{tr})\circ(_{tr},)\circ(1-(_{tr},)))^T +$

Our goal is to minimize this Loss function which by definition is always convex so we can apply the theorems learned in class to study the order of convergence of a function. Three optimisation methods (first derivative methods to be more precise) have been chosen, which differ in their way of obtaining the descent direction:

- The **Gradient Method**:
  $d_g{}^k = -f^k$
- The **Quasi-newton Method**:
  ${}^k_Q = -(^k)^{-1}\,{}^k$, where the B matrix is an approximation to the Hessian matrix of the Loss function and is estimated iteratively through *BFGS update* using only first derivatives
- The **Stochastic Gradient Method**:
  We get an unbiased estimator of the gradient taking the average gradient of a minibatch of the training set (sampled at random iid). Therefore, the computation of the gradient of this sample will be much less time and resource consuming than the gradient of the full training set.
  $X_s{}^{TR} = [\,x_{s_1}{}^{TR},\, x_{s_2}{}^{TR},\, \dots ,x_{s_m}{}^{TR}]\,;\; y^{TR} = [\,y_{s_1}{}^{TR},\, y_{s_2}{}^{TR},\, \dots ,y_{s_m}{}^{TR}]$
  And the descent direction is computed from the previous consideration:
  ${}_k \leftarrow -(\,;\, {}_s{}^{TR}, {}_s{}^{TR},\, \lambda)$

2. Convergence

This is the result that we have obtained with our training and test seeds, having applied our optimization function with all of these combinations (a total of 90):

- For every one of the individual digits, 0 to 9.
- For every value of the regularization parameter $\lambda \in$ 0.0,1.0,10.0.
- For every optimization algorithm: GM, QNM and SGM.
- Having: isd=1 is the GM, isd = 3 is the QNM and isd=7 is the SGM

num; la; isd; niter; tex; tr_acc; te_acc; L*
1; 0.0; 1; 3; 0.0773; 100.0; 100.0; 1.65e-19;
1; 0.0; 3; 3; 0.1648; 100.0; 100.0; 2.53e-19;
1; 0.0; 7; 20; 0.0806; 100.0; 100.0; 1.37e-03;
1; 1.0; 1; 94; 0.6200; 100.0; 100.0; 3.44e+00;
1; 1.0; 3; 21; 0.1245; 100.0; 100.0; 3.44e+00;
1; 1.0; 7; 1000; 0.0736; 100.0; 100.0; 1.10e+01;
1; 10.0; 1; 49; 0.1822; 100.0; 100.0; 1.41e+01;

