OTDM - Neuronal Network Unconstrained Optimization

Prat Sicart, Joan
UPC Barcelona Tech
joan.prat.sicart@est.fib.upc.edu

Porta, Marcel
UPC Barcelona Tech
marcel.porta@est.fib.upc.edu

Saturday 19th October, 2019

This document supposes the deliverable of the Unconstrained Optimization part of the subject Optimization Techniques for Data Mining of the MIRI master. This deliverable is divided according to the statement provided by the lecturers and it's accompanied by the corresponding scripts.

1 Introduction

This assignment consists in build a single layer Neuronal Network able to identify whether the target values are in the data set pictures or not, since the pictures have 35 pixels, in the neuronal network there will be 35 input signals as well, where a sigmoid function will be applied to range their values between 0 and 1, then each of this inputs will be weighted by a variable W, and finally, the final output will be the sigmoid of the sum of the weighted different inputs. Therefore, the Neural network will be depicted as it follows:

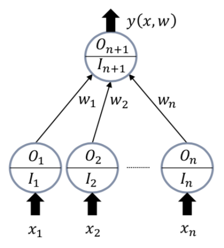


Figure 1: Neuronal Network schema

Moreover, to obtain the maximum accuracy, it's necessary minimize the lost function, and here is where Unconstrained Optimization part appears, therefore to solve this optimization problem will be applied First derivative methods such as the Gradient Method or the Quasi-Newton Method.

$$\tilde{L}(w; X^{TR}, y^{TR}, \lambda) = \sum_{j=1}^{p} \left(y\left(x_j^{TR}, w\right) - y_j^{TR}\right)^2 + \frac{\lambda}{2} \cdot \sum_{i=1}^{n} w_i^2$$

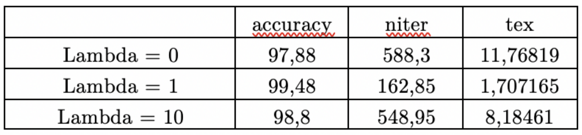
Figure 2: Objective function (loss function)

2 Results evaluation

The output of this project can be found in the file *uonnbatch.csv*

2.1 λ evaluation.

To obtain conclusions regarding how lambda effects on the final output, such as the number oft iterations, the execution time, and the accuracy, what we've done is calculate the mean of this this values for the different targets and methods in function of the value of lambda, and this are the obtained results:



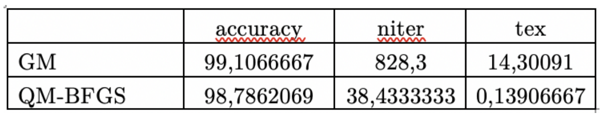
	accuracy	niter	tex
Lambda = 0	97,88	588,3	11,76819
Lambda = 1	99,48	162,85	1,707165
Lambda = 10	98,8	548,95	8,18461

Figure 3: values niter, tex, accuracy in function with lambda

The function of λ is exclusively regularize the objective function and by looking at the figure 2, we can observe that what does is help minimizing the function by telling them to look for lower w, therefore what does is add convergence to the objective function, however if the λ is too big, may enter in conflict with the objective function itself (without the regularization), and may have two different points of convergence one due to the objective function itself and the other one due to the regularization term (which is where all the w=0). Therefore, to summarize, a λ has to had the optimal value to help to converge the objective function, otherwise, if it's too big will just confuse and more iterations will be needed (The reasoning regarding accuracy in section 2.3).

2.2 First Derivative Methods evaluation.

We performed the same strategy as before to see how the different methods (GM and QM-BFGS) affect the final output (The reasoning regarding accuracy in section 2.3):



	accuracy	niter	tex
GM	99,1066667	828,3	14,30091
QM-BFGS	98,7862069	38,4333333	0,13906667

Figure 4: values niter, tex, accuracy in function with the method

As it can be appreciated the GM needs further more iterations to converge than the QM-BFGS, however the execution time for each method it's not proportionally with the number iterations, for instance the time per iteration for the GM is: $t_{ex}/n_{iter} = 0,017265375$, and for the QM-BFGS is: $t_{ex}/n_{iter} = 0,003618387$ that's mainly because in the difference between directions, a better direction which is the case of the QM-BFGS ease to find an alpha that full fill the strong Wolfe Conditions, and that's why an iteration of the QM-BFGS takes less time than and iteration of the GM because of the iterations needed to find a good alpha.
Regarding the number of iterations of the different methods is due to the different local convergence of them, while GM has a linear order of convergence, QM-BFGS has a superlinear order of convergence when it's close to the solution.

