
Report on Pattern Recognition with Single Layer Neural Network (SLNN)

Noa Mediavilla Southwood

Rebeca Torrecilla Domínguez

30 abril 2025

tr_seed = 469986

te_seed = 461895

sg_seed = 565544

1 Introduction

A neural network is a type of machine learning model that makes decisions in a manner reminiscent of the human brain: it employs processes that mimic the way biological neurons learn. Neural networks excel at pattern recognition and are widely used to solve a variety of problems in artificial intelligence and machine learning.

A neural network is typically described in terms of its layers. The first layer, the input layer, receives the raw input signals and forwards them to the next stage. The subsequent layers, known as hidden layers, act as a "black box" where feature extraction and intermediate computations occur. Often, there are multiple hidden layers, each performing successive transformations on the input data. Finally, the output layer produces the network's final result.

Connections between layers are established by assigning random weights. Each input signal is multiplied by its corresponding weight, and a bias term is added. The weighted sum of these inputs is then passed through an activation function, which determines which nodes (neurons) fire and contribute to feature extraction. As the signal propagates through the hidden layers, each neuron computes its own weighted sum and activation independently.

In the problem we address, we focus on a neural network with a single layer. This configuration provides only one stage of decision-making when recognizing specific patterns in the input signal to categorize the output.

Using vector notation, our single-layer network can be defined as follows, where I_i represents the input to neuron i , O_i its activated output, and $y(x, w)$ the network's final output signal (equal to 1 if the input matches the target class, and 0 otherwise):

$$I_i = x_i, i = 1, 2, \dots, n; I_{n+1} = \sum_{i=1}^n w_i O_i, O_i = \sigma(I_i), \sigma(x) = \frac{1}{(1 + e^{-x})}$$
$$y(x, w) = \sigma(I_{n+1}) = \sigma\left(\sum_{i=1}^n w_i O_i\right) = (1 + e^{-(\sum_{i=1}^n w_i (1 + e^{-x_i})^{-1})})^{-1}$$

To prevent overfitting and improve generalization, we introduce regularization, which adds prior information in the form of constraints during loss minimization. Specifically, we employ L2 regularization (also known as Ridge regression). The regularized objective function is given by:

$$\tilde{L}(X^{\text{TR}}, y^{\text{TR}}, \lambda) = L(w; X^{\text{TR}}, y^{\text{TR}}) + \lambda \|w\|_2^2,$$

where X^{TR} and y^{TR} denote the training data and labels, w is the vector of weights, and λ (the regularization coefficient) controls the strength of the penalty applied to large weight values.

Through fine-tuning of hyperparameters, particularly the regularization coefficient λ , we seek to optimize the model's performance, as measured by metrics such as accuracy. By systematically varying λ , we balance the trade-off between fitting the training data and maintaining model simplicity to avoid overfitting.

2 Study of Convergence: Global and Local Convergence

2.1 Global Convergence

In order to analyze both global convergence (the attainment of the optimal solution) and local convergence (the speed of convergence towards the optimal solution), we examine the differences in the number of iterations across various optimization methods. Specifically, the algorithms have been executed for three different values of λ (0.0, 0.05, and 0.1):

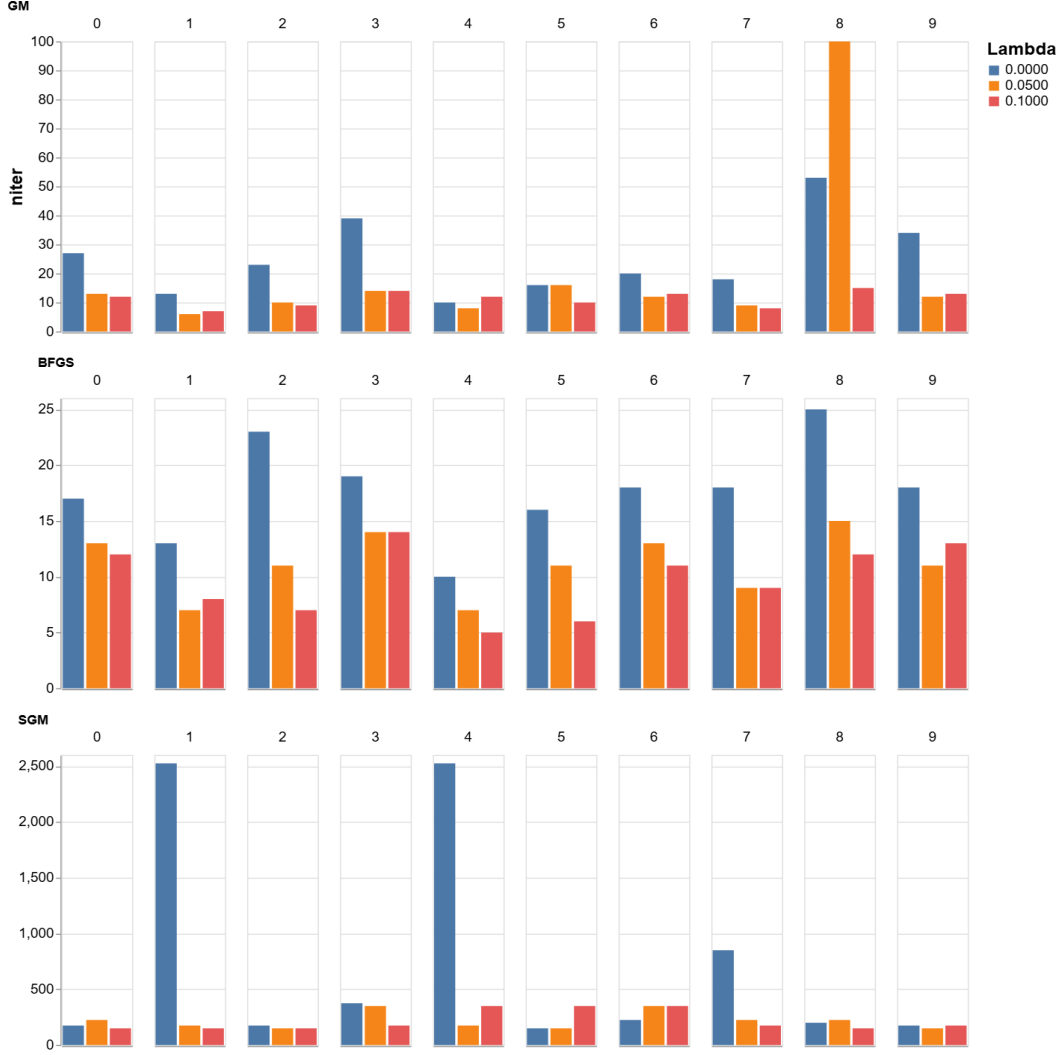


Figure 1: Number of iterations as a function of the method (GM, BFGS, SGM) and lambda

The neural network under consideration is not excessively complex, as it consists of only a single hidden layer. Each layer can be conceptualized as a progressive extraction of features that contribute to the representation of the input data. The goal of this model is to recognize patterns in images in order to correctly predict their class. The hidden layer is responsible for determining which combinations of features are relevant for each category by adjusting the weights to maximize prediction accuracy. It is important to recall that the optimization objective is to minimize the regularized loss function, which reduces classification error while simultaneously penalizing overfitting. In the following section, we analyze the results obtained with the different methods.

GM

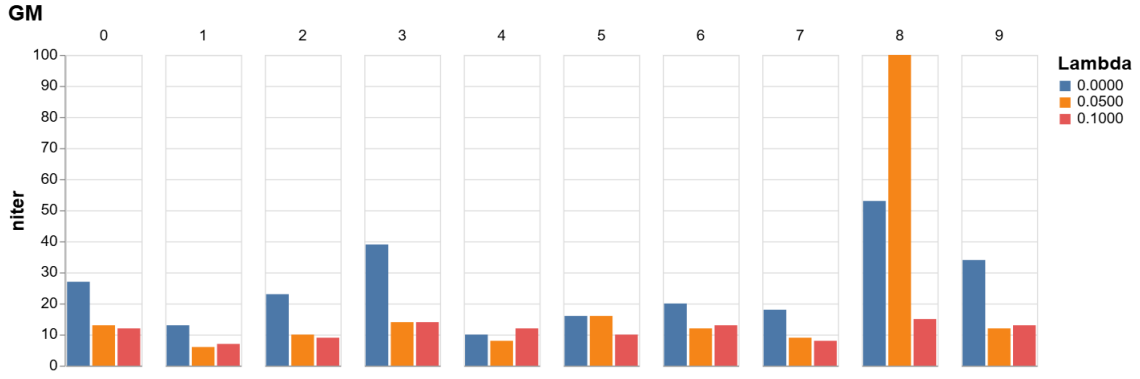


Figure 2: Number of iterations using the gradient method with respect to the different values of lambda and target number.

When considering global convergence using the Gradient Method (GM), we observe a behavior that is highly sensitive to the regularization term λ .

