Lab 1 OTDM Report

Àlex Martorell i Locascio

October 2022

0 Goal of this report

In this report, the objective is to study in depth the convergence of the three algorithms that were used to compute the optimal w^* in the Single Layer Neural Network (SLNN) structure. The w^* are the weights of the NN that produce the final output.

There are two important theoretical aspects that are covered in depth in this report. Firstly, the presence of the λ term in the loss function, a common technique in regularization based learning (i.e. ridge regression) that captures the complexity of a model. Secondly, it is explained how does the usage of the Stochastic Gradient Method affect the general concept of Gradient Descent.

Covering this allows to perform an in-depth analysis of the results in different variables when running the program: Execution time, Number of iterations, values of L, values for λ. The goal is to extract general conclusions that can allow the user to choose the adequate algorithm for any case scenario in this problem.

Finally, the seeds to run the code are: `tr_seed = 23867484; te_seed = 123456; sg_seed = 565544;`

1 Study of the convergence

The penalization term λ.
The loss function for this problem is:

$$L(X^{TR}, y^{TR}, \omega) = \frac{1}{p}\sum_{j=1}^{p}\left(y(x_j^{TR}, \omega) - y_j^{TR}\right)^2 + \lambda\frac{\|\omega\|^2}{2} \tag{1}$$

Note that a penalization term is introduced. This is to prevent overfitting by controlling the complexity of the model. Observe that in this case, the technical term for this is L_2-regularization, because the penalty term is $\lambda\|\omega\|^2$, with the euclidean norm. This λ clearly penalizes high weights, and makes the weights go to zero. This is why this procedure is referred sometimes as *weight decay*.

In the gradient method, for instance: $d_k = -\nabla \tilde{L}^k$. In particular, the penalization term in the gradient of the loss function becomes $\lambda\omega_i \quad \forall i$. [2]

$$\omega^{k+1} = \omega^k + \alpha^k d^k$$

$$\omega_i^{k+1} \propto -\frac{\partial \tilde{L}}{\partial \omega_i} - \lambda\omega_i \quad i = 1 \div n$$

Notice how the weights shrink faster. Analogously, for methods of the Newton family (recall that QNM is actually a First Derivative Method), we have $d^k = -\nabla^2 \tilde{L^k}^{-1} \nabla \tilde{L^k}$. However, since the restriction of this project is to "First Derivative" methods, the effect of λ in the Hessian are not a concern.

1.1 Global convergence

Global convergence refers to the possibility of an algorithm finding a solution. This can be proven by verifying the Zoutendijk Theorem and has been done in theory. In this report the study is restricted just to the $\tilde{L}$ function and a numerical analysis more than a mathematical proof is performed of the convergence for each algorithm.

Recall the loss function for this problem seen in (1), which presents two distinct terms. The first one is the difference in the prediction for the training set and the actual values. The second one, is the regularization term, whose objective is to penalize complexity in a model. Obviously, the lower the value of $\tilde{L}$, which represents the loss, the better. We check the value of $\tilde{L}$ based on the regularization parameter λ. Figure 1 shows a comparison of the three different algorithms (GM, SGM, QNM) for different $\lambda = 0, 0.01, 0.1$.

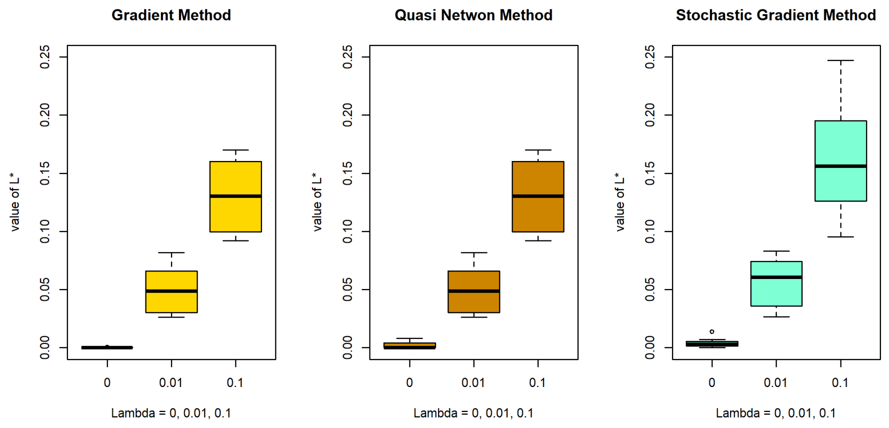

Figure 1: Global convergence comparison by method

The first term in the loss function is the difference in prediction in the training set. If the accuracy is 100, this first term has to be equal to 0. Observe that most training set accuracies are 100, which translates to a L^* value of 0 for $\lambda = 0$ in the three methods. The divergence from 0 comes from the second term in the loss function, which cannot be zero when $\lambda \neq 0$. However, since weight decay forces that the components ω_i are close to 0, the deviation from absolute zero is low. (Observe that the maximum is 0.25 for the Stochastic Gradient Method, which is due to a low accuracy in prediction for a specific number).