1; 10.0; 3; 38; 0.1707; 100.0; 100.0; 1.41e+01;
1; 10.0; 7; 1000; 0.0891; 99.6; 99.2; Inf;
2; 0.0; 1; 34; 0.2032; 94.8; 94.4; 1.30e+01;
2; 0.0; 3; 1000; 4.0694; 94.0; 94.4; 1.46e+01;
2; 0.0; 7; 51; 0.0375; 99.2; 99.6; 1.18e+00;
2; 1.0; 1; 292; 0.6877; 99.6; 99.2; 8.34e+00;
2; 1.0; 3; 21; 0.1401; 99.6; 99.2; 8.34e+00;
2; 1.0; 7; 1000; 0.0825; 94.4; 94.4; 2.05e+01;
2; 10.0; 1; 80; 0.3140; 95.6; 96.0; 2.31e+01;
2; 10.0; 3; 38; 0.1959; 95.6; 96.0; 2.31e+01;
2; 10.0; 7; 1000; 0.0921; 98.4; 99.2; Inf;
3; 0.0; 1; 759; 1.9755; 100.0; 97.6; 1.93e-06;
3; 0.0; 3; 1000; 4.0034; 99.6; 97.2; 1.00e+00;
3; 0.0; 7; 97; 0.0411; 92.4; 92.4; 1.49e+01;
3; 1.0; 1; 493; 1.5339; 98.4; 98.8; 1.22e+01;
3; 1.0; 3; 22; 0.1606; 98.4; 98.8; 1.22e+01;
3; 1.0; 7; 1000; 0.0897; 96.4; 97.6; 2.93e+01;
3; 10.0; 1; 117; 0.3620; 97.2; 98.8; 3.15e+01;
3; 10.0; 3; 37; 0.1619; 97.2; 98.8; 3.15e+01;
3; 10.0; 7; 1000; 0.0705; 54.4; 52.0; Inf;
4; 0.0; 1; 3; 0.0461; 100.0; 100.0; 2.36e-14;
4; 0.0; 3; 3; 0.0525; 100.0; 100.0; 1.94e-14;
4; 0.0; 7; 9; 0.0458; 99.2; 99.6; 9.47e-01;
4; 1.0; 1; 111; 0.3143; 100.0; 100.0; 3.47e+00;
4; 1.0; 3; 20; 0.1142; 100.0; 100.0; 3.47e+00;
4; 1.0; 7; 1000; 0.0963; 100.0; 100.0; 1.10e+01;
4; 10.0; 1; 60; 0.2391; 100.0; 100.0; 1.40e+01;
4; 10.0; 3; 33; 0.2586; 100.0; 100.0; 1.40e+01;
4; 10.0; 7; 1000; 0.1440; 93.2; 94.8; Inf;
5; 0.0; 1; 2; 0.0455; 100.0; 99.6; 3.08e-12;
5; 0.0; 3; 2; 0.0787; 100.0; 99.6; 3.08e-12;
5; 0.0; 7; 9; 0.0887; 100.0; 99.6; 6.58e-03;
5; 1.0; 1; 175; 0.8939; 100.0; 99.6; 4.76e+00;
5; 1.0; 3; 19; 0.1448; 100.0; 99.6; 4.76e+00;
5; 1.0; 7; 1000; 0.0828; 100.0; 100.0; 1.53e+01;
5; 10.0; 1; 88; 0.6620; 100.0; 99.6; 1.85e+01;
5; 10.0; 3; 40; 0.4742; 100.0; 99.6; 1.85e+01;
5; 10.0; 7; 1000; 0.0982; 88.8; 88.0; Inf;
6; 0.0; 1; 14; 0.1059; 100.0; 98.8; 5.80e-08;
6; 0.0; 3; 11; 0.2288; 100.0; 100.0; 1.57e-09;
6; 0.0; 7; 36; 0.0562; 100.0; 99.2; 3.88e-02;
6; 1.0; 1; 236; 0.7663; 100.0; 99.6; 6.29e+00;
6; 1.0; 3; 22; 0.4421; 100.0; 99.6; 6.29e+00;
6; 1.0; 7; 1000; 0.1345; 100.0; 99.6; 1.90e+01;
6; 10.0; 1; 99; 0.5335; 100.0; 100.0; 2.20e+01;
6; 10.0; 3; 37; 0.2600; 100.0; 100.0; 2.20e+01;
6; 10.0; 7; 1000; 0.0950; 56.8; 51.6; Inf;
7; 0.0; 1; 17; 0.1173; 100.0; 100.0; 8.34e-08;
7; 0.0; 3; 10; 0.0901; 100.0; 100.0; 3.07e-07;
7; 0.0; 7; 10; 0.0442; 100.0; 100.0; 3.59e-02;
7; 1.0; 1; 142; 0.5109; 100.0; 100.0; 4.39e+00;
7; 1.0; 3; 21; 0.1933; 100.0; 100.0; 4.39e+00;
7; 1.0; 7; 1000; 0.0867; 100.0; 100.0; 1.30e+01;
7; 10.0; 1; 52; 0.3149; 100.0; 100.0; 1.61e+01;
7; 10.0; 3; 39; 0.2193; 100.0; 100.0; 1.61e+01;
7; 10.0; 7; 1000; 0.0992; 94.8; 96.8; Inf;
8; 0.0; 1; 1000; 3.1805; 100.0; 95.2; 4.45e-01;
8; 0.0; 3; 19; 0.1065; 99.6; 92.8; 1.00e+00;
8; 0.0; 7; 234; 0.0671; 98.0; 96.8; 4.02e+00;
8; 1.0; 1; 685; 1.7605; 97.6; 96.8; 1.53e+01;
8; 1.0; 3; 30; 0.1677; 97.6; 96.8; 1.53e+01;
8; 1.0; 7; 1000; 0.0790; 84.8; 84.8; 3.33e+01;
8; 10.0; 1; 1000; 1.9629; 97.2; 96.8; 3.49e+01;
8; 10.0; 3; 35; 0.1721; 97.2; 96.8; 3.49e+01;
8; 10.0; 7; 1000; 0.0790; 68.8; 68.4; Inf;
9; 0.0; 1; 945; 2.6344; 100.0; 99.2; 1.89e-06;
9; 0.0; 3; 1000; 4.1077; 99.2; 98.0; 1.87e+00;
9; 0.0; 7; 111; 0.0525; 98.0; 98.0; 2.66e+00;
9; 1.0; 1; 452; 1.2256; 99.6; 98.4; 1.15e+01;
9; 1.0; 3; 24; 0.1449; 99.6; 98.4; 1.15e+01;
9; 1.0; 7; 1000; 0.0766; 96.4; 98.8; 2.77e+01;
9; 10.0; 1; 130; 0.3998; 96.8; 99.2; 3.03e+01;
9; 10.0; 3; 43; 0.2022; 96.8; 99.2; 3.03e+01;
9; 10.0; 7; 1000; 0.0792; 67.6; 70.4; Inf;
0; 0.0; 1; 713; 1.9675; 100.0; 96.8; 1.51e-06;
0; 0.0; 3; 15; 0.0906; 100.0; 94.4; 1.98e-12;
0; 0.0; 7; 55; 0.0436; 100.0; 97.6; 2.15e-01;
0; 1.0; 1; 334; 0.8350; 100.0; 97.6; 7.89e+00;
0; 1.0; 3; 27; 0.1510; 100.0; 97.6; 7.89e+00;
0; 1.0; 7; 1000; 0.0833; 99.6; 98.8; 2.24e+01;
0; 10.0; 1; 114; 0.4599; 100.0; 98.0; 2.52e+01;
0; 10.0; 3; 39; 0.2172; 100.0; 98.0; 2.52e+01;
0; 10.0; 7; 1000; 0.0905; 52.0; 52.0; Inf;