Firstly, in the figure above, we can see that the global convergence has been fulfilled in the most part of the cases for two main reasons: the algorithm has stopped before reaching the maximum number of iterations and the value of the objective function presents a sufficiently low, stable, and consistent value in different initializations. The only case where the first condition it's not met lies at the moderate regularization case when searching for the number 8. As it arrives the 100 iterations, we can deduce that the algorithm must have stopped at this point, concluding without arriving to a stationary point.

When analyzing the number of iterations in general, we observe a highly sensitive behavior with respect to the regularization term λ . Without regularization ($\lambda = 0.0$), the model takes an average of 25.3 iterations to reach the convergence threshold, with extreme cases such as digit 8 requiring up to 53 iterations. The most notable difficulties arise in digits with similar pixel patterns or complex shapes (for instance, digits 3 and 9 also exceed thirty iterations). When moderate regularization is introduced ($\lambda = 0.05$), the average drops to 20.0 iterations; however, one particular case (again digit 8) escalates to 100 iterations, indicating that, as previously mentioned, for this class the algorithm fails to converge before reaching the maximum number of iterations. Finally, with $\lambda = 0.1$, the penalization of large weights significantly accelerates convergence, reducing the average to 11.3 iterations, and the worst case (still digit 8) only reaches 15 iterations. These results confirm that regularization, in addition to aiding generalization, can also smooth the loss landscape, reducing both volatility and the total number of steps required to reach a satisfactory minimum.

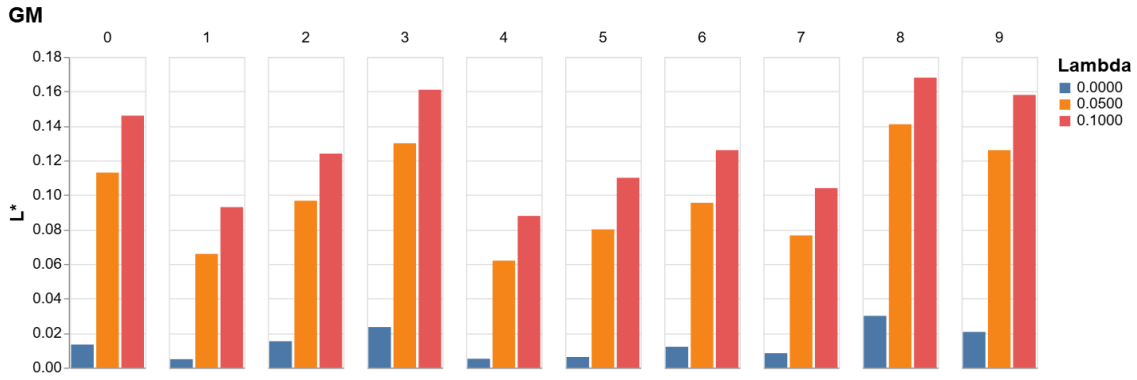


Figure 3: Value of the objective function with the gradient method with respect to the different values of lambda and target number.

If we focus now on the value of the objective function, this second figure illustrates that increasing the regularization parameter λ systematically increases the final objective function value across all classes (digits 0-9). Without regularization ($\lambda = 0$), the values of L^* are consistently low, ranging approximately between 0.01 and 0.03, reflecting that the model fits the training data with minimal error and little penalty on the weights.

When λ is increased to 0.05, L^* rises notably, generally falling between 0.06 and 0.12, depending on the digit. The increase stems from the added regularization term, which penalizes large weights, effectively balancing model fit and complexity. We can see also, that the regularization term doesn't affect the same way to all numbers: at the same time that we obtain high values for digits like 8, 9 or 3 (with a value of the objective function falling between 0.13 and 0.14) we also get lower ones at the time for searching for numbers like 1 or 4 (with resulting objective functions of, approximately, 0.06). That led us to think again that some digit shapes may be relatively more difficult to distinguish or detect.

With strong regularization ($\lambda = 0.1$), the final values of L^* climb even further, frequently exceeding 0.1 and peaking around 0.16-0.17 for the most complex digits (notably digit 8 and digit 3, which share a very similar shape).

These results confirm the trade-off induced by regularization: while it helps smooth the loss landscape and accelerates convergence (as seen in the reduced number of iterations), it also leads to slightly higher final loss values due to stronger constraints on the model parameters. However, in all scenarios, the convergence criterion is satisfied, indicating that the algorithm successfully minimized the objective function under the imposed regularization conditions.

BFGS

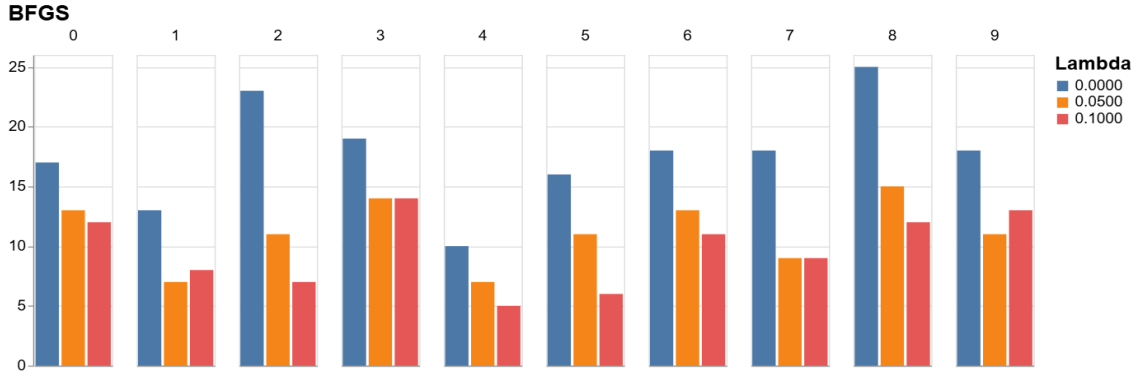


Figure 4: Number of iterations using the BFGS method with respect to the different values of lambda and target number.

The Quasi-Newton approach using BFGS leverages second-order information to significantly accelerate the convergence rate: without regularization, the average number of steps drops to 17.7, with a maximum of 25 iterations for digit 8. With $\lambda = 0.05$, the average is further reduced to 11.1 iterations, and the worst-performing digit (again, 8) requires only 15 iterations. When $\lambda = 0.1$ is used, the average decreases to 9.7 iterations, with a maximum of 14 iterations observed for digit 3. This behavior illustrates how the approximated Hessian matrix guides a more direct descent path toward the minimum, more than compensating for the additional cost of computing the sensitivity matrix. In practice, this implies that BFGS can be the optimal choice when each iteration is computationally expensive and fast convergence in few steps is desired.

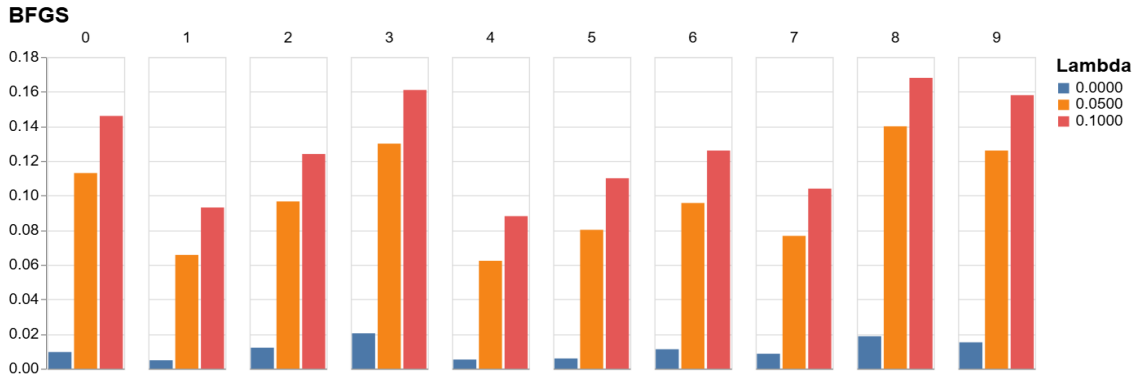


Figure 5: Value of the objective function with the BFGS method with respect to the different values of lambda and target number.

This figure shows the final value of the objective function L^* attained by the BFGS algorithm across all target digits (0-9) and for the three values of the regularization parameter λ (0.0, 0.05, 0.1). Since all optimizations stopped before arriving at the maximum number of iterations and the values achieved seem to be stable and sufficiently low, we can state that global convergence has been reached in all cases. Let's analyze now how the final quality of the solution varies with λ and target classes.

The blue bars represent the scenario with no regularization ($\lambda = 0.0$). For all digits, BFGS achieves very

low values of L^* , typically between 0.005 and 0.02, with minimal variability across digits. This suggests that, in absence of regularization, the algorithm successfully minimizes the empirical risk (training error) and finds a sharp minimizer. Given the low objective values and consistent results across classes, we can consider this an effective practical convergence to a global optimum (or at least a very good local optimum) in the unregularized setting.