2.3 Accuracy reasoning.

As it can be appreciated in the figures 3 and 4 the accuracy depends on both lambda parameter and method applied to implement the unconstrained optimization. In one hand, the lambda may affect the accuracy improving it adding a little bit of convergence or make it worse by means of giving a value too large and in this case changing the optimal point of the original lost function to another distinct closer

to the point where all the W are 0, and it's obvious that a NN where all the W are 0 won't have a good accuracy (arround 50%).
In the other hand, even though the QM has a super linear order while the GM just has a linear order of convergence, in the GM the global search is always guaranteed since that always full fill the descendent condition and Zountendijk (CAC) condition however the QM may not always have a global convergence, because although will also have always a descendent direction (if matrix Bbfgs is + def), the Zountendijk (CAC) condition may be rejected, so that's because the GM have better accuracy than QM.

2.4 Analizing the worst case.

The worst case is given when analysing as a target the number 8 with a lambda of value 0 and the QM as the chosen method, therefore the code will be ran with the following parameters:

```
uo_nn_solve(8,0.5,47904864,250,47904864,250,0,10^-6,5000,1,2,30,10^-3,0.01,0.45,3,2,2,1,0)
```

And the plot from this the following:

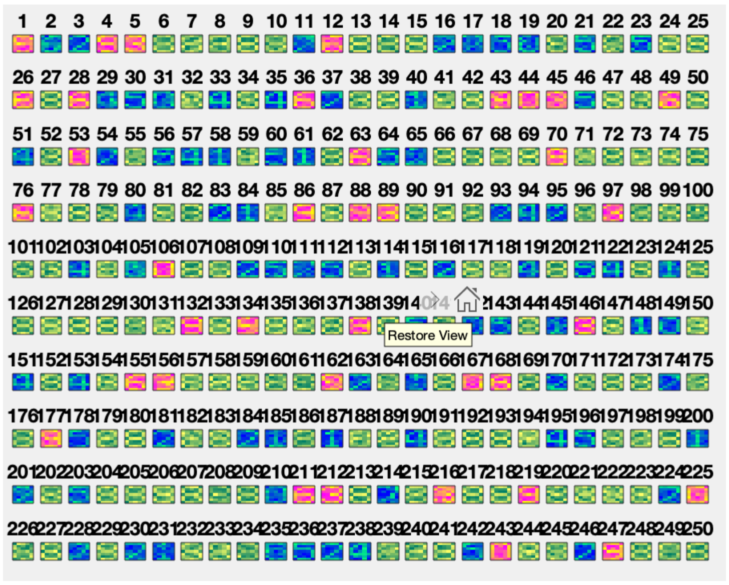


Figure 5: target:8,isd:3,lambda:0

With this parameters the accuracy for the training set it's only 85 % and that's because as we've said before the QM may not always have a Global convergence, because although will also have always a descendent direction (if matrix Bbfgs is + def), the Zountendijk (CAC) condition may be rejected (due to big condition numbers).

3 Recognise series of 5 digits

In order to make the GM and QM able to detect blurred sequences, we've followed the next procedure:

First of all, we've modified the data set creation code, to output as $y = 1$ for all the digits that full fill the sequence target. In the figure below can be appreciated the plot of this dataset, as don't have W's predicted all the predictions will be wrong:

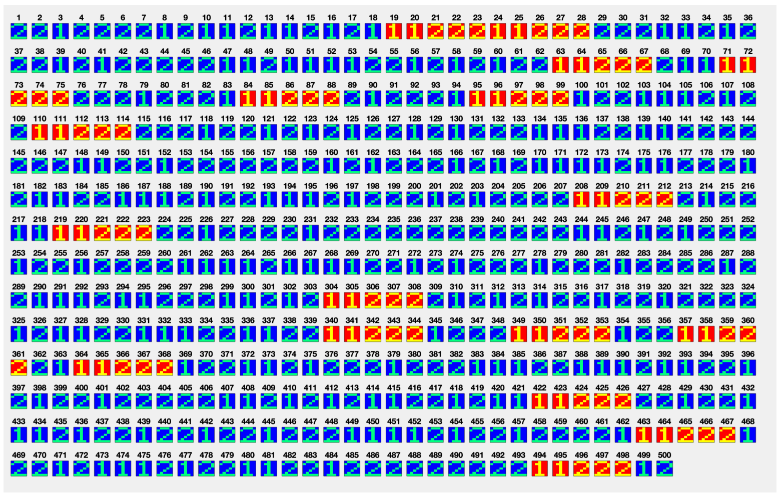


Figure 6: sequence dataset generated

In the next step, we've computed the unconstrained optimization algorithms as X the values of the actual position plus all the 4 previous positions, therefore now we didn't had a matrix $X(35, numColimns)$ now we had $X(175, numColimns)$ since $175 = 5 * 35$ and for the Y was also needed to be modified a little bite, since just we wanted to be as 1 the last y position of the sequence detected, as will be the only one that the X in this position will have the 5 numbers of the targets, and the algorithms will be able to perform a good calculation.

So finally, we just needed to change few lines of code from plot function to adapt it to the new dimensions and transformations made in the matrices X and Y, and with the following execution for instance:

```
uo_nn_solve_sequence([1,1,2,2,2],1,123456,500,47904864,500,1,10^-6,5000,1,2,30,10^-3,
0.01,0.45,1,2,2,1,0)
```

The output was the following:

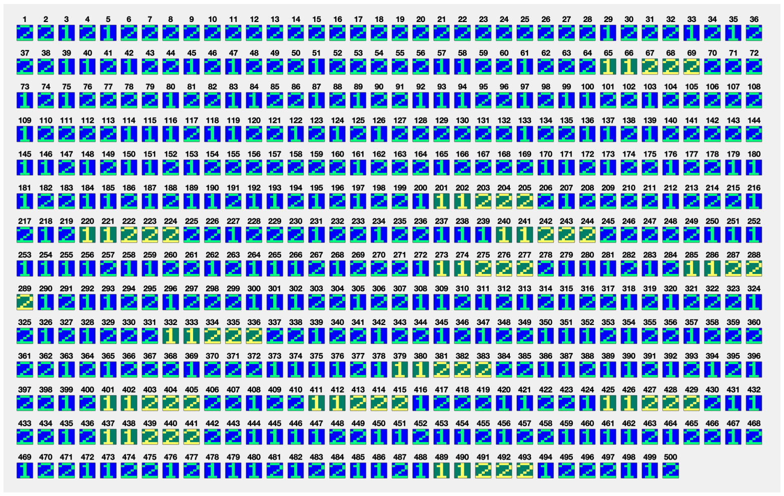


Figure 7: prediction plot with clear digits

As it can be appreciated in the image above, since the digits were clear the accuracy was 100 %, blurring the digits we we obtained a accuracy around the 95 %.

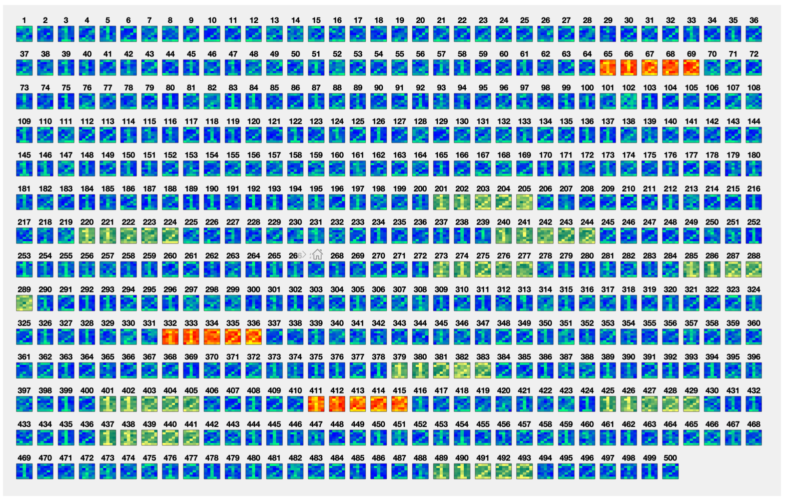


Figure 8: prediction plot with blurred digits

To conclude, in order to increment the probabilities of having sequences and therefore prove the performance the code, we've set the frequency as 1 and the number of training's and tests to 500. To observe the performance of the code for the 10 different random sets of 5 digits you can find attached the uonnsequencebatch.csv which is the result of running the uonnbatch.m file.