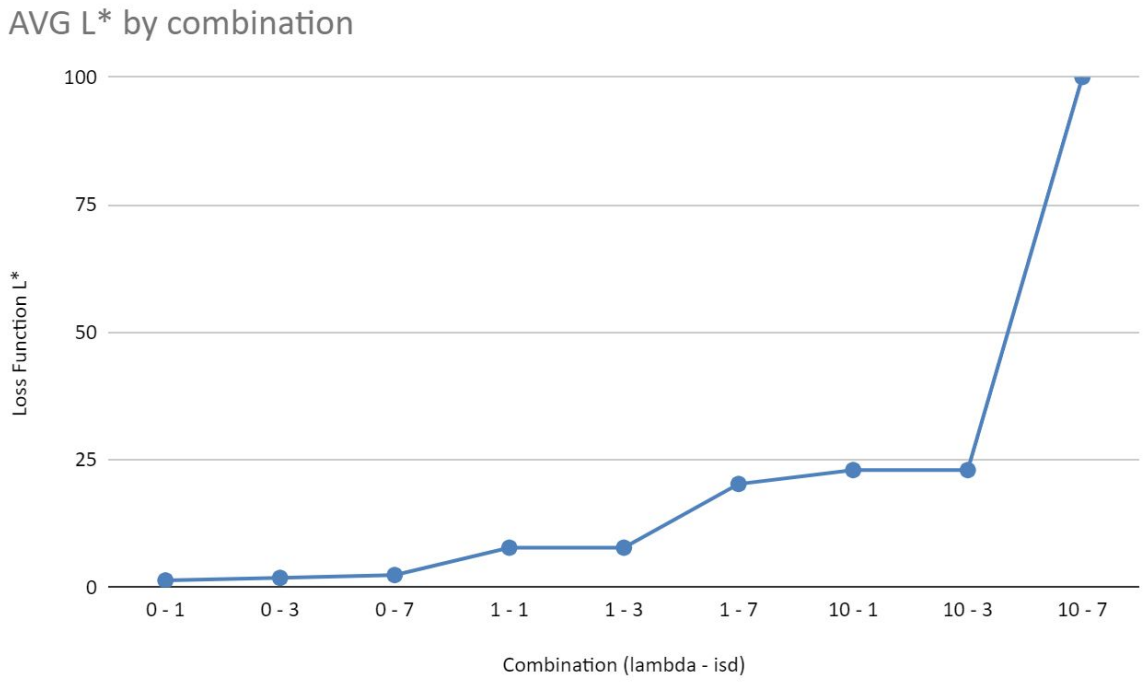
- **Global Convergence:**

As a general matter we say that an algorithm achieves global convergence if $\{x^k\} \rightarrow_{k \rightarrow \infty} x^*$ tends to a stationary point. We have a maximum number of iterations (kmax = 1000) for each instance to find said point and we establish that the algorithm doesn't converge when no minimizer is found before those maximum 1000 iterations (ie. if niters=1000 → doesn't converge globally).