The Stochastic Gradient Method, which will be covered more in depth in the next section, presents global convergence. The Zoutendijk condition is verified as long because d^k is an approximation of $\nabla \tilde{L}$. There also exists a sufficient condition for convergence which will be discussed later.

In summary, it is believed that the three algorithms find the optimal solution w^* for the problem. As the graph shows, if $\lambda = 0$, the Loss function is almost exactly zero. The same can be said for $\lambda = 0.01, 0.1$ However, we have to keep in mind that the data set is small and the number of parameters to estimate (the 35 parameters corresponding to the pixels of the number because it is just a Single Layer NN). Also, it should be said that with a larger data set, the values for L^* are found to be extremely low for (in a case with high accuracy). This is because a misclassification has less impact on L^*.

1.2 Local convergence

The effect of the Stochastic Gradient.
The gradient of the loss function is of the additive type:

$$\frac{\partial \tilde{L}(\omega; X^{TR}, y^{TR}, \lambda)}{\partial \omega_i} = \frac{1}{p} \sum_{j=1}^{p} 2\big(y(x_j^{TR}, \omega) - y_j^{TR}\big) \frac{\partial y(x_j^{TR}, \omega)}{\partial \omega_i} + \lambda_i$$

meaning that the computational cost is $O(p \cdot n)$, where p is the size of the training data set. n is fixed for this problem, but when p grows the cost becomes significant. The idea behind Stochastic Gradient is that the gradient is estimated with a **minibatch** $X' = (x^1, \ldots, x^m)$ of samples. (m ¡¡ p). Therefore, the estimate of the gradient now is:

$$\nabla \tilde{L} \approx \nabla \tilde{L}(\omega; X', y', \lambda)$$

There are also other differences, such as the step length, which is computed in a different way.

1.2.1 Speed of convergence in terms of execution time and the number of iterations

In this first section, the real effects of λ on the speed of convergence are not verified. Observe that theoretical results are verified: QNM has quadratic convergence, and in SGM and GM it is only linear. However, as explained above, SGM was introduced to combat the computational costs of the GM. The goal is to see if this can be observed in the execution time for a data set of just size 250.

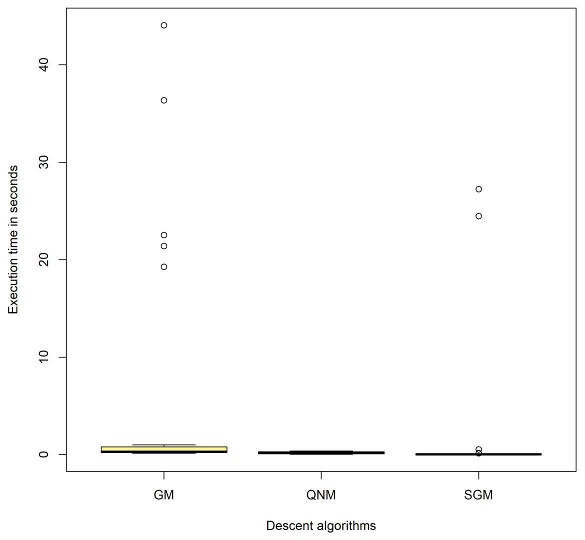

Figure 2: Average time of execution by algorithm

Figure 2 shows a Box plot of the average execution time for each algorithm. Since this plot may not bring much clarity to the issue, table 1 below shows the different position measures for the execution time variable per method.

Method	Min	Q1	Median	Q2	Max	Mean
GM	0.138	0.207	0.285	0.792	44.060	5.099
QNM	0.022	0.074	0.156	0.285	0.383	0.175
SGM	0.0180	0.0270	0.0335	0.0660	27.2200	1.782

Table 1: Execution time metrics for the different algorithms

Observe that although in some cases, GM converges very quickly, it is already slower than the SGM. However, the QNM outpeforms the other two because of its quadratic convergence, with a maximum of 0.383 seconds for a full run.