The orange bars indicate that introducing a moderate regularization term ($\lambda = 0.05$) leads to systematically higher final values of L^* - now ranging from 0.06 to 0.14, depending on the digit. This increase is expected because the regularization term penalizes large weights, trading off some fitting accuracy for better generalization. There is also more variability across digits:

- Digits like 1, 4, 5, 7 yield lower L^* (around 0.06-0.08)
- Digits like 0, 3, 8, 9 yield higher L^* (around 0.11-0.14)

This suggests that digits with more complex or ambiguous patterns (e.g., 0, 3, 8, 9) are more affected by the regularization constraint. Despite the higher L^* , the convergence is still numerically sound and the solutions are stable across classes.

Lastly, the red bars show the case of strong regularization ($\lambda = 0.1$). As expected, L^* increases further, with values now ranging from approximately 0.09 to 0.17. Again, digits like 3, 8 or 9 experience the highest final objective values, reflecting their higher intrinsic complexity under the regularization constraint. The increased spread and higher values are consistent with a stronger trade-off enforced by $\lambda = 0.1$, leading the optimizer to favor smaller weights at the expense of fitting accuracy. Still, in all cases, the algorithm reaches a stable stationary point.

It's interesting to compare the BFGS results with the GM ones, who seem to share a very similar behavior. In both algorithms, increasing the regularization parameter systematically leads to higher and alike final objective values (L^*) and amplifies variability across digits - with complex digits such as 3, 8 and 9 consistently showing the highest L^* . This parallel pattern suggests that the underlying optimization landscape and regularization effects dominate the convergence behavior, rather than algorithm-specific dynamics. Therefore, both GM and BFGS can be considered robust and consistent optimizers for this problem, converging to qualitatively similar solutions under equivalent regularization regimes.

SGM

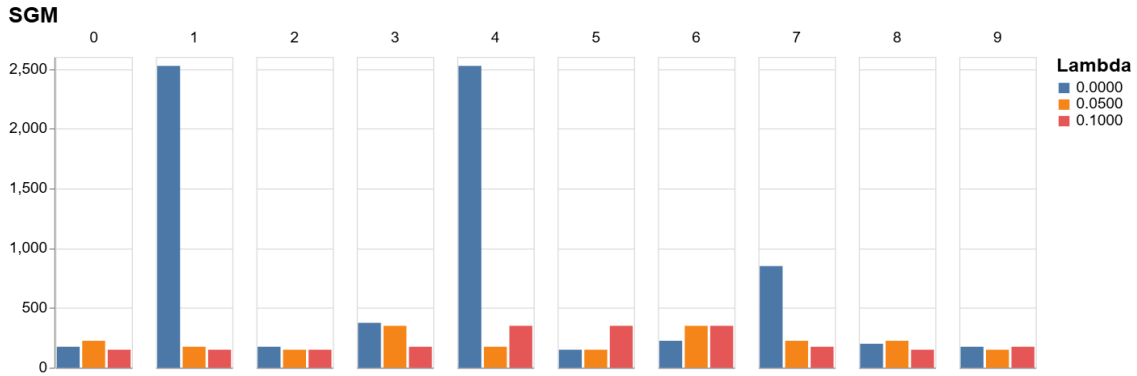


Figure 6: Number of iterations using the SGM method with respect to the different values of lambda and target number.

The Stochastic Gradient Method (SGM) exhibits a markedly different dynamic: the average number of iterations reaches 737.5 for $\lambda = 0$, 217.5 for $\lambda = 0.05$, and 220.0 for $\lambda = 0.1$. This behavior in the not-regularized executions can be partially attributed to three outliers that stand out among all cases. These may correspond to runs where global convergence was not met within a reasonable number of iterations. Although the algorithm lacks an explicit stopping criterion, the significantly elevated values—especially when compared to the surrounding bars, which remain at relatively stable levels—suggest that these executions were unstable and would likely have failed to converge under stricter constraints.

Analyzing the results by regularization value, we observe that in the absence of regularization, the extremes of 2,525 iterations for digits 1 and 4 reflect the inherent variability of stochastic updates. This supports the view that the algorithm performs poorly when no regularization is applied, as the average iteration count increases dramatically. With $\lambda = 0.05$, the slowest cases (digits 3 and 6) drop significantly to 350 iterations, and a similar pattern is observed with $\lambda = 0.1$, where several digits also reach a maximum of 350 iterations. Despite the surprisingly high number of steps, the computational cost per iteration in SGM is so low that, in terms of real training time, it often proves more efficient than GM or BFGS. Consequently, SGM appears particularly well-suited for large datasets, where the speed of stochastic updates more than compensates for the increased number of iterations required.

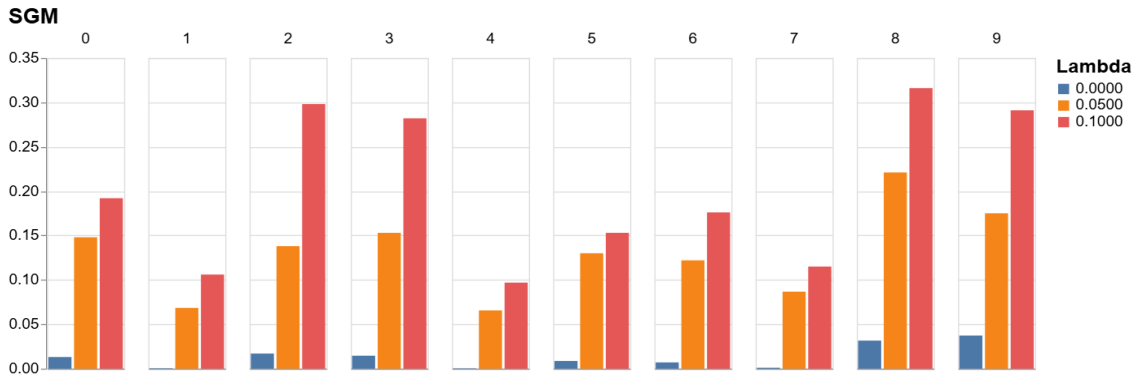


Figure 7: Value of the objective function with the SGM method with respect to the different values of lambda and target number.

The results for the Stochastic Gradient Method (SGM) show a clear dependence on the regularization parameter λ , similar to the previous algorithms.

For the case with no regularization ($\lambda = 0.0$), the final objective values (L^*) remain consistently low across all target numbers (digits), generally under 0.03, with only minor variation. This indicates stable convergence to satisfactory minima in absence of regularization.

When introducing a moderate regularization of $\lambda = 0.05$, L^* increases noticeably across all digits - reaching values roughly between 0.06 and 0.22, depending on the class. Digits like 2, 3, 8 and 9 again exhibit higher L^* values, confirming their relative complexity or separability challenges.

Under strong regularization ($\lambda = 0.1$), SGM shows the most pronounced increase in L^* among the three algorithms analyzed so far. Final objective values now span from 0.10 to over 0.32, with particularly high peaks in digits 2, 3, 8 and 9, exceeding 0.27 in several cases. This sharp escalation suggests that SGM, as a stochastic method, is more sensitive to regularization pressure, which may hinder its ability to finely tune weights - especially under high λ values. Overall, while global convergence is still achieved in all cases (since the final values seem constant and relatively low), the quality of the solution (in terms of minimal loss) deteriorates more aggressively with higher regularization parameters compared to deterministic methods like GM or BFGS.

In particular, SGM exhibits the same qualitative trends as the two previous algorithms: increasing λ leads to higher objective function values and greater dispersion across digits. However, the magnitudes of L^* reached with SGM are systematically higher, especially under strong regularization (in contrast to GM and BFGS, which showed a rise in the value of the objective function in both low regularization and strong regularization). As said, this indicates that SGM may converge less effectively to sharp minima under high penalization due to its stochastic updates and inherent gradient noise. Nevertheless, the pattern of digits affected (notably digits 2, 3, 8 and 9) and the relative ranking of L^* values across λ settings mirror those observed in GM and BFGS. This consistency reinforces the idea that problem complexity and regularization dominate, while algorithmic differences mainly modulate the sharpness and speed of convergence.



Figure 8: Value of the objective function depending on the method (GM, BFGS, SGM), lambda and target number

In summary, while each algorithm shows distinct convergence patterns, especially in response to regularization, some general insights emerge regarding their strengths, limitations, and failure modes.

Regarding the cases in which the algorithms fail to converge —understood here as not reaching the desired minimum loss within a limited number of iterations—three key factors must be considered. First, the randomness and heterogeneity of the input data play a critical role: since SGM processes only one sample per iteration, it is highly sensitive to gradient variance and may oscillate around the minimum, requiring significantly more iterations to accurately estimate the optimal descent direction. Second, the configuration of the regularization parameter λ directly influences the geometry of the loss surface. Smaller values produce a poorly conditioned landscape, which tends to benefit second-order methods like BFGS. In contrast, higher λ values create flatter and more intricate regions with weaker gradients, which can slow down convergence for both GM and SGM. Finally, the predefined maximum number of iterations may hinder full convergence—even when the algorithm is capable of reaching the minimum—simply because the stopping condition is met before the loss has stabilized.