The results show that each combination may change in terms of global convergence depending on the instance but they all tend to either converge or not in most cases. Once these nuisances are noted, we can conclude whether each of the algorithms with different lambda parameters converge globally or not :

Lambda (λ) / isd	1	3	7
0	Yes (1 exception)	Yes (3 exceptions)	Yes
1	Yes	Yes	No
10	Yes	Yes	No

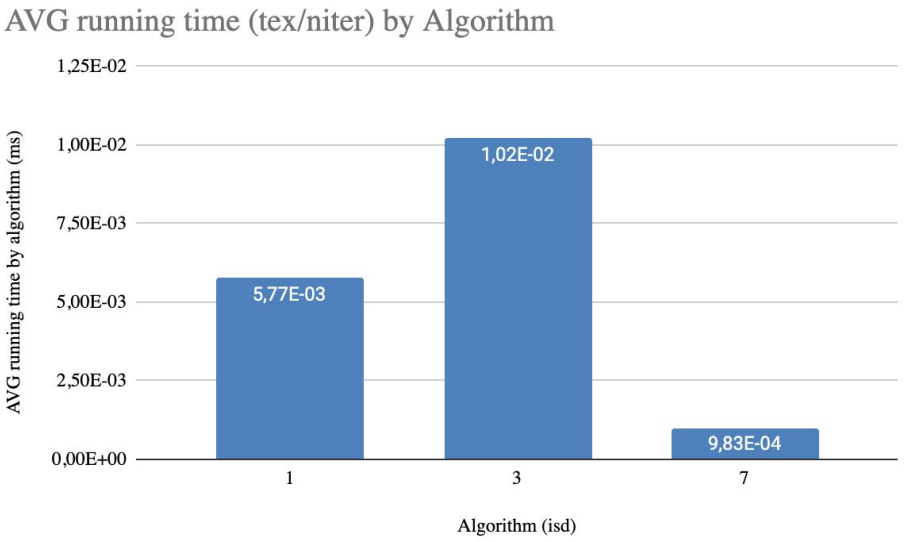
Those are the average values of the loss function L* evaluated at the minimizer point found with each combination:

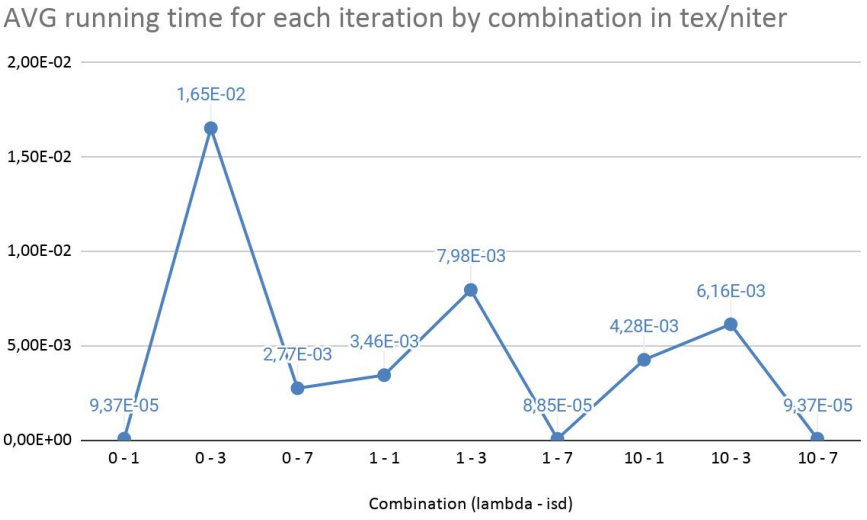


The infinite given by the matlab solver has been represented as a 100 in order to be able to show the behaviour of each combination studied which is the case of L* for the combination of λ=10 and isd=7=SGM. The interpretation of this graphic is that the lower this L* is for the combination, the better the algorithm performs with it and therefore it makes less classification errors when using it. As previously stated two algorithms find a stationary solution (converge), the 1 and 3 and the number 7 finds the solution only with lambda equal to zero.

