Single Layer Neural Network (SLNN)

Pattern recognition

Carlos Arbonés & Joel Solé
Universitat Politècnica de Catalunya, Barcelona
April 16, 2023 (seeds in table 4)

ABSTRACT

The aim of this project is to develop a single layer neural network for pattern recognition tasks. The network will be trained using a loss function to minimize the error between the predicted output and the actual output. The project will also focus on analyzing the global and local convergence of the algorithms used to train the network. The network will be implemented using Matlab and its performance will be evaluated on a dataset of images for pattern recognition. The project will involve a detailed analysis of the network's performance, including its accuracy and the time taken to train the network.

1 INTRODUCTION

Pattern recognition is a fundamental area of research in computer science, with many real-world applications such as image classification and speech recognition. One popular approach for developing pattern recognition systems is to use neural networks. Neural networks have been successful in solving many complex pattern recognition problems, and their performance can be further improved by optimizing their weights using various optimization algorithms.

In this project, we will be developing a pattern recognition system using a single layer neural network. The goal is to train the network using unconstrained optimization algorithms, namely gradient, BFGS, and stochastic gradient methods. We will be measuring the global and local convergence of the algorithms used to optimize the network's weights, and studying their accuracy for large problems.

The project will involve a comprehensive analysis of the performance of the network, including its accuracy and time taken to train the network. We will be working with large datasets to evaluate the network's performance, and the results obtained will contribute to the development of more efficient and accurate pattern recognition systems.

Overall, this project will provide valuable insights into the use of neural networks for pattern recognition, and the optimization algorithms used to train them.

2 METHOD

Digits are defined as a grid of 7 rows and 5 columns (35 pixel values), arranged in a vector, where each value (pixel) of the digit is an input used in the single layer neural network.

- *Input signal (encoded digit):*

$$I_i = x_i, \quad i = 1, 2, \cdots, n \qquad I_{n+1} = \sum_{i=1}^{n} w_i \cdot O_i$$

An activation function is also defined to introduce non-linearity into the output allowing the network to capture more complex patterns and relationships in the data.

- ***Activation function (Sigmoid)***

$$O_i = \sigma(I_i), \qquad \sigma(x) = \frac{1}{(1 + e^{-x})}$$

- ***Output signal (recognition)***

$$y(x, w) = \sigma(I_{n+1}) = \sigma\left(\sum_{i=1}^{n} wi \cdot O_i\right) = \sigma\left(\sum_{i=1}^{n} w_i \cdot \sigma(x_i)\right)$$

In an ideal situation:

$$y(x, w) = \begin{cases} 1 & \text{if } x \text{ is the target digit} \\ 0 & \text{if } x \text{ is not the target digit} \end{cases}$$

2.1 Training

In the training of the SLNN, an objective function has been defined which needs to be minimized. This objective function is the *Mean Loss Function*, denoted by $L(w; X^{TR}, y^{TR})$, where it has been incorporated L^2 regularization into the optimization process. Hence, the optimized objective function with L^2 regularization is given by:

$$\tilde{L}(w; X^{TR}, y^{TR}, \lambda) = L(w; X^{TR}, y^{TR}) + \frac{\lambda}{2} \cdot \|w\|^2$$

where

$$L(w; X^{TR}, y^{TR}) = \frac{1}{p} \sum_{j=1}^{p} (y(x_j^{TR}, w) - y_j^{TR})^2$$

2.2 Test

To evaluate the generalization performance of the SLNN we will measure its accuracy on a test set using the *optimum weights* (w^*), i.e., the optimal weights found for each of the algorithms that we will test.

2.3 Accuracy

In order to evaluate the precision of the algorithms in both the training and test data, we will utilize the following:

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

where

$$\delta_{x,y} = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$$

Project in Mathematical Optimization, UPC,
2023.

3 STUDY OF THE CONVERGENCE

In this section we are going to study:

(1) How the regularization parameter λ affects the results.
(2) The relative performance of the different algorithms *(GM, QNM, SGM)*

In order to make good assessments of the global and local convergence of the algorithms with different parameters we will be executing several realitzacions with some modifications. We will be executing a total of 90 realizations:

- For every digit from 0 to 9.
- For every value of the regularization parameter $\lambda \in 0.0, 0.01, 0.1$
- For every optimization algorithm: *GM, QNM, SGM*

3.1 Global convergence

Firstly, we will study the global convergence of each execution checking whether the found w^* is in a stationary point or not. To do so, we will study the value of $\nabla\tilde{L}(w^*)$, since if we have that w^* is a stationary point, $\nabla\tilde{L}(w^*) = 0$. In our executions, we consider a value of the gradient equal to or less than $\epsilon_G = 10^{-6}$ as zero, and hence that the algorithm has converged.

