OTDM Unconstrained Optimization Lab Assignment: Pattern recognition with Single Layer Neural Network

Mikołaj Małkiński, Egon Ferri

20th of OCTOBER 2019

Introduction

The aim of this project is to develop an application, based on the unconstrained optimization algorithms studied in this course, that allow to recognize the numbers in a sequence of blurred digits. The procedure to achieve that goal will be to formulate a Single Layer Neural Network that is going to be trained to recognize the different numbers with First Derivative Optimization methods.

All the code and the partial result can be found in the folder. At the end of this document there is a summary that describe them.

TASK a) Based on the data collected with the batch file, determine:

Which is the value of the regularization parameter λ that gives the best results for the overall set of digits and optimization methods.

Since two different formulas to calculate the $\alpha^{\max}$ was given, we decided to add another parameter to our training loop to evaluate the different performance of the neural network with different $\alpha^{\max}$ settings. The three formula tried are:

Static:

$$\alpha_1^{\max} = 2;$$

Dynamic:

$$\alpha_2^{\max} = \alpha^{k-1} \frac{\nabla f^{k-1^T} d^{k-1}}{\nabla f^{k^T} d^k};$$

$$\alpha_3^{\max} = \frac{2\left(f^k - f^{k-1}\right)}{\nabla f^{k^T} d^k}$$

Since the $\alpha_2^{\max}$ formula is clearly the best (the other two formulas fail to reach an high accuracy more often), we only show and comment the result reached with that. The complete csv can be consulted to see what happened with the other two formulas.

We also decided to cut the maximum number of iterations to 1000, since empirically it seems enough as upper value.

The table shows, in order: The target number, the lambda, the algorithm used for the descent direction (1= GM, 3= QM-BFGS), the formula used to compute the $\alpha^{\max}$, the number of iteration, the execution time and the accuracy on the test dataset.

1	0.0	1	2	4	0.0540	100.0
1	0.0	3	2	4	0.0546	100.0
1	1.0	1	2	103	0.2992	100.0
1	1.0	3	2	36	0.1475	100.0
1	10.0	1	2	53	0.1942	100.0
1	10.0	3	2	544	13.882	100.0
2	0.0	1	2	15	0.0861	100.0
2	0.0	3	2	406	0.9774	80.8
2	1.0	1	2	304	0.8591	100.0
2	1.0	3	2	63	0.2391	100.0
2	10.0	1	2	103	0.3397	100.0
2	10.0	3	2	1000	26.338	100.0
3	0.0	1	2	1000	27.844	98.0
3	0.0	3	2	1000	23.051	98.0
3	1.0	1	2	552	14.168	99.2
3	1.0	3	2	86	0.2868	99.2
3	10.0	1	2	153	0.4472	99.2
3	10.0	3	2	1000	22.043	99.2
4	0.0	1	2	3	0.0549	100.0
4	0.0	3	2	3	0.0516	100.0
4	1.0	1	2	118	0.3521	100.0
4	1.0	3	2	35	0.1477	100.0
4	10.0	1	2	49	0.1917	100.0
4	10.0	3	2	1000	21.520	100.0
5	0.0	1	2	5	0.0543	99.6
5	0.0	3	2	5	0.0632	99.6
5	1.0	1	2	201	0.5358	99.6
5	1.0	3	2	51	0.1791	99.6
5	10.0	1	2	74	0.2364	99.6
5	10.0	3	2	130	0.3541	99.6

