Pattern Recognition with Single Layer Neural Network

Oriol Domingo Roig, Roger Creus Castanyer

May 2019

1 Introduction

In this work we aim to classify a set of numbers using a **Data Mining** technique, known as **Neural Network**. We are going to use a Single Layer Neural Network that will be trained on pictures of size 7x5 pixels per image. Thus, Layer will consist of 35 neurons, each one representing a pixel. Consequently, we will have 35 weights that link a final node with the previous nodes and assign a probability value. This value represents whether the number is in the target number set or not.

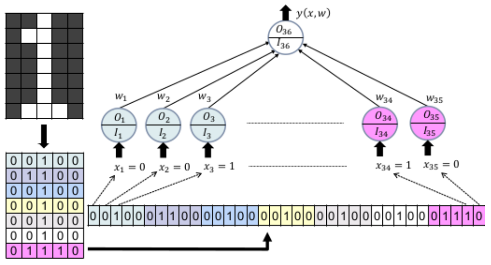


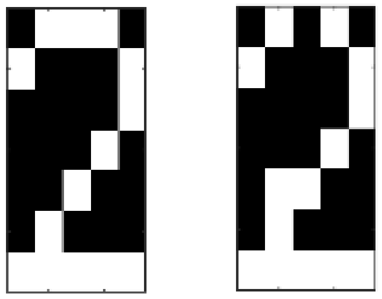
Figure 1: Training Model [1]

2 Materials and Methods

This project is divided into three components which ease the understanding of global workflow. Firstly, we developed some scripts that generate training data and test data. Both data sets play a specific role in the whole project. Then, it was necessary to build some objective functions that represents the work of the Neural Network. These functions were nurtured by training data. Following, we apply some mathematical programming algorithm to find the optimum value of such functions. Finally, our team reported the behavior of each model using some performance metrics respect to training and test data.

2.1 Data Generation

Training Data and Test Data is generated from the file *"dataSets"* which requires two seeds, number of samples in the training data, target number data set, frequency of target in data and image noise frequency. It is necessary to specify seeds in order to allow users to recompute model parameters under the same conditions. Moreover, providing the number of samples in the training data is enough, since test data will be built using ten times the amount of training data. Finally, image noise frequency is a mechanism to add some noise in an image, i.e. changing some bits from 0 to 1 and vice versa, then not only is it harder for the model to predict, but also makes it more realistic.



(a) Without noise (b) With noise

Figure 2: Sample

Gathering all this information, *dataSets* is capable of building training and test data under the specific conditions that we imposed. Such script returns four parameters, the first and the third correspond to a matrix in which each row represents 1 pixel out of 35, and each column represent a whole number. Hence, all matrix have the same amount of rows, 35, and differ in terms of columns, number of samples. Then, second and fourth parameters correspond to a Boolean vector representing whether such sample belongs to the target number set or not. First and second parameter is used for training and third and fourth for test.

2.2 Neural Network Functions

The goal of our Neural Network is to train a model in such a way that it is highly accurate when classifying unseen examples. Thus, this model must be built considering the error at each prediction. Our model works with an input, in this case will be an image of 35 pixels with 0 or 1 value, representing each one a neuron. Then, some weights are applied to each neuron and linked all of them to a single value, last neuron, the output. This value is a probability representing whether the image correspond to the target number set or not, as shown in Figure 1.

This result into a loss function that works with some weights, which play a non-linear role respect to input values, and consider predicted values. Moreover, we should apply a common technique in mathematics, known as **L2 regularization**, in order to penalize models that tend to over-fit training data.

$$L(X, y, \lambda) = min_{w \in R^n} L(w; X, y, \lambda) = \sum_{j=1}^{p} (y(x_j, w) - y_j)^2 + \lambda \frac{||w||^2}{2} \tag{1}$$

$$y(x, w) = (1 + e^{-\sum_{i=1}^{n} w_i (1+e^{-x_i})^{-1}})^{-1} \tag{2}$$

Hyper-parameter definition:

