Unconstrained Optimization Lab Assignment

Pattern recognition with SLNN

Single Layer Neural Network

David Bergés Lladó i Roser Cantenys Sabà

Table of Contents

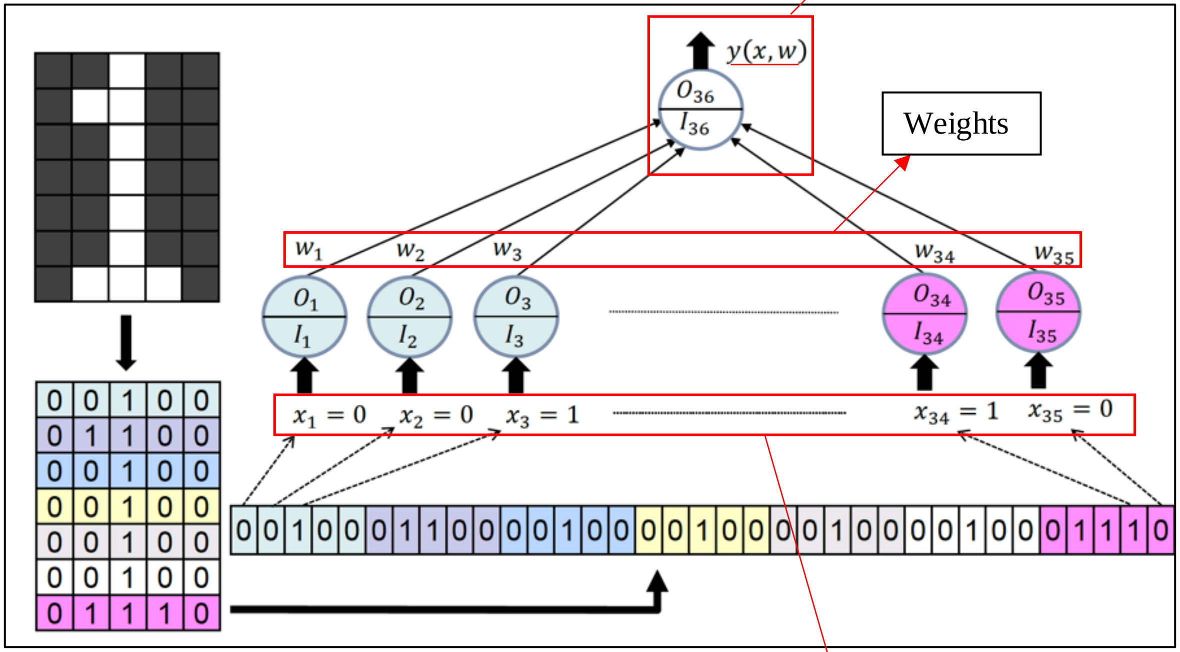
INTRODUCTION

In this work, we aim to classify the numbers between zero and nine (both included). To do it, we are going to use a Single Layer Neural Network (SLNN). Each number is represented by a picture of 35 binary pixels(7x5) which are vectorized and taken as the inputs of the SLNN. As we have a single layer, we have 35 neurons and each one will represent a pixel.

We have to find the 35 weights associated with each neuron, connected to the output layer. The latter will provide a probability value which indicates whether the digit is in the target set (it can be a single number or a set of numbers), or not.

The process is shown in the following figure:

Figure 1. SLNN process


Output signal defined by the function y(x,w)
$y(x,w)$
O_{36}
I_{36}
Weights
w_1 w_2 w_3 w_{34} w_{35}
O_1 I_1 O_2 I_2 O_3 I_3 O_{34} I_{34} O_{35} I_{35}
$x_1 = 0$ $x_2 = 0$ $x_3 = 1$ $x_{34} = 1$ $x_{35} = 0$
0 0 1 0 0
0 1 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 1 1 1 0
0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0
Single entry of the NN

METHODS

In this section, we are going to address the problem explained before and the mathematics behind this.