6	0.0	1	2	3	0.0437	99.6
6	0.0	3	2	3	0.0437	99.6
6	1.0	1	2	297	0.7781	100.0
6	1.0	3	2	72	0.2323	100.0
6	10.0	1	2	113	0.3333	99.6
6	10.0	3	2	156	0.4214	99.6
7	0.0	1	2	46	0.1502	98.8
7	0.0	3	2	1000	21.396	99.2
7	1.0	1	2	165	0.4793	100.0
7	1.0	3	2	1000	23.304	100.0
7	10.0	1	2	61	0.2033	99.6
7	10.0	3	2	539	13.163	99.6
8	0.0	1	2	1000	23.855	97.2
8	0.0	3	2	1000	22.309	97.2
8	1.0	1	2	743	18.993	98.0
8	1.0	3	2	1000	22.300	98.0
8	10.0	1	2	194	0.5459	91.6
8	10.0	3	2	1000	21.912	90.0
9	0.0	1	2	1000	24.969	96.8
9	0.0	3	2	1000	23.110	98.8
9	1.0	1	2	502	13.140	99.2
9	1.0	3	2	1000	23.019	99.2
9	10.0	1	2	154	0.4478	99.2
9	10.0	3	2	1000	21.698	98.4
0	0.0	1	2	408	10.511	99.2
0	0.0	3	2	1000	22.023	98.0
0	1.0	1	2	362	0.9371	99.2
0	1.0	3	2	155	0.4649	99.2
0	10.0	1	2	127	0.3874	99.2
0	10.0	3	2	1000	21.226	88.8

The best value for the parameter lambda it's clearly 1.0, in fact, ceteris paribus, the best accuracy is reached, and analyzing the number of iteration and execution time, we see that this results holds without getting worse on performance.

Which is the best optimization algorithm, GM, or $QM - BFGS$, given λ determined in the previous section, based on the analysis of the variables $accuracy^{TE}$, $niter$ and tex. Describe how the execution-time per iteration ($\frac{tex}{niter}$) behaves for the two different methods and find an explanation.

Once we fix $\lambda = 1$ and we use the first of the dynamic formulas to calculate $\alpha^{\max}$, we can analyze the performance of optimization algorithms.

We see that, in term of accuracy, the two algorithms have exactly the same predictive power. Looking at the number and time of iterations, we get ambiguous result. Sometimes the gradient descent performs better, sometimes worse. We could expect that a better time performance from the GM algorithm, because it is simpler on a computational level, but we see that the computational time is almost the same because, since we are working with problems that are not so complicated and the time consumption is not a lot, we preferred

num_target	la	isd	typealma	niter	tex	te_acc	tex/te_acc
1	1.0	1	2	103	0.3371	100.0	0.0034
1	1.0	3	2	36	0.1558	100.0	0.0016
2	1.0	1	2	304	0.8567	100.0	0.0086
2	1.0	3	2	63	0.2205	100.0	0.0022
3	1.0	1	2	552	15.411	99.2	0.0155
3	1.0	3	2	86	0.3277	99.2	0.0033
4	1.0	1	2	118	0.3851	100.0	0.0039
4	1.0	3	2	35	0.2108	100.0	0.0021
5	1.0	1	2	201	0.6308	99.6	0.0063
5	1.0	3	2	51	0.2130	99.6	0.0021
6	1.0	1	2	297	0.9119	100.0	0.0091
6	1.0	3	2	72	0.3032	100.0	0.0030
7	1.0	1	2	165	0.5283	100.0	0.0053
7	1.0	3	2	1000	26.202	100.0	0.0262
8	1.0	1	2	743	19.780	98.0	0.0202
8	1.0	3	2	1000	22.652	98.0	0.0231
9	1.0	1	2	502	13.189	99.2	0.0133
9	1.0	3	2	1000	22.847	99.2	0.0230
0	1.0	1	2	362	0.9733	99.2	0.0098
0	1.0	3	2	155	0.4411	99.2	0.0044

to have more compact and clear functions instead of optimized functions not-so-easy to be read, so in this case the time is similar because our formula computes the hessian also in the GM, but it does not use it to find the direction.