To check whether a algorithm-λ combination converges for all cases, we will look at the worst-case value of the gradient.

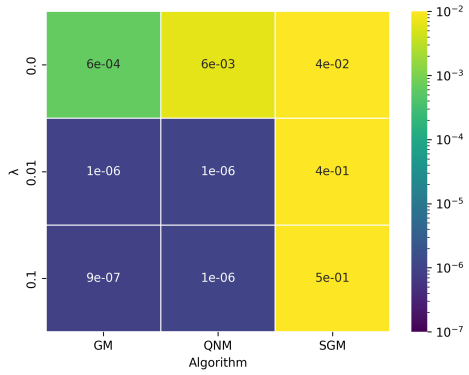

Figure 1: Largest $\|\nabla\tilde{L}(w^*)\|$ found for each algorithm-λ

In Figure 1, we can see that both GM and QNM algorithms converge when λ is set to 0.1 or 0.01, since the norm of the gradient at w^* is smaller or equal than ϵ_G for all executions. This is because, despite the fact that the two algorithms have a theoretical assured global convergence, some cases would have needed more than 1000 iterations to converge to a stationary point (the maximum number of iterations that the two algorithms have been able to perform). The fact that GM and QNM have assured global convergence means that, even in cases that do not converge by the number of iterations, the gradient norm is still "relatively" small (as can be seen in Figure 2 and Figure 3).

On the other hand, SGM does not have an assured theoretical global convergence, since we use an approximation of the gradient and not the actual gradient. Therefore, the global convergence results are much worse than for the other two algorithms. In fact, as we can see in Figure 4, this method does not converge in any of the cases, since the norm of the gradient at w^* is much larger than ϵ_G.

Next, we will examine each algorithm in detail and investigate the effect of each value of λ on the optimization process. Additionally, we will identify patterns that exhibit a higher gradient value at the optimal point.

- ***Gradient Method***

  As evidenced by Figure 2, the magnitudes of $\|\nabla\tilde{L}(w^*)\|$ are nearly zero for all values of λ, with the exception of 8 and 9 for $\lambda = 0$, which fail to converge to a stationary point.

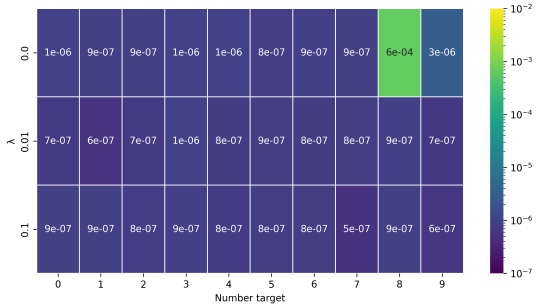

Figure 2: $\|\nabla\tilde{L}(w^*)\|$ for GM-λ

- ***Quasi Newton Method***

  It is evident that the algorithm converges for nearly all combinations, while the gradient with $\lambda = 0$ is considerably smaller in certain cases compared to other λ values. However, it is worth noting that the algorithm fails to converge for numbers 5 and 8 when $\lambda = 0$.

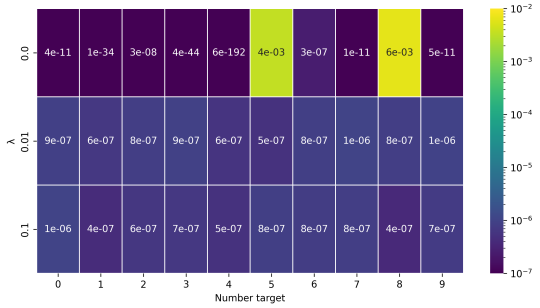

Figure 3: $\|\nabla\tilde{L}(w^*)\|$ for QNM-λ

- ***Stochastic Gradient Method***

  As previously noted, it has been observed that the algorithm fails to converge for any combination of lambdas. Interestingly, it has also been found that the lowest values of $\|\nabla\tilde{L}(w^*)\|$ are attained when λ is set to zero.