From a practical standpoint, BFGS proves to be the most suitable option when working with relatively

small datasets and when rapid, robust convergence is prioritized, despite the higher computational cost per iteration. GM can serve as a reasonable compromise for those seeking a balance between simplicity and moderate efficiency, particularly in settings where full gradient computation is feasible. For large-scale problems or online learning contexts, SGM stands out due to its exceptionally low per-iteration cost. However, to achieve reliable convergence, it becomes essential to reduce gradient variance—either through techniques such as mini-batching or adaptive learning rates—and to increase the maximum number of iterations permitted.

Ultimately, the choice of optimization algorithm depends on a careful trade-off between computational resources, data scale, and convergence requirements. While BFGS offers superior convergence speed in terms of iterations, SGM provides greater scalability. GM, meanwhile, occupies a middle ground, offering a simple yet reasonably effective alternative when the problem size and complexity are moderate.

2.2 Local Convergence

To analyze local convergence, we evaluate the convergence speed of each optimization method in terms of execution time and number of iterations, considering different values of the regularization parameter. This analysis follows a similar structure to the previous one, with the key distinction that we exclude from the graphs any cases in which global convergence was not meet, as it would be meaningless to assess local convergence in those scenarios. Specifically, we omit all instances that reached or exceeded 100 iterations for the GM and BFGS methods and 500 iterations for the SGM (as there was no explicit number of maximum iterations in the stochastic gradient case we considered the three outliers seen in the previous study as those who didn't met global convergence), which was the maximum number allowed. The results obtained under these conditions are presented below:

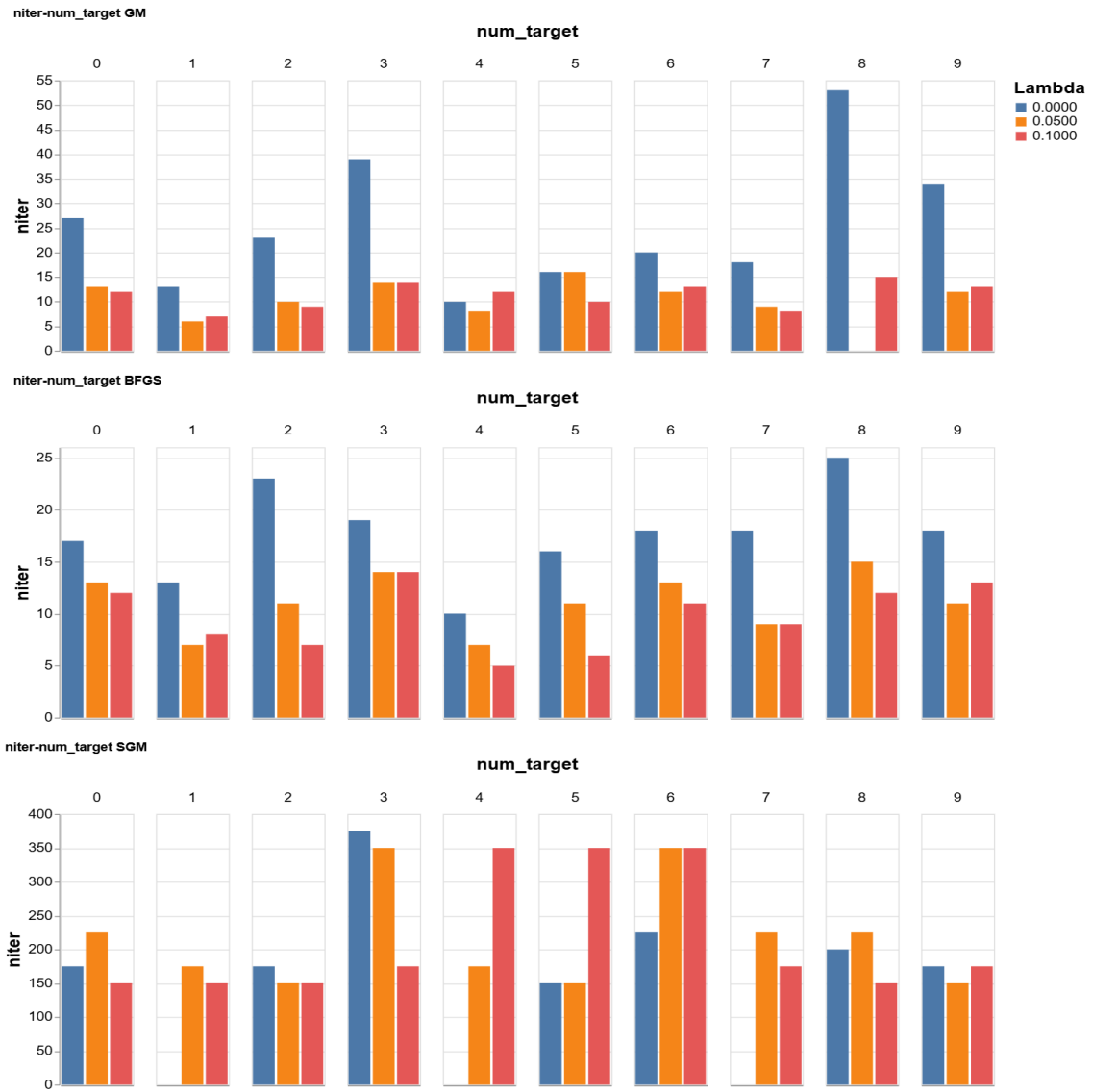


Figure 9: Number of iterations for every used method with respect to the different values of lambda and target number after extracting the cases where the global convergence was not meet.

GM

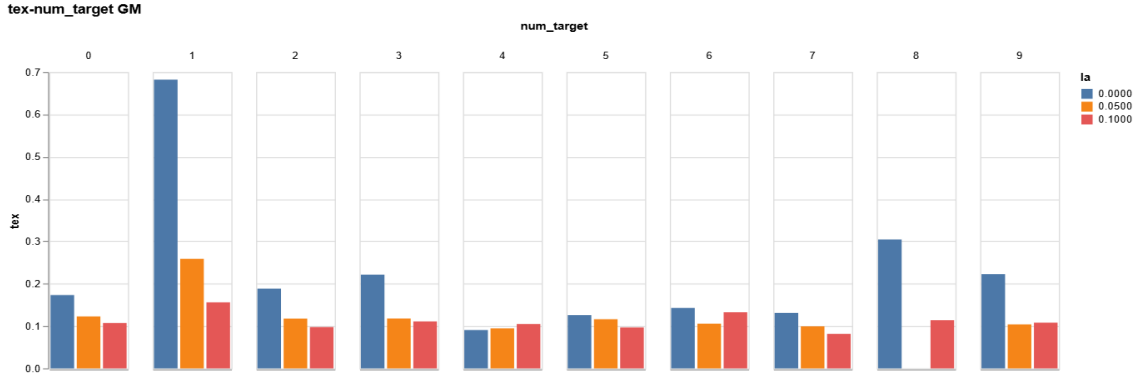


Figure 10: Value of the time of convergence in the gradient method with respect to the different values of lambda and target number.

Firstly, in terms of execution time, we can see the velocity of convergence for every case depending on lambda and target number using the gradient method. The second one provide us a visual representation of the ratio in every case of lambda and target number. Let's analyze them individually.

For the case with no regularization ($\lambda = 0.0$), which corresponds to the blue bars, the convergence time across target classes are generally higher and more variable. In particular, number 1 exhibits the highest convergence time by a large margin, exceeding 0.68, suggesting that in the absence of regularization, the optimization process struggles with this class - likely due to a more complex or less smooth loss surface. Conversely, target numbers like 4 or 5 show the lowest convergence times under this configuration (between 0.1 and 0.2 seconds, approximately), indicating a relatively stable and easily optimizable landscape for these digits without the need for regularization.

Under a moderate level of regularization ($\lambda = 0.05$), it appears to stabilize and reduce convergence times for most target classes. The most significant improvement is again observed for target number 1 (0.25 seconds), whose convergence time drops markedly compared to the unregularized case. Interestingly, however, target number 8 shows the highest convergence time under this setting, reaching approximately 0.44, which deviates from the overall trend and may reflect class-specific sensitivity to this level of regularization. The lowest convergence values in this scenario are found in the digits 4, 6, and 7, where the optimization remains efficient.