The number of iterations of SGM is always significantly higher than GM and QNM, as it can be seen in table 2. However, note that the number of iterations

Method	GM	QNM	SGM
Median	52	18	1500
Mean	165.03	22.9687	11358.33

Table 2: Number of iterations for each algorithm

in the SGM is not really comparable to other methods, in the sense that an epoch (e) has many sub-iterations k consisting of the different batches of the training data set. In other words, although it is necessary to count k because there is a new approximation of ∇L and the computation of a new descent direction d^k , but in truth the methods are not really comparable in the number of iterations.

Furthemore, it is partly a black box to know why the SGM converges a lot slower in number of iterations. If the values of α^k and d^k (step length and descent direction) are printed out, it is observed that for GM, the α grow to large values, whereas in SGM, the α^k are defined to hit a minimum and stabilize after a certain k iteration. This comes from the sufficient condition of the SGM in [1], which requires $\sum_{k=1}^{\infty}(\alpha^k)^2 < \infty$. In other words the step length cannot be left to grow like in the GM case. On a more general note, this makes sense because for this algorithm the computation of the gradient is inexact which could lead to wrong d^k. Finally, there also exists a theoretical result that shows that the error cannot decrease faster than $O(1/k)$.

1.2.2 Convergence speed in terms of λ

Here the effect of λ in each algorithm is taken into consideration For the Gradient Method and the Stochastic Gradient Method, the results are clear: for values $\lambda = 0.01, 0.1$ the number of iterations decreases. It has been stated before that the introduction of λ favors smaller weights, as well as a faster convergence. It is easy to see that the descent directions have a faster decrease due to presence of λ.

The expression for the derivative of the loss function has two main terms, the latter one being an additive term $\lambda\omega$. If the ω_i are smaller, the derivative of the loss function will be closer to 0 . So in the end, every iteration is a trade-off with the value of λ and $\|\omega\|$.

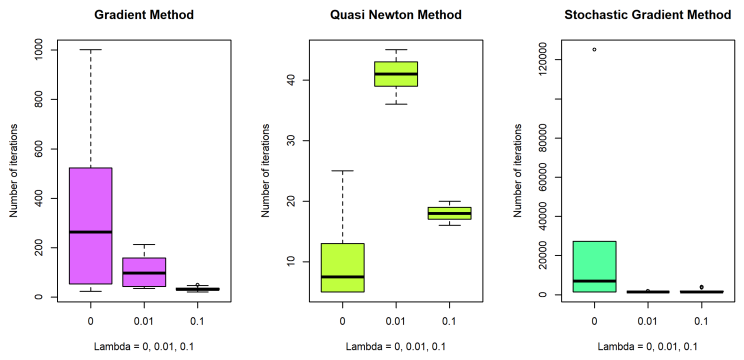

Figure 3: Number of iterations in terms of lambda

The box plot in Figure 3 shows surprising results for the Quasi Newton Method, which already has a very fast convergence for $\lambda = 0$. (mean of 10.2

iterations). With higher values for λ, the number of iterations increases. This a clear sign of the trade-off in the derivative of the loss function just mentioned. Also, in d^k, there is the effect of H^k, the BFGS matrix which is computed using $\nabla\tilde{L}^k$ and $\nabla\tilde{L}^{k-1}$.

The number of iterations for SGM is similar for $\lambda = 0.01$ and $\lambda = 0.1$: The mean is 1500 and 1988 respectively.

1.2.3 Running time per iteration

To compare the running time with the number of iterations, we calculate the ratio

$$r_m = \frac{\text{execution time}}{\text{iterations}} \qquad m = GM, QNM, SGM$$

and study its behavior. The three plots below (Figure 4) study the aforementioned ratio. It is convenient to fit a linear model and see if there exists a linear relationship between the execution time and the number of iterations. Note that if the slope > 1, that means that the execution time increases more than linearly with respect to the number of iterations. The three final give an adjusted R^2 over 95%, hence it can be concluded that there exists a linear relationship between the number of iterations and execution time. If any non-linear behavior were to be observed, than we could discuss bigger differences between algorithms, but that is not the case so we restrict ourselves to comparing the "speed" or the slope in each linear fit.