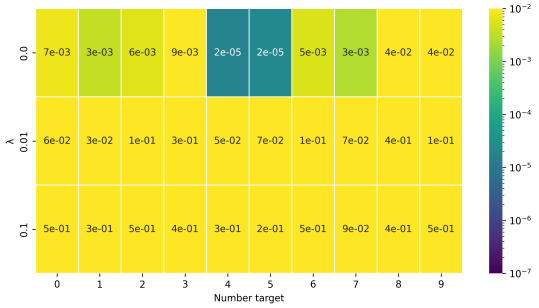

Figure 4: $\|\nabla\tilde{L}(w^*)\|$ for SGM-λ

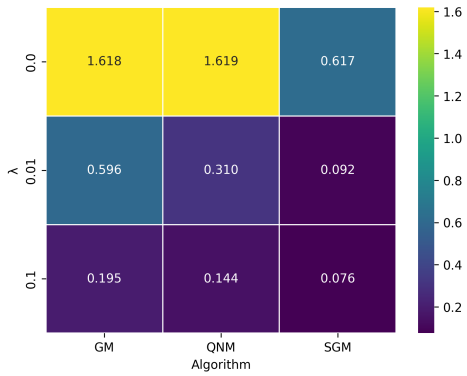

Figure 6: Average execution time (in s) for each algorithm-λ

Best λ-algorithm for Global Convergence

Based on the experimental results, it can be concluded that the GM and QNM algorithms perform the best for values of $\lambda = 0.1, 0.01$. This is because these algorithms successfully converged for all cases tested, indicating their superior performance compared to other algorithms evaluated.

3.2 Local convergence

In this section, we will focus on the study of local convergence, analysing the convergence speed of each algorithm in terms of execution time and number of iterations.

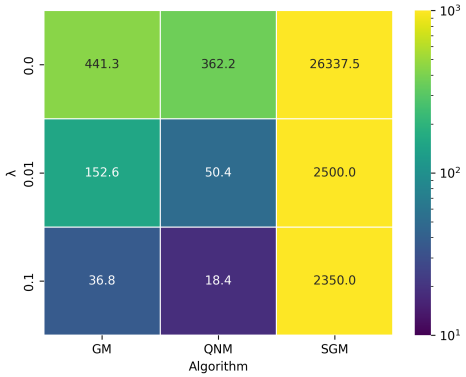

Figure 5: Average number of iterations for each algorithm-λ

It can be observed that the GM and QNM algorithms have a similar magnitude of iterations, with QNM performing better for all values of λ. In contrast, SGM requires significantly more iterations (we are counting the iterations in terms of mini-batches, not epochs). All three algorithms exhibit a clear pattern of decreasing iterations as the value of λ increases. The QNM algorithm shows the most significant decrease in iterations, while the SGM algorithm experiences a considerable decrease when switching from $\lambda = 0$ to $\lambda = 0.01$, with no significant difference when increasing to $\lambda = 0.1$.

There is an obvious relation between the number of iteracions made by the algoithm and the time of execution of it. We expect that for larger number of iterations, larger is the execution time.

Figure 4 demonstrates that, when comparing the running time of each algorithm, higher values of λ result in lower execution times.

In general, a lower number of iterations results in lower execution times when comparing different methods. However, this trend is not always consistent, as exemplified by the fact that GM exhibits a lower execution time than QNM despite having a higher number of iterations for $\lambda = 0$.

SGM, on the other hand, consistently achieved the lowest execution times for all values of λ, despite having the highest number of iterations. This is due to SGM's use of mini-batch updates to its model parameters, which can result in faster convergence compared to updating the entire training data at once. Put differently, SGM updates the model parameters more frequently using smaller amounts of data, which results in faster convergence and ultimately lower execution time.