Lastly, in the higher case of regularization ($\lambda = 0.1$), the convergence times are relatively uniform and generally lower across all target numbers. The most substantial benefit continues to be for target number 1, whose convergence time is further reduced (now to 0.15 seconds), reflecting the smoothing effect of stronger regularization. Target numbers 2, 3, 5, and 7 exhibit the shortest convergence times under this setting, all remaining close to or slightly above 0.1, which suggests that higher regularization consistently promotes stable and rapid convergence as we don't explore as many solutions as before. Target number 8, while no longer the slowest, still shows a relatively elevated convergence time compared to other in this group.

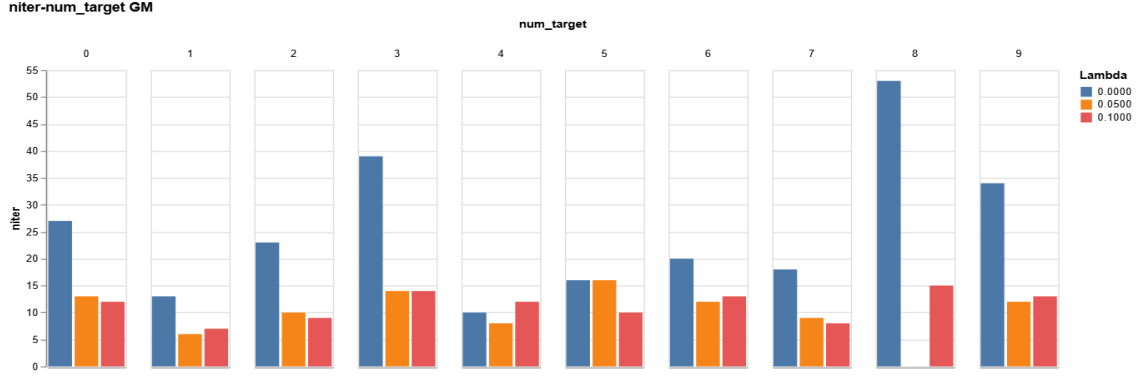


Figure 11: Number of iterations in the gradient method with respect to the different values of lambda and target number.

If we now turn to the speed of convergence in terms of the number of iterations, we observe that the rate at which the model refines its solution near a minimum is highly dependent on the degree of regularization. Without any penalization ($\lambda = 0.0$), the average number of iterations required to stabilize around a local region is relatively high—25.3 steps—and in extreme cases, more complex patterns such as the digit 8 require up to 53 iterations, approaching the limit of 100. Digits like 3 and 9 also exhibit significant resistance, needing 39 and 34 iterations respectively, suggesting that the loss landscape includes flat regions or gentle slopes that hinder rapid descent.

When a moderate regularization is introduced ($\lambda = 0.05$), the average number of iterations drops to 11.11. This overall decrease suggests that penalizing large weights tends to smooth the local landscape, though it may also introduce local complexities that make certain instances harder to unlock.

With a stronger regularization ($\lambda = 0.1$), the average number of iterations drops sharply to 11.3. In this case, even the most difficult digit—again, digit 8—requires only 15 steps, while digits 3 and 6 converge in just 14 and 13 steps, respectively. Altogether, this highlights that although faster local convergence is desirable for speeding up the optimization process, excessive penalization must be carefully managed to avoid confining the model to overly restrictive local minima.

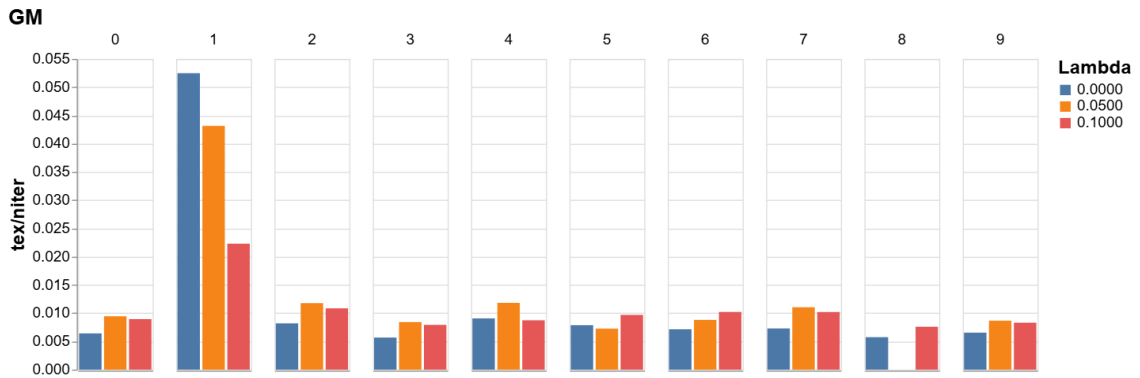


Figure 12: Value of the ratio (time of convergence / number of iterations) in the gradient method with respect to the different values of lambda and target number.

Analyzing the ratio of execution time to number of iterations, we start from the premise that the gradient method requires a very small number of iterations to reach convergence—typically around a dozen in most cases. However, each iteration is relatively expensive due to the computational cost of recomputing the entire prediction vector along with its gradient over the orthogonal projection of the full dataset. This results in typical ratios of about 0.05 seconds per iteration when $\lambda = 0.0$, and even lower values—around 0.01 to 0.02 seconds per iteration—when mild regularization is applied ($\lambda = 0.05$ or 0.1).

It is also important to highlight the enormous value of the ratio achieved by having 1 as the target number. It seems strange until we analyze where the values to calculate the ratio come from. Knowing that this is obtained as the division of the execution time by the number of iterations, the high result makes sense because, when looking for the digit 1, we have a lot of execution time with a low number of iterations.

BFGS

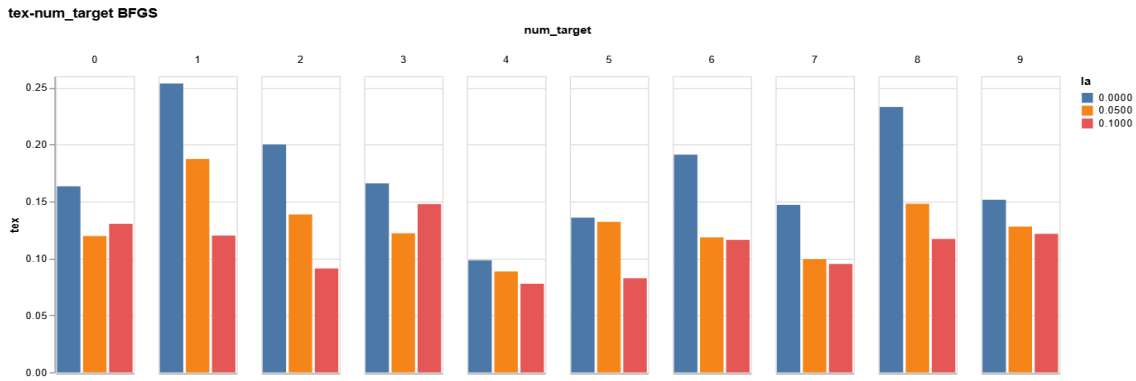


Figure 13: Time of convergence in the BFGS with respect to the different values of lambda and target number.

The BFGS algorithm demonstrates a consistent improvement in convergence time as the regularization parameter λ increases. In the absence of regularization ($\lambda = 0.0$), convergence times are noticeably higher across most target classes. The digit 1 stands out again with the longest convergence time, slightly above 0.25 seconds, followed by targets 8, 2 and 6, all showing relatively slow convergence. This pattern indicates that, without regularization, the BFGS algorithm may struggle with certain class-specific complexities.

In a moderate regularization level ($\lambda = 0.5$) we observe a substantial reduction in convergence time for all classes. The most marked improvements occur in target classes 1, 2 and 8, which had shows significant delays in the unregularized case. At this level, the optimizer becomes noticeably more efficient, especially for digits 4, 5, and 7, where convergence times drop below 0.12 seconds. This behavior suggests that a small regularization term helps stabilize the Hessian approximation, enhancing convergence reliability.

When λ is increased further to 0.1, convergence becomes even more consistent across classes, with generally low and tightly grouped values. Targets 2, 5, and 4 exhibit the fastest convergence under this setting, while no class shows pronounced delays. The optimizer benefits from the stronger regularization by achieving greater numerical stability and a smoother descent trajectory. Interestingly, target 3 shows

a slight increase in convergence time relative to others, but remains within narrow performance range.

Comparing these results with those obtained using the gradient method (GM), several distinctions emerge. First, BFGS shows much smaller variance in convergence times across classes, even in the unregularized case, indicating a more robust behavior. In GM, target number 1 was an extreme outlier under $\lambda = 0.0$, while in BFGS this discrepancy is more contained. Additionally, GM appears more sensitive to changes in λ , particularly with target 8, which showed an atypical behavior at $\lambda = 0.5$. In contrast, BFGS demonstrates more predictable and gradual improvements with increasing regularization, making it a more stable and efficient choice for local convergence across diverse class targets.