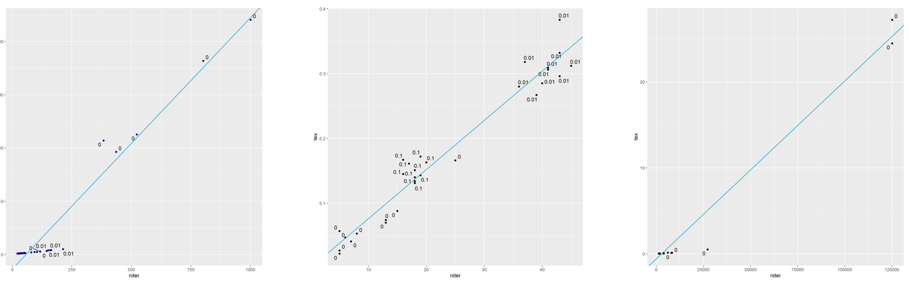

Figure 4: Ratio running time per number of iterations of the GM, QNM and SGM

The slope represents the speed of each algorithm. The linear fits have the following slopes: It can be inferred that the GM performs an iteration of the

Method	GM	QNM	SGM
Slope	0.04715	0.00758	0.00021

Table 3: Slope of the relation between Running time and iterations

method faster than QNM and SGM. However, it is obvious that this cannot be the only reason to favor the usage of one algorithm over another, because the QNM performs many less iterations to find the a solution.

1.3 Discussion

Find below a summary of results for the different algorithms

- Global convergence: For a small data set, they all do well in minimizing the loss function. Even with the penalization term, L^* is close to zero.
- Local convergence:
    1. QNM has quadratic convergence, whereas SGM and GM have at most linear. That is verified clearly when tracking two of the metrics that resemble "speed of convergence" like number of iterations and execution time.
    2. The presence of λ in the Loss function affects the number of iterations: In the GM and SGM case, it decreased but for QNM it increased.
    3. A linear dependence between the execution time and number of iterations has been shown. Also, the GM, has the highest speed of convergence if we define the speed has the ratio between the execution time and the number of iterations, followed by the QNM and SGM.

In conclusion, if it is believed that L^* is minimized in all cases considered, the best results for each lambda are in Table 4.

Method	GM	QNM	SGM
Lambda	0.1	0	0.01

Table 4: Best results after all the considerations above

2 Study of the recognition accuracy

First observe that the variable `tr_q = 0.0` meaning that all digits are represented equally in the data set, a more realistic case.

The results in terms of the test set accuracy for the data set of size 250 are summarized in Table .

Method	GM			QNM			SGM		
Lambda	0	0.01	0.1	0	0.01	0.1	0	0.01	0.1
Test acc.	98.64	99.04	95.6	98.6	99.04	95.6	99.04	98.88	96.48

Table 5: Test accuracy per λ value

So, for this section, we choose $\lambda = 0.01$ for GM and QNM and $\lambda = 0$ for SGM.

2.1 Training speed and recognition accuracy for a large data set

Table 6 shows the results of execution time and test accuracy for the data set of size 20000. Note that all values are presented with their mean across the numbers 1 through 10, which adds some bias.

Method	GM	QNM	SGM
Iterations	64.2	39.4	100100
Execution time	387.2	326.4	1195.59
Test acc.	99.05	99.05	99.62

Table 6: Mean of the iterations, speed and test accuracy for each model

From the table above, there is not really any of the methods that outperforms the others in terms of accuracy. For the values of λ chosen, the accuracy is nearly 100% in all three cases. The Stochastic Gradient Method is slightly better than the other, giving nearly 100% test accuracy on nearly all numbers. Differences are seen once again in the number of iterations and execution time. The Quasi Newton method is the best computationally speaking, although observing the table already shows that an iteration of the QNM is computationally more expensive than a GM one, as shown before.

2.2 Final comparison of results

In the first section, it was concluded that QNM was the best for the minimization of L^*, as it outperformed the other two methods in all metrics: execution time was the fastest and the number of iterations the smallest. For a larger data set, Stochastic Gradient Method seems to be the option by a slight margin, but the running time is almost four times as much. Perhaps with a more complicated detection problem one would see the SGM outperforming the QNM and GM in terms of recognition accuracy. Also, observe that a low value of λ (0, 0.01) seem to work better with smaller data sets, as the problem seemed to have little or no over-fitting. This low value is also maintained for the larger data set.

It is already seen here that the GM gets very close to the QNM in execution time when the data set is large. This is why it is important to point out that quadratic convergence is very remarkable in a optimization algorithm, but can come at a cost if ω^{k+1} is hard to obtain. One iteration of the QNM is costly because it requires many calculations in order to compute the BFGS matrix. It would be interesting to see how Newton Methods perform in comparison to the ones studied here.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[2] Anders Krogh and John Hertz. A simple weight decay can improve generalization. In J. Moody, S. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991.