2.1 Neural network

First of all, we have to define what is a neural network. A neural network is a biological inspired programming technique which enables the computer to learn from observational data.

More specifically, it is a network of simple elements called artificial neurons, which receive input, change their internal state (activation) according to that input, and produce output depending on the input and activation.

In our case, we have a single layer of 35 neurons, each one representing a single pixel of the 7x5 vectorized image, as we have seen in the figure displayed in the introduction. Our activation function is the sigmoid function $O_i = \sigma(I_i),\ \ \sigma(x) = \frac{1}{1+e^{-x}}$, which is a very special function, because it is a bounded (0,1), differentiable, real and it is defined for all real input values that allows us to convert any value into probabilities. As our network has only a single layer, the output signal is defined as follows: $y\left(x_j^{TR}, w\right) = (1 + e^{-\sum_{i=1}^{n} w_i \cdot \left(1+e^{-x_{i,j}^{TR}}\right)^{-1}})^{-1}$ and returns the probability that a particular image is from our target values.

Each neuron generates a weight and all of them, with their activation function, are connected to the output layer which provides a probability value that indicates whether the digit is in the target set or not. As we want an accurate model such that classifies unseen examples almost perfectly we have to consider the error made at each prediction and try to minimize it. We want to minimize the output's error, so we have to minimize the error committed at each prediction.

2.2 Loss function

As the NN is reduced to an optimization problem, we have to minimize a function.

The function we want to minimize is:

Equation 1. Function we want to minimize

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

where $y\left(x_j^{TR}, w\right) = (1 + e^{-\sum_{i=1}^{n} w_i \cdot \left(1+e^{-x_{i,j}^{TR}}\right)^{-1}})^{-1}$; $w =$ weight vector; $y =$ target vector; $\lambda =$ parameter of L2 regularization

The last term in the previous equation is the L2 regularization with parameter λ that allows us to penalize overfitting.

2.3 First derivative methods

We want to use the backtracking linesearch algorithm but it cannot handle conveniently the SLNN problem, so we have to introduce two modifications in the computation of the linesearch. The first one consists in changing the maximum step length. Unlike the Alg.BLS it cannot be a constant for every iteration. Instead, it must be updated dynamically using information of the local behavior of f near the iterated point at each iteration. In our case, we are going to use the following formula taken from N&W page 58.

Equation 2. Maximum step length updated dynamically

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

The second modification that we have to apply is a BLS based on interpolations, as the one proposed in Alg. 3.2 and 3.3 of N&W, implemented in the function that we were given om_uo_BLSNW32.

To minimize the loss function, we are going to use first derivative methods seen at class. We will use: Gradient method, Conjugate Gradient method and BFGS method.

As regards the conjugate gradient method we will try Polak-Ribiére and Fletcher-Reeves methods but we will only use restart condition 2 of parameter 0.1 since we don't know the optimal number of iterations that we have to choose to apply restart condition 1. Therefore, we will restart when $\frac{\nabla f^{k^T} \nabla f^{k-1}}{||\nabla f^k||^2} \geq 0.1$

Finally, we have to define the constants with which all the experiments have been carried out. These are shown in the following table and have the same names as in the scrip.

Table 1. Experiment constants

tr_freq	noise_freq	tr_p	te_p	tr_seed	te_seed	la	eps	kmax	almax	kmaxBLS	epsBLS	c1	c2	nu
0.5	0.1 // 0.2	500	5000	123456	347	0	1e-6	1000	2	30	1e-3	0.01	0.9	0.1

So far, we have seen how our artificial neural network works internally in order to make predictions for a specific target over a given set of data. Now it's time to estimate the accuracy of our predictions and make sure we are not overfitting the training data. That is, when our accuracy with the training set is way bigger than the one obtained with the testing set. In other words, we want to make sure that our NN has not memorized samples but learnt patrons.