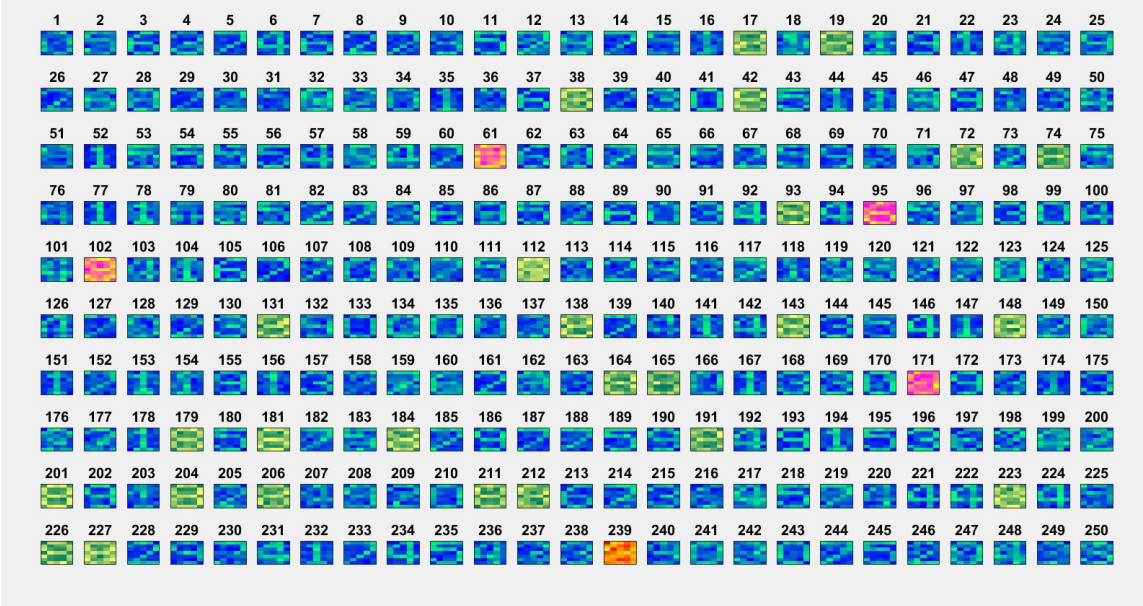
TASK b) Describe how the accuracy of the pattern recognition ($accuracy^{TE}$) depends on the digit for the value of λ and optimization method determined in section a). For the digit with the worst value of $accuracy^{TE}$, try to guess the reasons for the bad recognition rate with the help of function that plots the result.

From our batch it's clear that ,at least in our case, once λ is fixed, the accuracy depends only on the digits (since also the optimization methods give the same results, in different number of iterations).

In general we have a really satisfying result since all the accuracies are really close to 100%.

The worst value is given by the digit 8, the result over the test is the following:

We see that we have five errors of two different types:

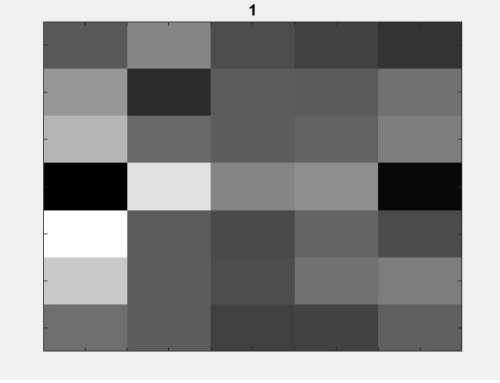


4 false positives (a 3, a 0, a 9 and a number that is so blurred that is hard to recognize even for us).

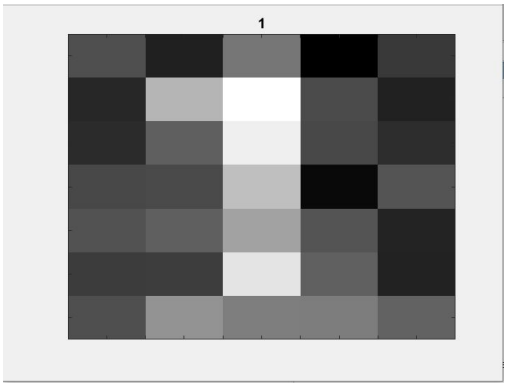
1 false negative (an 8 heavy blurred)

The 8 can be difficult to find for the neural net because it is similar to a lot of numbers of our set $(0, 3, 6, 9)$.

We can also plot the optimal weight of the 8 compared to the optimal weight of a very easy number like the 1 to see that the weights of the 8 are a lot less defined.