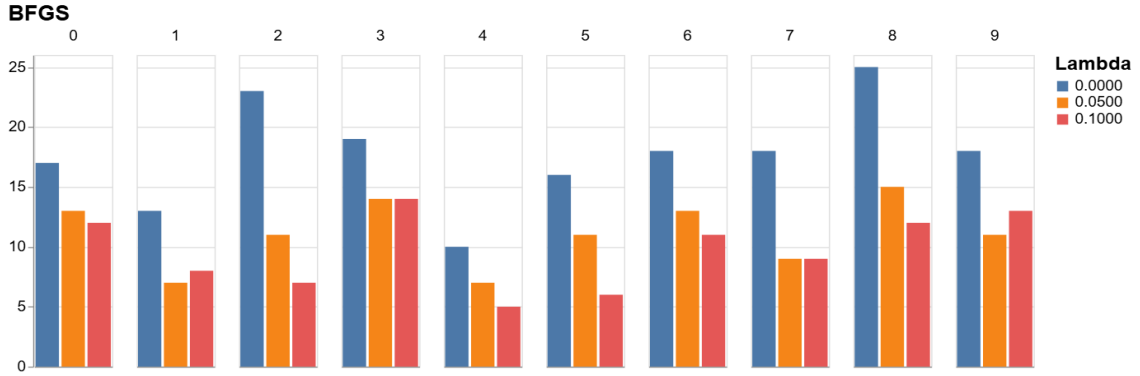


Figure 14: Number of iterations in the BFGS with respect to the different values of lambda and target number.

Let us now analyze the relationship between local convergence and the number of iterations required by the method. Without regularization ($\lambda = 0$), the algorithm explores the curvature of the loss function and adjusts its step sizes accordingly, resulting in an average of 17.7 iterations and eliminating the most stagnant outliers: digit 8, previously the most computationally expensive case, drops to 25 steps, while digits 2 and 3 require 23 and 19 iterations, respectively.

Introducing $\lambda = 0.05$ reduces the average to 11.1 iterations, with the maximum needed to refine the local minimum (again for digit 8) being just 15 iterations, and digits 3 and 6 requiring 14 and 13, respectively. Finally, with $\lambda = 0.1$, local refinement averages 9.7 steps, and the most demanding case—digit 3—is resolved in 14 steps, with digits 9 and 8 requiring 13 and 12.

This improved efficiency stems from the use of the Hessian matrix (or its approximation), which guides the search directly toward the bottom of the local basin, drastically reducing both the number of steps and the likelihood of becoming stuck in flat regions. However, this advantage comes at the cost of additional second-order computations, which increase the per-iteration cost.

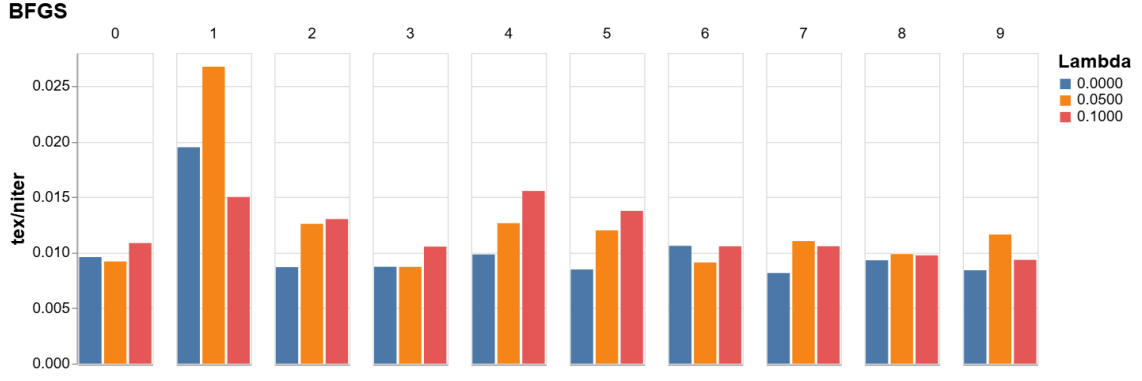


Figure 15: Value of the ratio (time of convergence / number of iterations) in the BFGS with respect to the different values of lambda and target number.

We now turn to a discussion of the ratio values. The BFGS algorithm, like GM, converges in a small number of iterations—on the order of ten—despite each step being more complex: at every iteration, it updates an approximation of the Hessian matrix. Nevertheless, the maximum time-per-iteration ratio drops to half or even less compared to GM, reaching values around 0.026 s/iter for $\lambda = 0.5$, the rest of the cases stabilizing around 0.010 s/iter, a little bit higher than the GM results in average.

SGM

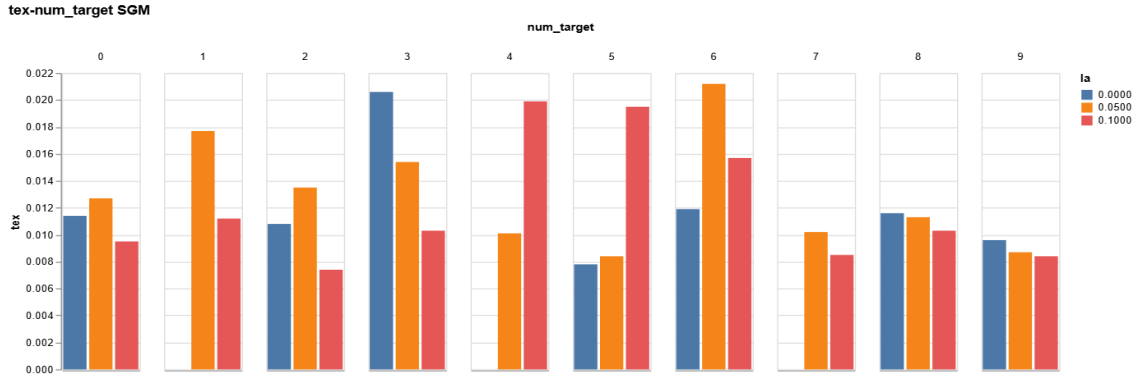


Figure 16: Value of the time of convergence in the stochastic gradient method with respect to the different values of lambda and target number.

The Stochastic Gradient Method (SGM) presents a notably different convergence profile compared to GM and BFGS. In the absence of regularization ($\lambda = 0.0$), the method displays sharp disparities across target classes. In particular, target number 1 experiences a significant delay in convergence (0.20 seconds), while other classes such as 2, 5, and 9 converge almost instantaneously. This suggests that SGM is particularly sensitive to data variability and class imbalance when no regularization is applied. Target 4 and 7 also show moderately elevated times, reinforcing the instability of SGM under low-regularization conditions.

When introducing a small regularization term ($\lambda = 0.05$), convergence becomes significantly more uniform and efficient. All target classes fall within a tight band of convergence times (0.01 to 0.02), indicating that even minimal regularization helps stabilize the stochastic updates. This is particularly evident in

formerly problematic classes like 1 and 4, which benefit from improved convergence. At $\lambda = 0.1$, this trend is maintained or slightly improved, suggesting that SGM quickly reaches a regime of diminishing returns where further regularization does not dramatically change its behavior, unlike more deterministic methods.

In essence, SGM proves to be the fastest and most regularization-sensitive of the three methods. While potentially unstable without regularization, it becomes extremely efficient under moderate λ values, with negligible variance across targets. This aligns with theoretical expectations, as regularization dampens stochastic noise, improving gradient stability and allowing rapid descent.

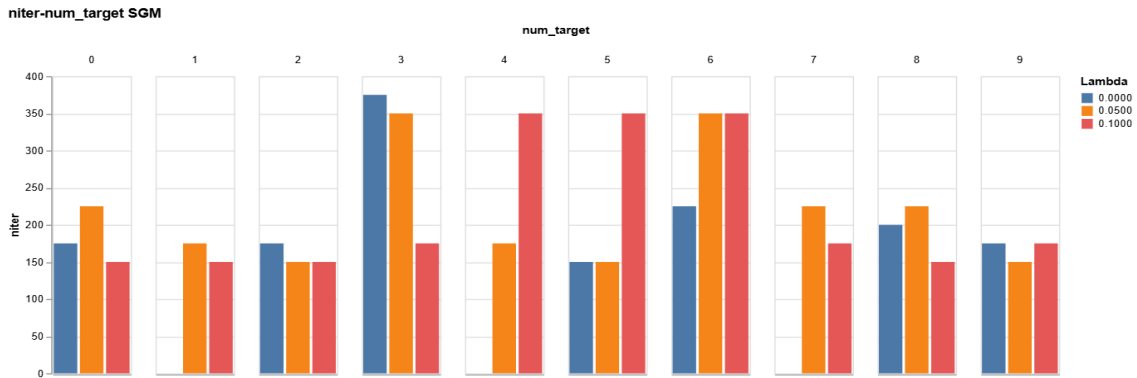


Figure 17: Number of iterations in the stochastic gradient method with respect to the different values of lambda and target number.