In this application, we know the outputs for both sets before testing them, so that we can compare the true value with the one obtained with our predictions, thus we can calculate the accuracy:

Equation 3. Accuracy

$$Accuracy^{TR} = \frac{100}{p} \cdot \sum_{j=1}^{p} \delta_{\left[y\left(x_j^{TR}, w*\right)\right], y_j^{TR}} \; ; \quad Accuracy^{TE} = \frac{100}{p} \cdot \sum_{j=1}^{p} \delta_{\left[y\left(x_j^{TE}, w*\right)\right], y_j^{TE}}$$

ASSIGNMENT

3.1 FIRST TASK

In this part, we will solve the pattern recognition problem for **`num_target=[4]`** and **`num_target=[8]`** (separately) with all the optimization methods and variants studied in this course. We will only take the first order methods (gradient, conjugate gradient and BFGS). We take $k^{max} = 100$ and $\lambda = 0.0$. We will report the behavior observed for each method with a table and we will analyze the convergence of each method and the value of $Accuracy^{TR}$(training accuracy) and $Accuracy^{TE}$(test accuracy).

Table 2. Methods and results

num_target	*isd*	*icg*	*irc*	*k*	*grad_g*	*L_opt*	*tr_accuracy*	*te_accuracy*
4	1	-	-	1000	0.0037	0.0078	100	100
8	1	-	-	1000	1.5875	21.6812	95.80	96.68
4	2	1	2	1000	1.52e-4	3.61e-4	100	100
8	2	1	2	1000	0.8752	17.3448	96.20	96.78
4	2	2	2	1000	4.66e-4	8.57e-4	100	100
8	2	2	2	1000	1.2834	18.4079	96.20	96.72
4	3	-	-	23	2.95e-8	4.10e-9	100	99.8
8	3	-	-	37	2.19e-77	8	98.4	95.74

ACCURACIES

In this experiment, all methods are capable of predicting digit 4 with a 100% of train and test accuracy, except BFGS that provides a 99.8% of test accuracy.

On the other hand, if we take a look at the accuracies of number 8 we will see that we obtain noticeably worse results. However, despite this decrease in performance, all our accuracies are above the 95% because we have taken very large samples in order to train our SLNN.

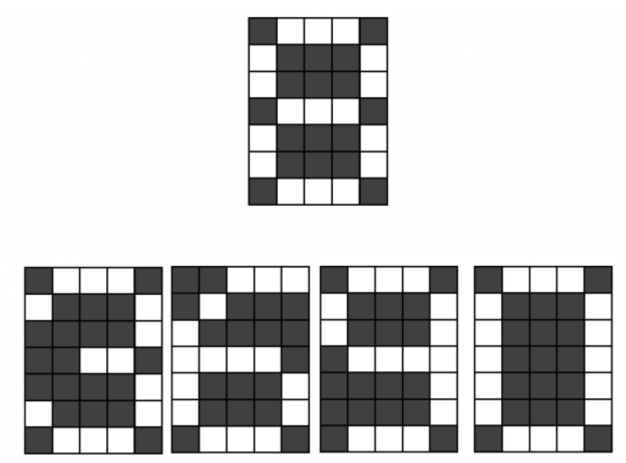
We can explain it as any method applied is effective enough to make loss function (L_opt) close to zero and $||g*|| \neq 0$ (grad_g $\neq 0$).

So far we have done all of our testing with tr_p = 500 and te_p = 5000, but if we take different values, for example tr_p = 100 and te_p = 1000, notice that we get slightly different results for the prediction of number 8: we can see an increase with the training accuracy and a decrease of the testing accuracy. For example, BFGS method with these new parameters gets a tr_accuracy of 99% and a te_accuracy of 89.9%.

This leads us to think that lower values of tr_p tend to overfit the data, so that we get worse test accuracy results since $tr_{accuracy} \gg te_{accuracy}$.