Influence of λ on the speed of convergence

As previously mentioned, an increase in the values of λ leads to a decrease in the number of iterations and consequently, in the execution time. From the expression of the mean loss function with regularization, we can compute its gradient and hessian.

$$\tilde{L}(w; X^{TR}, y^{TR}, \lambda) = L(X^{TR}, y^{TR}) + \frac{\lambda}{2} \cdot \sum_{i=1}^{n} w_i^2$$

$$\nabla\tilde{L}(w; X^{TR}, y^{TR}, \lambda) = \nabla L(X^{TR}, y^{TR}) + \lambda w$$

$$\nabla^2\tilde{L}(w; X^{TR}, y^{TR}, \lambda) = \nabla^2 L(X^{TR}, y^{TR}) + \lambda I_n$$

We denote $\tilde{\lambda}_i$ and λ_i the eigenvalues sorted in increasing order of $\nabla^2\tilde{L}$ and $\nabla^2 L$ respectively. Since $\nabla^2\tilde{L} = \nabla^2 L + \lambda I_n$, we can deduce that $\tilde{\lambda}_i = \lambda_i + \lambda$ (Proof 5).

Using Theorem 5.1 , we have an upper bound for the reduction of the error of the Gradient Method:

$$\left(\frac{\tilde{\lambda}_n - \tilde{\lambda}_1}{\tilde{\lambda}_n + \tilde{\lambda}_1}\right)^2 = \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1 + 2\lambda}\right)^2$$

We observe, that the bigger the λ, the lower the upper-bound for the reduction of the error of the GM is. This would be true if we were using an exact line search method to find the α^k, but in our case we are using a backtracking line search (BLS) based on interpolations, that satisfies strong Wolfe conditions. However, since the resulting α^k we would get using exact line search belongs to the acceptance region of the strong Wolfe conditions, it is likely that the α^k obtained by the BLS based on interpolations algorithm is "close" to the value of the exact line search. Therefore, having a lower upper-bound for the reduction error using the exact line search means, in most cases, also a lower reduction error when using a modified BLS, leading to a decrease in the number of iterations. In addition, since SGM is like GM but with a gradient approximation, a reduction in the number of GM iterations is also reflected in the SGM iterations.

Furthermore, adding a positive term λ to the diagonal of the Hessian matrix can result in faster local convergence rates for optimization algorithms as it implies that the eigenvalues are more evenly distributed. In other words, the condition number of the Hessian matrix is smaller, indicating that the ratio of the largest eigenvalue to the smallest eigenvalue is closer to one. When the condition number of the Hessian matrix is smaller, optimization algorithms such as quasi-Newton methods converge faster.

Running time per iteration

Now we are going to analyse and compare the running time per iterations of the three algorithms. We do have to keep in mind that iterations in the SGM are computationally cheaper than in the other methods, since in each iteration we compute the gradient using a subset of the data. The cost of one iteration of the gradient method (evaluating $\nabla\tilde{L}$ using all the training data) is approximately the cost of one SGM epoch. In our case, in each epoch we divide the total training data into 100 mini-batches, so we have iteration times approximately 100 times shorter than with GM.

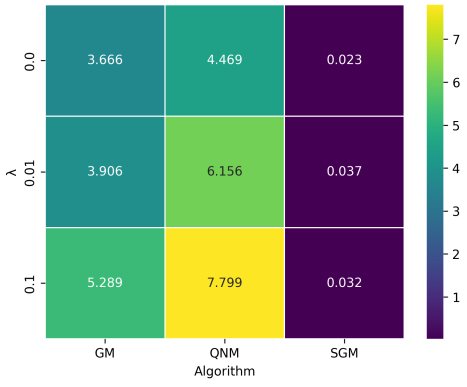

Figure 7: Average execution time per iteration (in ms) for each algorithm-λ

Upon initial inspection, a noticeable disparity exists between the Stochastic Gradient Method (SGM) and the other two methods, with the former demonstrating significantly lower execution time per iteration. This discrepancy arises because, as previously stated, each iteration of the SGM is computationally less expensive since it does not require evaluation of $\nabla\tilde{L}$ over the full training data. Though minor differences exist between the Gradient Method (GM) and Quasi-Newton Method (QNM), it appears that the GM is slightly faster than the QNM for all values of λ. This can be attributed to the fact that the GM only necessitates computation of the descent direction $d = -\nabla\tilde{L}$ at each iteration, whereas the SGM requires computation of both the descent direction $d = -H^k \cdot \nabla\tilde{L}$ and H^{k+1}, thereby rendering its iterations more computationally expensive than the GM.