We notice that this algorithm exhibits a local convergence behavior that balances a high number of steps with an extremely low per-step cost. Without regularization ($\lambda = 0$), the average number of iterations required to refine the local minimum is 737.5, with extreme cases—such as digits 1 and 4—reaching up to 2,525 iterations. This reflects the high stochastic variability and the small random jumps that prevent stable descent.

When introducing $\lambda = 0.05$, the average number of iterations drops sharply to 217.5, with the most challenging cases (digits 3 and 6) requiring 350 steps, while the rest remain below 225. With $\lambda = 0.1$, the behavior is nearly identical, averaging 217.5 iterations, with similar maxima (350 for digits 4, 5, and 6). This pattern indicates that regularization mitigates part of the stochastic noise, though at the cost of slightly limiting local exploration: the model descends more smoothly toward the minimum, but may miss opportunities to discover deeper valleys. Nevertheless, due to its negligible per-iteration cost, SGM often proves more efficient in real-time when dealing with large and complex datasets, making it an attractive option when every microsecond per step is critical.

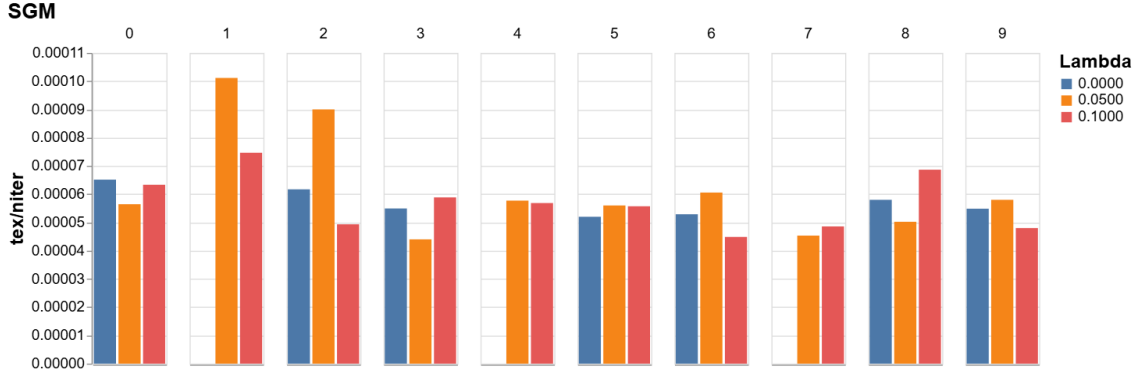


Figure 18: Value of the ratio (time of convergence / number of iterations) in the stochastic gradient method with respect to the different values of lambda and target number.

Focusing finally on the ratio between execution time and number of iterations, the stochastic method SGM requires thousands more iterations to reach a loss level comparable to the GM and BFGS algorithms. However, each iteration is extremely cheap—on the order of a few 10^{-4} seconds in the unregularized case (this can be confirmed manually by dividing the execution time by the number of iterations). This result is primarily due to the fact that, in the same time span during which the previous two methods complete around a dozen iterations, SGM can perform 20 times as many. Consequently, the time-per-iteration ratios are significantly lower in comparison. This highlights its algebraic simplicity—computing only a single partial gradient per image—and its potential for handling large-scale datasets, where reevaluating the entire network at each step would be prohibitively expensive.

Table 1: Average of different metrics based on the algorithm and the regularization term.

	λ	Number of iterations	Execution time	Objective function
GM	0.0	25.30	0.2283	$1.396 \cdot 10^{-2}$
GM	0.05	11.11	0.1263	$9.398 \cdot 10^{-2}$
GM	0.1	11.30	0.1110	$1.278 \cdot 10^{-1}$
BFGS	0.0	17.70	0.1739	$1.117 \cdot 10^{-2}$
BFGS	0.05	11.10	0.1282	$9.862 \cdot 10^{-2}$
BFGS	0.1	9.70	0.1100	$1.278 \cdot 10^{-1}$
SGM	0.0	210.71	0.0119	$1.847 \cdot 10^{-2}$
SGM	0.05	217.50	0.0129	$1.308 \cdot 10^{-1}$
SGM	0.1	217.50	0.0121	$2.026 \cdot 10^{-1}$

After the study of local convergence for the three algorithms we can agree that, in all cases, the execution time was dependent of the value of regularization λ . This relationship seem to be proportionally inverse as the execution time was lower as long as λ increased.

A possible explanation for the fact that a higher regularization term (λ) leads to a faster convergence of the algorithm to a stationary point lies in the effect that such penalization induces on the geometry of the objective function. Increasing λ strengthens the penalty on the network weights, which directly results in a reduction of their magnitude during training. This constraint smooths the loss function landscape, causing the admissible solution space to contract and the descent trajectories to exhibit less oscillation.

From a geometric perspective, a higher λ tends to eliminate narrow valleys and high-curvature regions that may cause optimization methods—particularly those based on gradients—to progress slowly or require multiple direction corrections. Instead, the new regularized landscape is characterized by a smoother topology, with broader regions of constant or gently decreasing slope, which favors a more direct descent toward a local or global minimum.

Moreover, by reducing the capacity of the model (by limiting the complexity it can achieve through large weights), the number of effectively relevant parameters in the optimization is also reduced. This leads to a simplification of the search space, which in practice can manifest as a faster descent toward a satisfactory stationary point, especially in contexts where the goal is not to reach the absolute minimum loss but rather a solution that generalizes well.

An interesting deviation from this general behavior is observed in the Stochastic Gradient Method (SGM), particularly in the cases of digits 4, 5, and 6. Contrary to the trend exhibited by GM and BFGS—where higher regularization consistently reduces the number of iterations—SGM shows an increase in execution time for these specific digits under stronger regularization. This anomaly can be attributed to the stochastic nature of the algorithm and the local characteristics of the loss surface for these classes. In these cases, it is plausible that the regularization term, rather than stabilizing the descent, causes the gradient signal to weaken excessively. As SGM updates parameters based on single samples, it becomes more sensitive to noisy or low-gradient regions. When the signal-to-noise ratio deteriorates—especially in flatter or more ambiguous areas of the loss surface—SGM may require a substantially greater number of updates to ap-

proximate a descent direction that leads toward a stationary point. This effect is exacerbated when the digit class introduces subtle features or decision boundaries that are harder to distinguish under strong penalization, making convergence more erratic and computationally demanding for those particular digits.

In summary, a high value of λ acts as a regularization mechanism not only from a statistical perspective but also from a numerical one: it dampens gradient variability, moderates the curvature of the loss function, and simplifies the optimizer's trajectory—together significantly reducing the number of iterations required to reach convergence.

2.3 Discussion

When analyzing in depth the effect of regularization through the λ coefficient, it becomes clear that increasing λ substantially transforms the loss landscape and facilitates convergence across all three optimization methods. Specifically, for $\lambda = 0.0$, the Gradient Method (GM) requires an average of 25.3 iterations, which decreases to 20.0 for $\lambda = 0.05$ and drops further to 11.3 with $\lambda = 0.1$. This trend is mirrored in BFGS, where the iteration count falls from 17.7 without regularization to 11.1 with $\lambda = 0.05$ and finally to 9.7 with $\lambda = 0.01$, illustrating the smoothing effect of weight penalization on the loss surface. The case of SGM is even more dramatic in terms of relative reduction: the average number of iterations falls from 737.5 at $\lambda = 0.0$ to 217.5 for both $\lambda = 0.05$ and $\lambda = 0.1$, although the absolute iteration count remains substantially high. This collapse in the mean reflects how regularization acts as a damper on small slopes and flat regions, homogenizing the descent speed across digit configurations.

Looking at the upper extremes of each method, GM reaches maxima of 53, 100, and 15 iterations for $\lambda = 0.0, 0.05$ and 0.1 , respectively, with digit 8 consistently being the most challenging case. BFGS, in contrast, shows much more contained maxima: 25 iterations without regularization, 15 with $\lambda = 0.05$, and 14 with $\lambda = 0.1$, demonstrating its robustness under adverse conditions. As for SGM, peak values reach as high as 2,525 iterations for $\lambda = 0.0$, dropping to 350 for both $\lambda = 0.05$ and $\lambda = 0.1$ —highlighting the variability and inherently stochastic nature of its updates. These figures confirm that, while GM may become trapped in low-curvature regions and SGM suffers from dispersion due to its randomness, BFGS leverages second-order information to ensure fast and stable convergence even in the most unfavorable scenarios.

When choosing the most appropriate strategy for our SLNN, it is essential to consider not only the number of iterations, but also the computational cost per step and the final accuracy achieved. BFGS with $\lambda = 0.1$ emerges as the most balanced configuration: with fewer than 10 iterations on average, a maximum of just 14 in the most difficult cases, and accuracy levels between 99% and 100%, it delivers a moderate total runtime and a manageable computational effort per update. Stochastic Gradient, despite requiring hundreds of iterations, is unbeatable when per-step cost is critical—as in large-scale datasets—since each update is extremely lightweight. Finally, although GM is the simplest method to implement, it falls slightly behind BFGS under moderate regularization, yet remains a valid choice when algorithmic simplicity and interpretability are prioritized.