Another conclusion from our experiment would be that predicting number 4 is easier than predicting number 8. It is clearly shown in the results displayed before, and we can explain this variation visualy. It turns out that number 8 is pretty similar to other numbers like 0, 3, 6 or 9, with which it shares a lot of bits as we can see in the following image:

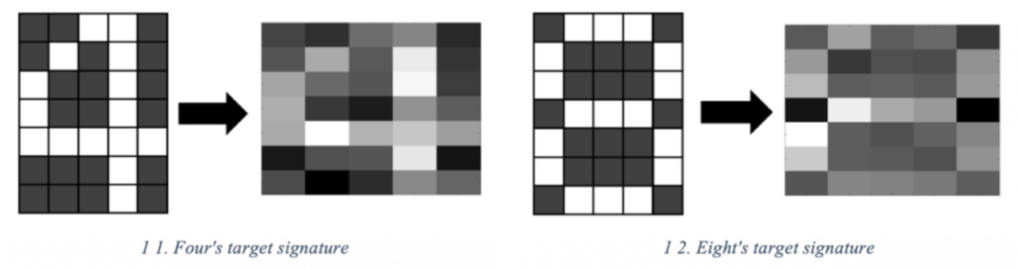
Figure 2. Similar numbers



When you add some kind of noise frequency (10% in our case), it is more difficult for our NN to identify this similar numbers, because they might look like other digits because of the distorsion.

We can also draw more conclusions from the target signature shown below:

Figure 3. Target signature



If we compare both signatures, we see that the one related with number 4 is closer to the original than it is the one related to 8 with his.

We see more bits with high level of whites within its real form, and more bits with very little level of white that are clearly not from number 4.

These bits characterize the optimal weights that the NN returned after its training, and it's straight forward to conclude that it will be easier to predict numbers with more discriminatory bits rather than more homogeneous signatures as the one that represents number 8 in the figure above.

CONVERGENCE

Regarding local convergence, we can see that the BFGS is capable to find better descent directions, as the number of iterations needed in order to reach the stopping criterion is way smaller than the other methods. Hence that neither the gradient nor the conjugate gradient methods are able to find a stationary point in less than 1000 iterations.

This could indicate that the latter two methods might have been stuck in some neighborhood of points that are close to stationary. This causes our optimization algorithms to keep moving with very small steps, $-d_k \nabla f$, that are actually close to 0 but are not small enough to meet the stopping criterion, $||\mathrm{g}*|| \leq \epsilon$, with $\epsilon = 1\mathrm{e}-6$. In other words, the algorithm keeps doing iterations of very small moving steps and it reaches the maximum number of iterations upper bound before it is capable to find the stationary point.

More specifically, we can see that for target #4 all methods are able to find an optimal solution, as the gradients (grad_g) are really close to 0, fact that means that their global convergence is good. However, we cannot make the same statement for number 8, since gradients take values bigger than 0, so it looks like they are not able to find optimal solutions and therefore their global convergence is worse.

The conjugate gradient method with restart condition 2 works better with Fletcher-Reeves beta update as ||g*|| and L_opt are closer to 0. Nevertheless, the best method as far as convergence is concerned for number 8 is the BFGS in all aspects, it has the smallest ||g*|| and L_opt.

3.2 SECOND TASK

In this part, we will solve the pattern recognition problem for every digit from 0 to 9 with BFGS mehod, $k^{max} = 1000,\ \lambda = 0.0,\ tr_{freq} = 0.5\ and\ noise_{freq} = 0.2$. We are going to report the behavior observed for each method with a table and compare the test accuracy for every digit and try to find an explanation for the observed differences.