Moreover, a trend can be observed accordingly the time per iteration increases, particularly in the GM and QNM, for higher values of λ. For $\lambda = 0$, the expression for the gradient simplifies such that $\tilde{L} = L$, making it a bit simpler computationally. However, for $\lambda = 0.01$ and $\lambda = 0.1$, calculating the gradient has an equivalent computational cost. As we see in the Figure 8, the increased time per iteration between $\lambda = 0.01$ and $\lambda = 0.1$ is not due to the computational cost of computing the gradient, but because the cost of computing α^k increases with higher values of λ. This also explains why in Figure 7 we see that all three algorithms are faster with $\lambda = 0$, but increasing λ from 0.01 to 0.1 only affects GM and QNM, while for SGM it has almost no effect.

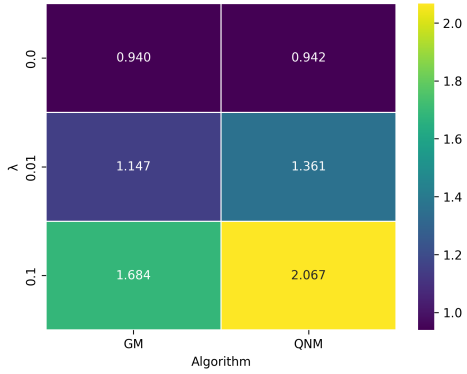

Figure 8: Average time for computing α^k per iteration (in ms) for each algorithm-λ

3.3 Conclusion

After all explained before we can get into the following results:

Table 1: Convergence Summary for GM

Convergence		
λ	Global	Local
0.0	Not Always	441 iterations (1.618s)
0.01	Always	153 iterations (0.596s)
0.1	Always	37 iterations (0.195s)

Table 2: Convergence Summary for QNM

Convergence		
λ	Global	Local
0.0	Not Always	362 iterations (1.619s)
0.01	Always	50 iterations (0.310s)
0.1	Always	19 iterations (0.144s)

Table 3: Convergence Summary for SGM

Convergence		
λ	Global	Local
0.0	Never	26337 iterations (0.617s)
0.01	Never	2500 iterations (0.092s)
0.1	Never	2350 iterations (0.076s)

Our experimental results have shown that the GM and QNM algorithms consistently outperform the SGM in terms of achieving global convergence. On the other hand, while the SGM algorithm requires more iterations to converge, its execution time is significantly lower than that of the other algorithms, enabling it to reach a solution faster in terms of local convergence.

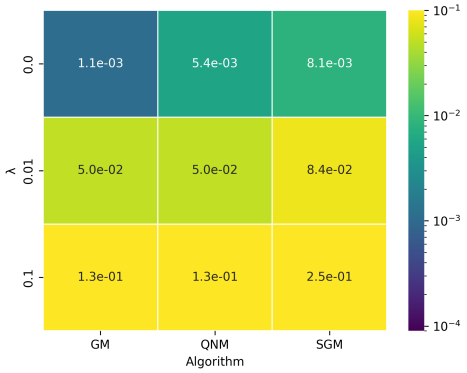

Figure 9: Average $\tilde{L}(w^*)$ for each algorithm-λ

If we seek to minimize the value of $\tilde{L}(w^*)$, we notice that the minimum average value is obtained with the combination 0.0-GM.

However, we have to keep in mind that the larger the λ, the larger the penalty term we will add to the objective function and therefore it will give always larger values for $w \neq 0$. That is why it is also worth mentioning that for $\lambda = 0.01$ and $\lambda = 0.1$, the minimum values of $\tilde{L}$ are found with GM and QNM, which present the same value of the objective function. That is why for these values of λ, the most efficient algorithm is QNM, since it has a faster local convergence than GM.

Best λ-algorithm for the minimization of $\tilde{L}$

As the values of $\tilde{L}(w^*)$ for different values of λ lack comparability, we adopt the criterion of selecting the algorithm that minimizes the value of $\tilde{L}(w^*)$ among those with the same λ, while simultaneously possessing superior global and local convergence properties. In this study, we identify the 0.1-QNM as the optimal algorithm, given that it exhibits the lowest $\tilde{L}(w^*)$ value among the three algorithms with $\lambda = 0.1$, and converges in all cases with the minimal time requirements.

4 STUDY OF THE RECOGNITION ACCURACY

Firstly, we will conduct an analysis to determine the optimal value of λ that yields the highest test accuracy for the different algorithms.