(a) w0 of number 8



(b) w0 of number 1

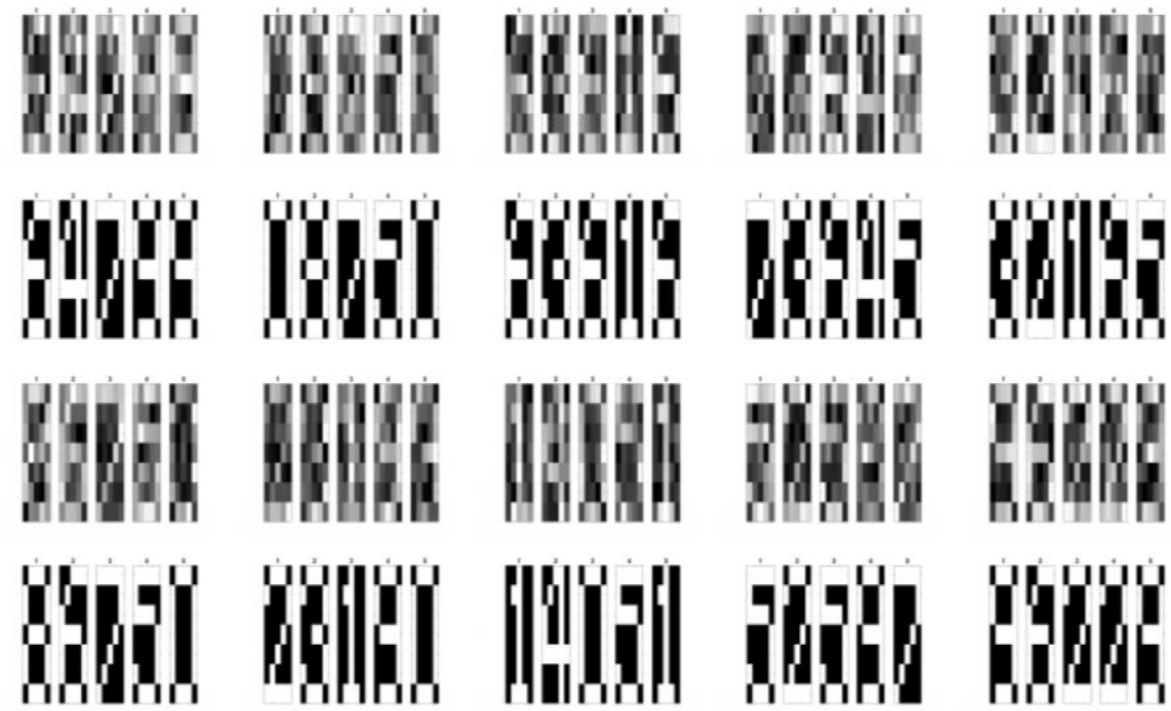
TASK c) Make use of the trained SLNN to develop a function that can identify series of 5 digits.

To use our neural networks that only gives 0 and 1 as a result of the question "is the digit 1?" or "does the digit belong to the set [0,1,2]?" we structured a sort of binary search: At the beginning we ask if the number is round (= is in the set [0,3,6,8,9]) than if is round we ask if is in the set [0,3,8], if is in that we ask if it is in [3,8] and in the end we ask if it is a 3 or not. If we get a no at some point, we proceed in the same way symmetrically (more details in the code).

We defined a function that takes an array and a seed, trains the network with the parameters that we defined in exercise a) and performs this binary-tree search on each element of the array. We also modified the plot functions to have a visualization of our result.

We repeated the analysis and the results are the following: in the first and third rows we have the input blurred data, in the second and fourth rows the visualization of our result.

The matching seems almost perfect.



uo nn solve.m — function of the optimization algorithm
uo nn batch.m — main to solve task $a)_1$
uo nn batch2.m — main to solve task $a)_2$
main2.m — main to solve task b)
main3.m — main to solve task c)
identifythenumber.m — function that takes a blurred array and returns the predicted array
uo nn dataset modified.m — function that plots a clean array

uo nn dataset.m —
uo BLSNW32.m — GIVEN
uo nn Xyplot.m— FUNCTIONS
uo solve plot.m —

identification results — partial results in HD for the 10 simulations
results clean — partial results in HD

uo nn batch.csv — batch with all parameters
uo nn batch2.csv — batch with only optimal parameters for lambda and almax
uo nn.pdf — source of tasks