Table 3. Results

num_target	*k*	*grad_g*	*L_opt*	*tr_accuracy*	*te_accuracy*
0	29	2.68e-09	2.00	99.60	95.90
1	12	2.20e-08	2.20e-08	100	99.74
2	28	7.10e-26	9.85e-27	100	97.06
3	51	9.05e-12	8.00	98.40	92.12
4	15	1.03e-16	1.41e-17	100	99.10
5	24	1.89e-13	2.67e-14	100	98.08
6	25	1.39e-30	1.97e-31	100	98.42
7	19	7.74e-24	1.00	99.80	98.10
8	35	1.05e-16	13.00	97.40	90.56
9	26	1.14e-17	6.00	98.80	92.16

The table displayed above reflects that there are numbers that are easier to predict than others. Numbers in subset $C_1 = \{0,3,8,9\}$ have a test accuracy lower than 96% while in subset $C_2 = \{1,2,4,5,6,7\}$ is greater than 97%. Thus, one could state that we two clearly differentiated groups, one where numbers are similar between each other and therefore are harder to distinguish and one that apparently present less similarities and can be predicted more accurately.

Figure 4. C1

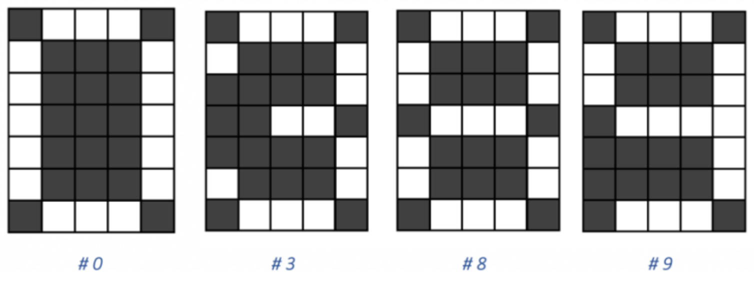
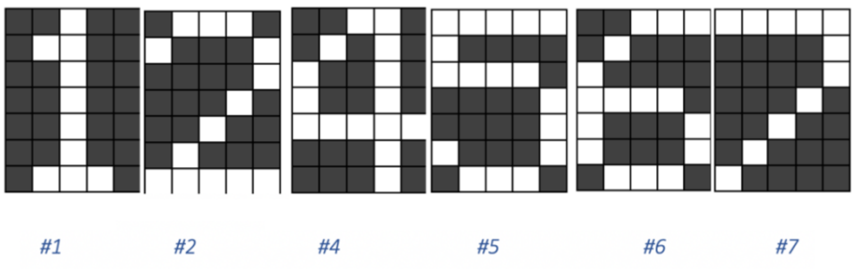


Figure 5. C2



All the experiments in this part have been performed with the BFGS method. We are able to find stationary points for all the targets with a reduced number of iterations. In the worst-case scenario, the number 3, it only takes 51 iterations so we can consider that the algorithm is quite robust.

If we now focus on the two sets described before, we observe that numbers forming C1 have an L_opt far from 0, which is the expected value of the optimization (we seek for zero loss in our predictions). However, numbers of C2 have an L_opt which is indeed close to 0. This can be explained, as we already did in the first task, as a consequence of the similarities between numbers of group 1. Apart from that, if we take into account that we are adding more noise to the image generation (20% specifically), it is still more difficult to distinguish between two different numbers of that group as there are very few pixels that mark the difference. Group 2 however, looks like has more accurate patrons in every digit and that makes prediction easier, and in consequence, a higher accuracy.

If we train both groups with less data, the first one is affected just like number 8 in the first task. That is, we would increase our training accuracy in exchange of a decrease in testing accuracy. In other words, we would be overfitting. However, the second one obtains results very similar to those observed in the table. We have experimentally checked that we can increment the first subset accuracy by augmenting tr_p and te_p parameters. Even though using tr_p = 1000 and te_p = 10000 respectively (duplicating the values used in the previous table) we achieve very small improvements (all of them less than 1%).

In conclusion, subset {1,2,4,5,6,7} is easier to determine than subset {0,3,8,9}.

BIBLIOGRAPHY

- *Numerical Optimization*, Jorge Nocedal and Stephen J. Wright, 1999, ISBN 0-387-98793-2 http://www.bioinfo.org.cn/~wangchao/maa/Numerical_Optimization.pdf
- Unconstrained optimization, mathematical optimization slides of GCED 2019, Javier Heredia.