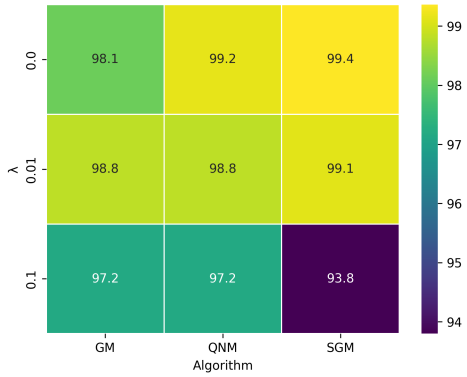

Figure 10: Average training accuracy for each algorithm-λ

We first note that the algorithm with the lowest $Accuracy^{TE}$ is SGM with $\lambda = 0.1$, which achieves a value of 93.8%. However, with the exception of this case, all other algorithms achieve an accuracy above 97%.

In the case of the Gradient Method, the optimal value of λ that maximizes $Accuracy^{TE}$ is $\lambda = 0.01$ with an accuracy of 98.8%. For the QN method, the best value of λ is $\lambda = 0.0$, which results in an accuracy of 99.2%. Finally, the highest test accuracy is obtained with SGM, achieving 99.4% when $\lambda = 0.0$.

We will proceed with conducting the training process using an extended dataset while incorporating the optimal values of the hyperparameter λ for each of the previously mentioned methods. Subsequently, we will proceed with a comparative analysis of the training speed and test accuracy across all three methods.

4.1 Results

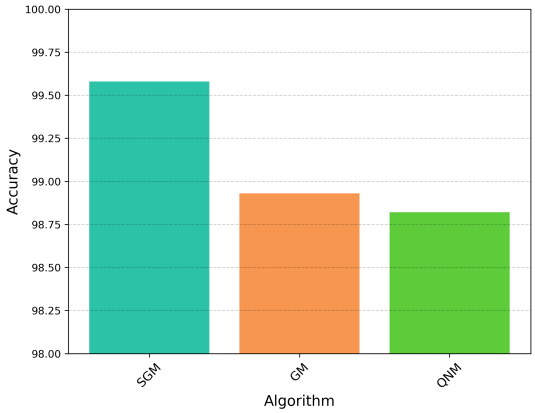

Figure 11: Average test accuracy for each algorithm-λ^{BEST}

Upon observation, it is evident that the Stochastic Gradient Method (SGM) outperforms the remaining two methods in average accuracy. Specifically, with an average accuracy of approximately 99.6%, the SGM outperforms the Generalized Method (GM) and the Quasi-Newton Method (QNM), exhibiting approximate average accuracies of 98.9% and 98.8%, respectively.

Furthermore, analyzing Figure 19, we observe that SGM not only has better average accuracies than GM and QNM, but that the accuracies obtained with SGM are always equal to or better than those obtained with the other two methods.

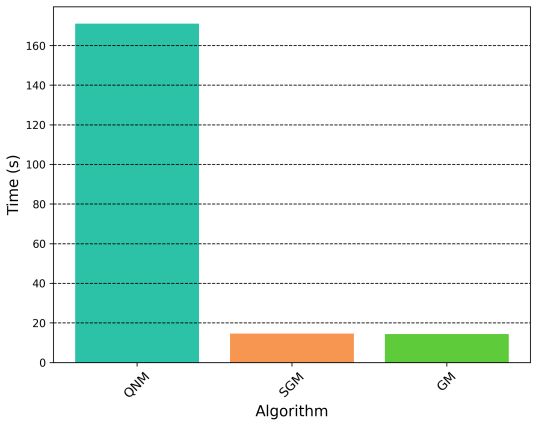

Figure 12: Average execution time for each algorithm-λ^{BEST}

It is noteworthy that the average execution times of SGM and GM are nearly identical. However, upon closer examination of Figure 20, it becomes apparent that the individual execution times for these two methods vary significantly when analyzed on a per-number basis, where SGM presents much higher variance in the elapsed time than GM.