- **Local Convergence:**

Studying the speed with which the algorithms converge we can establish information about local convergence. We can say that the nearest to a higher order of convergence are, the nearest they are to a lower runtime score. This however does not mean that it converges as we for example see the instances of the algorithm 7 (SGM) do not converge, but the computation of the results is much quicker, so its mean running time appears to be much better than the other algorithms that converge (as we have seen, this is due to the less cost of the computations of the gradient of the minibatch). We have calculated the average runtime of each algorithm and also of each combination, using time_execution(tex)/niters which is in *ms*:





Looking at the results, we can first see how the SGM has a very low runtime in average and also for all combinations for the reasons explained before. The gm has a higher average runtime, due to the gradient computation from the whole training dataset that has quite high cost. However, the λ=0 and isd=GM combination gives excellent results (the best) in terms of runtime (and also in terms of accuracy which will see in the next graphics). Finally, the QNM has an even higher runtime, due to the very high cost of the BFGS update matrix computations needed at each iteration of the algorithm that is used to calculate the descent direction, adding it cost of the gradient (same as GM).

If we were to calculate the rate of convergence we would be able to say for sure the order of convergence of the other algorithms, either linear or superlinear. We would not be able to find quadratic order of convergence as all the methods used are first derivative methods thus quadratic convergence cannot be met.

- **General analysis**

After analysing the local and global convergence we can discuss the general performance of the three algorithms. We can say whether a method of optimisation with certain conditions converges and additionally we can also compute the speed and the time it takes to converge. We have seen how the GM and QNM have behave very similar in terms of loss but the second one has higher runtimes, while the SGM has very low runtimes but does not guarantee convergence (despite having quite good accuracies with lambdas 0 and 1). Thus we can conclude that the proper method to minimize the loss function of the given problem is the GM, taking the lambda parameter as zero.

3. Accuracy

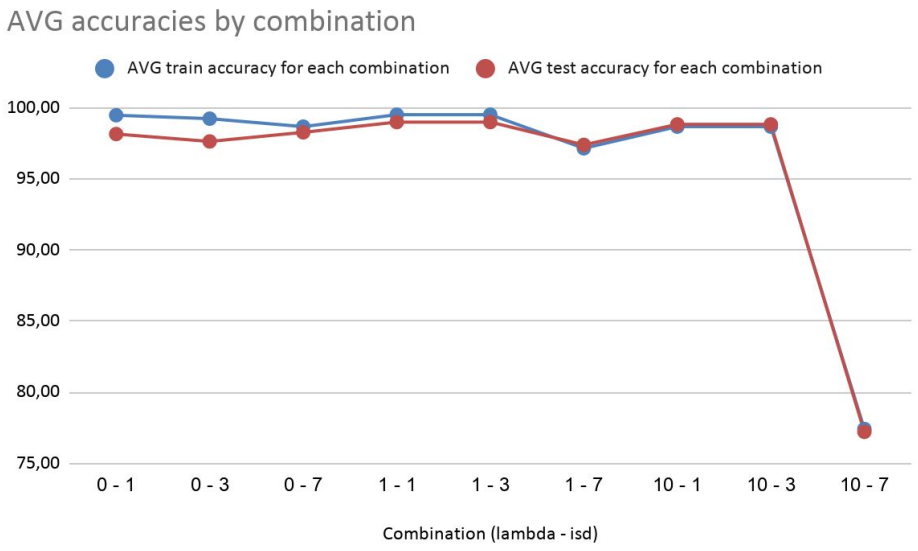
We have created two sets with our seeds, the training one which has been used to train the model (the w coefficients), and the test set which will be used to test this model and see how it fits data on which it hasn't been trained. Therefore, we will consider our neural network (model) has overfitting if the training accuracy is much greater than the test accuracy. In terms of the accuracy of our function, we have calculated for each execution the train and test accuracy given by the formula:

$$Accuracy_{TE} = \frac{100}{p} * \sum_{j=1}^{p} [y(x_j^{TE}, w^*)], y_j^{TE}$$

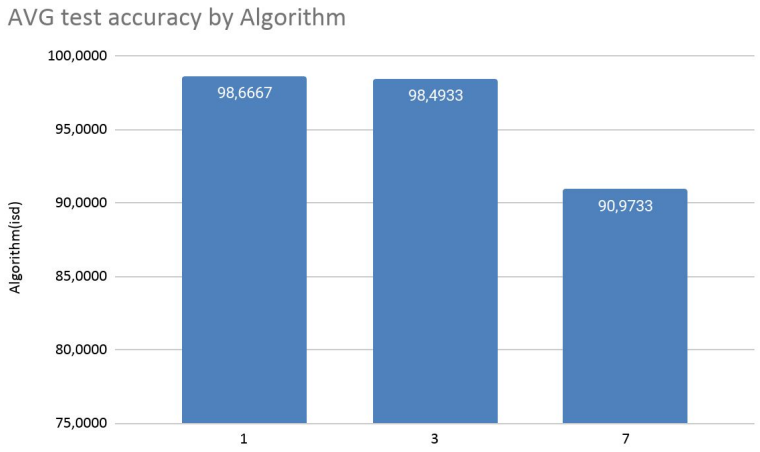
Having seen the results of our executions in terms of these accuracies, our first firm conclusion is that our neural network doesn't have overfitting, as in all cases there is not a large difference between the test and training accuracy. In most cases they are very similar as well as high which shows us that the model works quite well on average.

The total average training accuracy of our function after all 90 runs is: 96,48888889
The total average test accuracy of our function after all 90 runs is: 96,04444444

a) To analyze the performance of the different combinations of λ-algorithm we have plotted the average accuracies, and we will focus on the test accuracy:

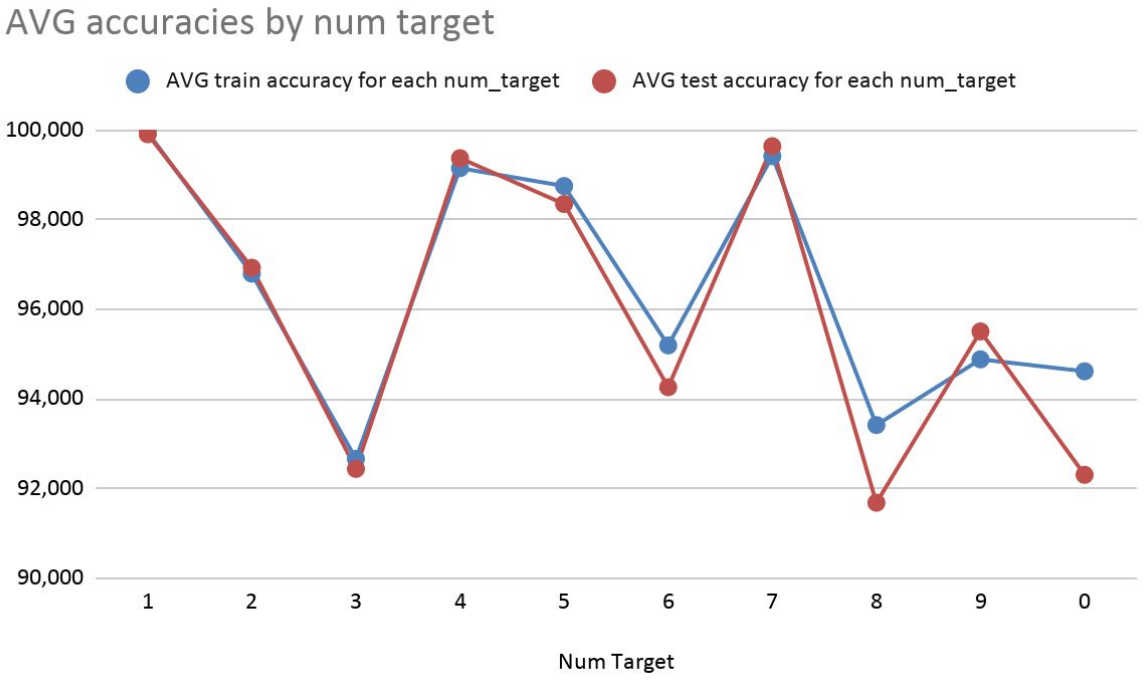


As we can see, the test accuracy is quite high for most combinations (around 97-98%) but it is quite lower for the combination **λ=10 and isd=7=SGM** (it has exactly 77,24% test accuracy), which indicates us that this combination is the only one that fails to give a proper recognition. Having seen this and the previous runtime plots, according to our numerical experiments, the recommended λ-algorithm combination terms of, both, the recognition capacity and the speed of convergence is **λ=0 and isd=1=GradientMethod** as it has the lower runtime (9,37E-05 ms) and also the highest test accuracy (99,911%).



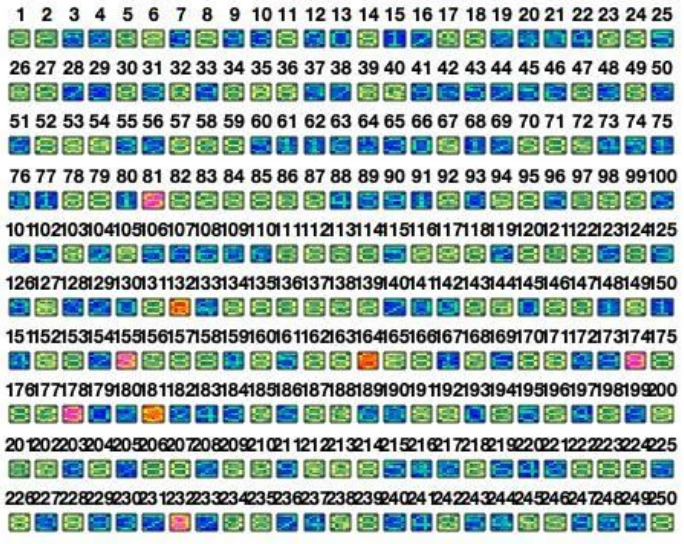
With this plot we can also see how the most accurate algorithm independently from the lambda combination is still the GM, being very close to QNM. However, all 3 algorithms present a very good test accuracy that is above 90%.

b) To find if there is any digit specially difficult to identify, regardless of the λ-algorithm combination we will calculate the averages accuracies for each target number:



As we can see, the hardest numbers to identify are 8, 3 and 0 as they are the ones that have both lower train and specially test accuracies. It is also interesting to highlight how the results obviously show lower test accuracies, as they are the sets on which the data hasn't been trained.

Finally, we have plotted the results for the num_target = 8, which is the number with worst average test accuracy, and we have used the λ=10 and isd=3=QNM for it →



The reason why 8 has such bad results is that the number has a very similar shape to many other numbers (specially 3, 0, 6 and 9) and has many curves (it's not as simple as numbers like 7 or 1 which are very straightforward to draw and therefore easier to recognize) and that makes it harder for our UO Algorithms to recognize.

4. Conclusion

We have seen that using different methods for minimizing the loss function of the single layer neural system we achieve different outcomes for its value at the optimum L*. That gives information about the ability of the method to train the neural system to work towards its goal of minimizing the loss and therefore minimizing the rate of misclassifications.
With our trials we have seen that the most efficient method is the GM given the lambda parameter as zero. It seems as if it doesn't converge for num_target = 8, but we can justify that this fact does not affect the overall performance as we have said that the number eight is difficult to train due to the similarities to other numbers. We can also see that the minimum L* reached is very close to zero (A mean L* of 1,344500547, the lowest amongst all) and its accuracy very is close to 100 in both test and training.

As a final note if the goal were to find an efficient method to minimize the loss function in terms of speed (low runtime) and low use of resources the SGM might be suitable when choosing the lambda parameter as 0 or 1 (not 10 as it gives L* = inf). Despite it doesn't converge in most cases with the maximum 1000 iterations the loss function has a very small value at the end. Given that we do not have constraints about the application further than the exercise we have chosen the most efficient overall (GM) as SGM in most cases is very efficient but doesn't converge.

Having done this projects, we have learnt a little bit more about neural networks and how the optimization techniques and methods learned in the theory lessons can be applied in real case

examples to train a neural network and use the optimum weights w* to make predictions for the test set.