3 Study of Accuracy

The analysis of the test-set accuracy as a function of the combination of optimization algorithms and the regularization parameter λ reveals clear trends in the model's ability to recognize individual digits. This metric tells us the proportion of test images that were classified correctly, and is therefore a key indicator of each configuration's generalization capacity.

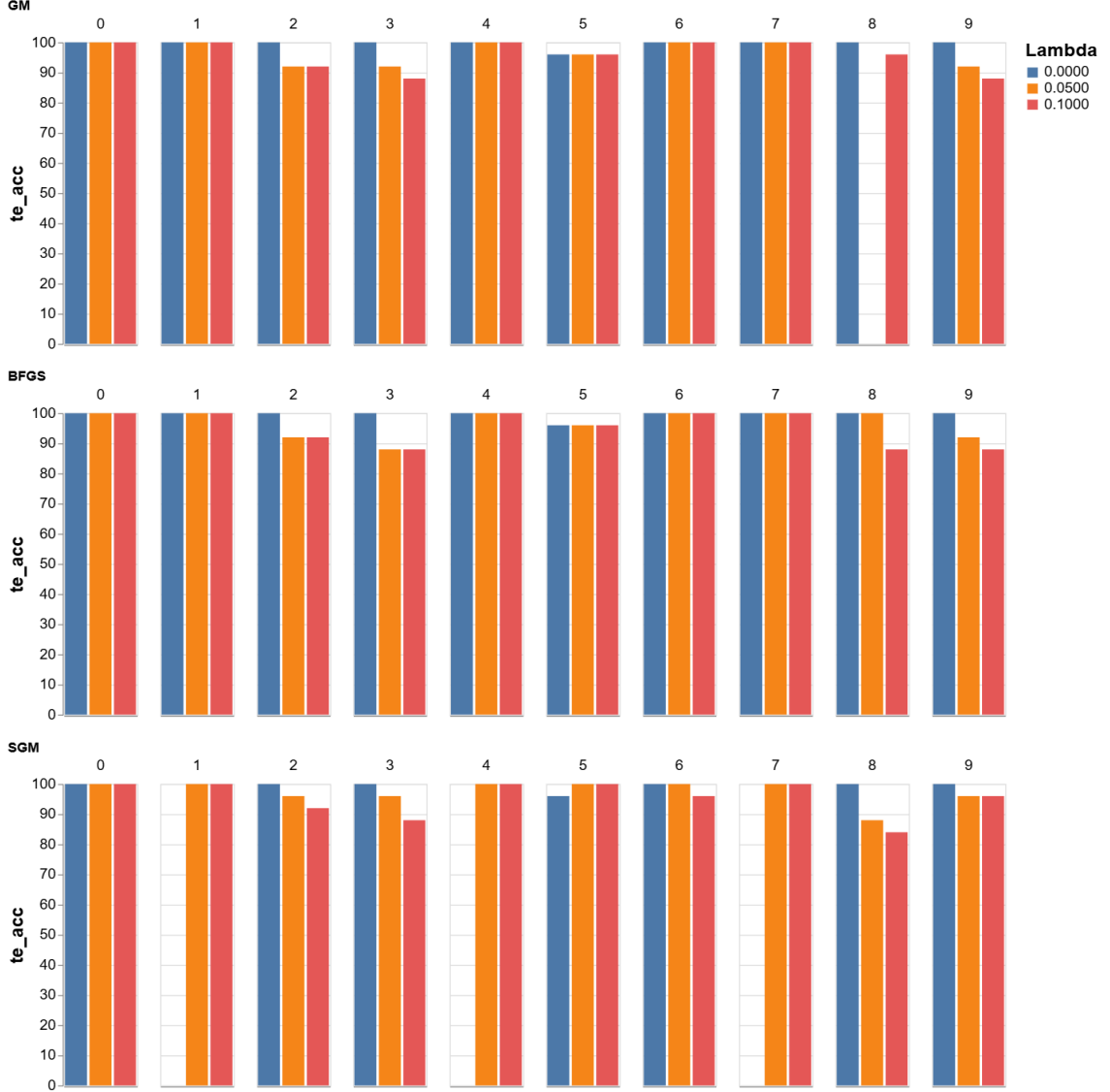


Figure 19: Value of the test accuracy based in the three different algorithms and lambda values.

First, we observe that for $\lambda = 0$ (i.e., no regularization), test accuracy is essentially 100% in almost every case, regardless of the method used or the digit in question. This suggests that, without regularization, the algorithms can fit the training set extremely well. However, there are small exceptions for some digits (2, 3, 8 and 9), where accuracy dips slightly to 98% or even 96%, which may indicate that these digits are harder to distinguish correctly due to visual similarities with other digits.

Another point to consider is that such high accuracy without regularization ($\lambda = 0$) can be a sign of overfitting: the model learns the particularities of the training set very well and makes perfect predictions on

the same type of data, but this extreme specialization can be detrimental when presented with a slightly different dataset (for example, with minor variations in handwriting style or image noise). In these cases, an overfitted model tends to lose precision and robustness, so introducing some regularization often helps improve generalization in real-world scenarios.

Table 2: Average of train and test accuracy based on the algorithm and the regularization term.

	λ	Train accuracy	Test accuracy
GM	0.0	99.28	99.60
GM	0.05	98.71	96.88
GM	0.1	97.56	96.00
BFGS	0.0	98.96	99.60
BFGS	0.05	98.08	96.80
BFGS	0.1	97.52	95.20
SGM	0.0	98.57	99.43
SGM	0.05	92.00	97.60
SGM	0.1	77.72	95.60

When we increase λ to 0.05 and 0.10, differential behavior between the algorithms becomes evident. The Gradient Method and BFGS generally maintain high test accuracy, but with slight reductions, often settling around 92%–96%. This drop isn’t dramatic but is consistent, indicating that even mild regularization begins to penalize the model’s ability to fit digit patterns. By contrast, the Stochastic Gradient Method shows a more polarized behavior: in some cases it still achieves 100% accuracy, while in others it plunges abruptly to around or below 60%, especially at $\lambda = 0.1$. This can be attributed to the method’s stochastic nature, which, when combined with strong regularization, can lead to suboptimal optimization and thus to models that fail to learn properly.

Table 3: Average of train and test accuracy based on the target number.

	Train accuracy	Test accuracy
0	97.07	100.0
1	99.90	100.0
2	91.64	95.11
3	91.33	93.33
4	99.95	100.0
5	98.98	96.89
6	97.51	99.56
7	99.45	100.0
8	85.35	94.50
9	91.51	94.67

A critical aspect emerging from the data is the dependence of accuracy on the specific digit being identified. For digits like 1, 4 or 7, accuracy stays virtually at 100% regardless of λ or algorithm. This is likely because these digits have distinctive, easily separable shapes. In contrast, for digits such as 2, 3, 8 or 9, accuracy shows greater variability—especially at higher λ , reflecting either the model’s intrinsic difficulty in capturing subtle distinguishing features or the visual similarity among those digits (for example, 3 and 8 share similar upper and lower loops).

It is also worth noting that the effect of λ on accuracy is most pronounced with the Stochastic Gradient Method. When λ is small, SGM can deliver excellent results (even outperforming other methods), but when λ is large, its performance tends to deteriorate significantly—likely because strong regularization overly constrains the weights in an already noisy update context. In contrast, the Gradient Method and BFGS exhibit more stable and robust behavior under changes in regularization, albeit with a slight efficiency loss when λ becomes too high.

Based on our analysis of test-set accuracy, taking into account both absolute performance and robustness to small data changes, the combination that offers the best overall balance is the BFGS algorithm with a moderate regularization factor, around $\lambda = 0.05$.

Firstly, BFGS demonstrates very high stability: accuracy remains above 96% for all digits—including the more challenging ones (2, 3, 8 and 9), with much lower variance than other configurations. This consistency is crucial when we want the model not only to hit performance peaks in certain classes but also to generalize well across the board. Unlike SGM, which can drop below 60% on some digits at high λ , BFGS with $\lambda = 0.05$ consistently achieves near 100% accuracy on easy digits (1, 4, 7) and around 96%–98% on more complex digits, ensuring uniform performance.

Secondly, introducing mild regularization ($\lambda = 0.05$) helps mitigate the overfitting risk seen in the no-regularization case ($\lambda = 0.00$), where, despite perfect test accuracy, the model may be capturing too much noise specific to the training examples, without penalizing the model so heavily that its precision suffers significantly. This is reflected in only a slightly higher loss than $\lambda = 0.00$, paired with better resilience to real-world data variations. Thus, BFGS with $\lambda = 0.05$ combines the best of both worlds: high average accuracy, low variability, and strong overfitting resistance, making it the optimal choice for most practical digit-recognition applications.