The most noticable time disparity can be found with QNM, being arround ten times slower than the other two methods whereas in section 3.3, the time difference was not so remarkable. This disparity with the other two methods in execution time can be attributed to the fact that, in each iteration of QNM, the approximation of the Hessian of dimensions $n \times n$ is updated, which necessitates updating a total of n^2 values. In contrast, in each epoch of SGM and each iteration of GM, we calculate the computationally equivalent to a gradient vector of size n. This disparity in computational complexity leads to QNM taking a progressively longer time than the other two methods as the dimensions of the training set increase.

4.2 Conclusions

Based on the previously presented results, we can conclude that the best algorithm for maximizing the $Accuracy^{TE}$ metric is 0.0-Stochastic Gradient Method (SGM). This result differs from the previous conclusion we drew for the best λ-algorithm for minimizing $\tilde{L}$, which was the 0.1-Quasi Newton Method (QNM). The reason for this discrepancy is that in the current analysis, we were focused on finding the best test accuracy, and the SGM method achieves higher test accuracy by evaluating the gradient at each epoch in the training data and minimizing the loss function not in the training data but in the test data.

5 APPENDIX

THEOREM 5.1. ***Order of convergence of the GM, general f*** *Suppose that $f \in \zeta^2$ and that $(x^k)_{k=0}^{\infty}$ is a sequence of iterates generated by the gradient method with exact line searches that converges to the stationary point x^* where $\nabla^2 f^*$ is positive definite. Then*

$$f^{k+1} - f^* \leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 (f^k - f^*)$$

Where $0 < \lambda_1 \leq \cdots \leq \lambda_n$ are the eigenvalues of $\nabla^2 f^*$.

PROOF. ***Eigenvalues of*** $\nabla^2 \tilde{L}$

Given

$$\nabla^2 \tilde{L} = \nabla^2 L + \lambda I_n$$

where $\nabla^2 \tilde{L}$ and $\nabla^2 L$ are the Hessian matrices of the functions $\tilde{L}$ and L, respectively, and I_n is the identity matrix of size $n \times n$.

Let λ_i denote the i-th eigenvalue of $\nabla^2 L$, and $\tilde{\lambda}_i$ denote the i-th eigenvalue of $\nabla^2 \tilde{L}$, sorted in increasing order. According to the definition of eigenvalues, we have:

$$\nabla^2 L \cdot v_i = \lambda_i \cdot v_i$$

where v_i is the i-th eigenvector of $\nabla^2 L$.

$$(\nabla^2 \tilde{L} - \lambda I_n) \cdot v_i = \nabla^2 L \cdot v_i = \lambda_i \cdot v_i$$

$$\nabla^2 \tilde{L} \cdot v_i - \lambda \cdot v_i = \lambda_i \cdot v_i$$

$$\nabla^2 \tilde{L} \cdot v_i = (\lambda_i + \lambda) \cdot v_i$$

$$\tilde{\lambda} = (\lambda_i + \lambda)$$

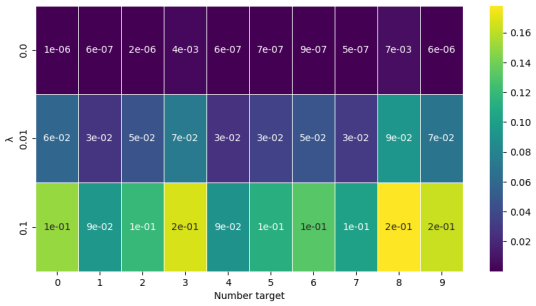

Figure 13: $\tilde{L}^{(\lambda)}$- GM for each number

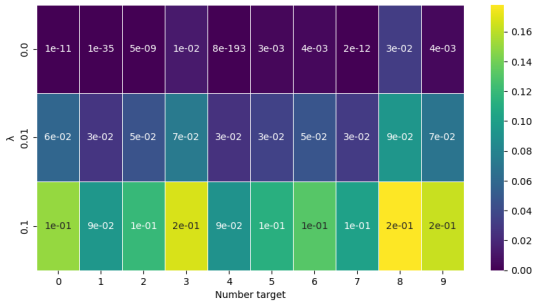

Figure 14: $\tilde{L}^{(\lambda)}$- QNM for each number

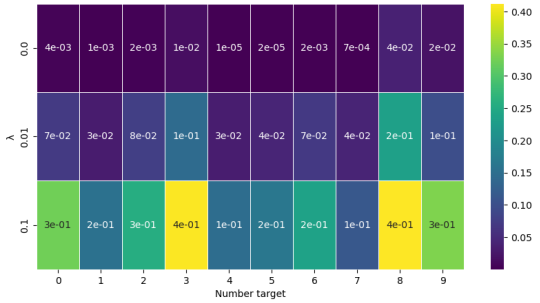

Figure 15: $\tilde{L}^{(\lambda)}$- SGM for each number

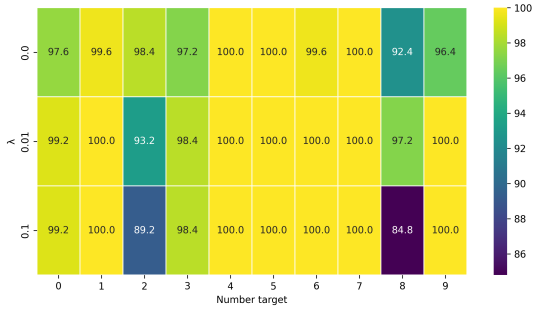

Figure 16: $Accuracy^{TE}$-GM for each number

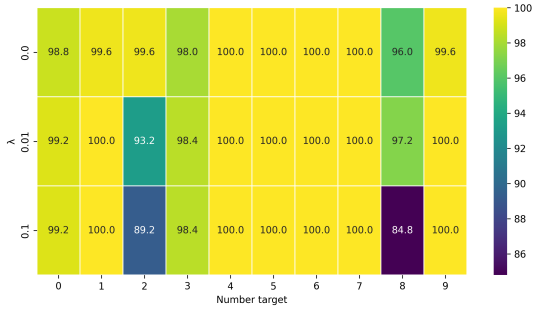

Figure 17: $Accuracy^{TE}$-QNM for each number

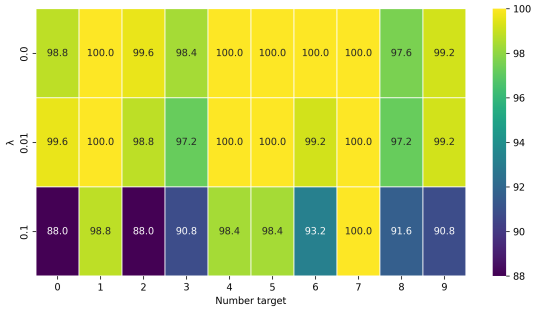

Figure 18: $Accuracy^{TE}$**-SGM for each number**

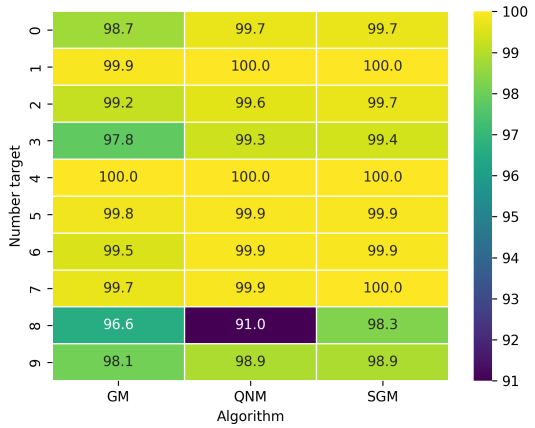

Figure 19: $Accuracy^{TE}$ **for each** λ^{BEST}**-Algorithm combination**

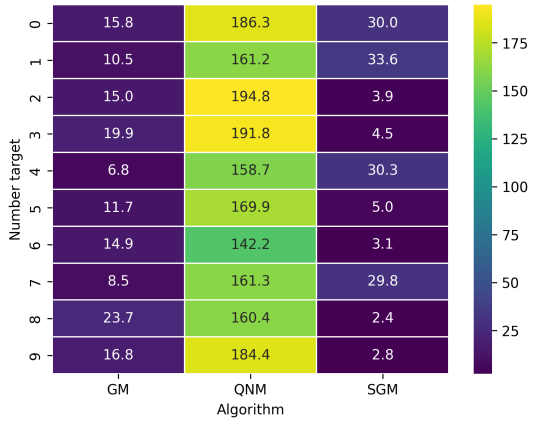

Figure 20: Average execution time (s) for each λ^{BEST}**-Algorithm combination**

Table 4: Seeds used for executing the algorithms

tr_seed	te_seed	sg_seed
590152	171251704	565544