- X_{np} = Training data matrix
- y_p = Target vector

Parameter definition:

- w_n = Weight vector

It is necessary to find the optimal values of those weights that minimize the error between predictions and target. It entails to work with first derivative respect to weights.

$$\frac{\partial L(w; X, y, \lambda)}{\partial w_i} = \sum_{j=1}^{p} 2((y(x_j, w) - y_j) y(x_j, w)(1 - y(x_j, w))(1 + e^{-x_{i,j}})^{-1} + \lambda w_i \tag{3}$$

Apart from that, we must calculate model performance in order to avoid over-fitting or under-fitting. Thus, metrics such training and test accuracy will be used in our work.

$$Accuracy = \frac{100}{p} \sum_{j=1}^{p} \delta(y(x_j, w^*) - y_j) \tag{4}$$

2.3 Mathematical Optimization Algorithm

Not only is it necessary to have some objective function, but its optimal values as well. This is the reason why we consider **First Order Derivative Methods** because they are designed to converge to an optimum solution. In fact, we will be working with three different algorithms:

- Gradient Method
- Conjugate Gradient Method [1]
- Broyden–Fletcher–Goldfarb–Shanno Method

All this methods require the same attributes. Firstly, user must provide some initial point, we always decided to use a $\vec{0}$ vector to represent the inactivity of all weights at first attempt, and the loss functions with its gradient. Secondly, some stopping criteria must be decided beforehand, such as at which value of the gradient's norm should the algorithm stop or which is the maximum number of iterations. We fixed both of them at $\epsilon = 10^{-6}$ and $kmax = 1000$ respectively. Finally, it will be needed to decide some features that **Backtracking Line Search** works with. Backtracking Line Search is a line search method to determine the maximum amount to move along a given search direction [2].In our case, we will be working with descent directions since we are searching a minimum. This method needs the maximum number of iterations, as well as the maximum step length, which is modified at each iteration automatically. Then, two values following this rule: $0 < c_1 < c_2 < 1$ must be given. This last values could be better assigned with further knowledge on FODM's convergence.

[1] Conjugate Gradient Method will be used with Polak-Ribière^{+} and restart condition : $\frac{{\nabla f^k}^t \nabla f^{k-1}}{||\nabla f^k||^2} \geq 0.1$

3 Experiments and Results

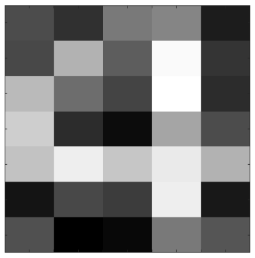
In this section, we show how algorithms performed in different situations, analysing behavior and convergence metrics.

First experiment consists of predicting number 4 and 8 using all methods previously presented. In this experiment, all methods were capable of predicting number 4 with 100% accuracy. In fact, they were able to minimize loss function. Nevertheless, these results did not hold when predicting number 8. Although these methods were trained to predict 8 with many more sample, they were not able to achieve the same results as in number 4. In this case, Gradient method did not find a stationary point since its $||g^*|| \not\approx 0.0$. Furthermore, any method was effective enough to make loss function close to 0, which implies that training accuracy will not be 100%.

Target	trFreq	noFreq	trSamples	teSamples	trSeed	teSeed	Method	λ	ϵ	k_{max}	α_{max}	k_{BLS}	ϵ_{BLS}	c1	c2	$\|\|g^*\|\|$	L^*	iter	trAcc	teAcc
4	0.5	0.1	10	100	123	1234	G	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$2e^{-3}$	$3e^{-3}$	10^3	100	100
8	0.5	0.1	500	5000	123	1234	G	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	1	13.84	10^3	97.4	94.4
4	0.5	0.1	10	100	123	1234	CG	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.45	$1.4e^{-3}$	$2.9e^{-3}$	10^3	100	100
8	0.5	0.1	500	5000	123	1234	CG	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.45	$3.9e^{-2}$	4.26	10^3	99.2	94.9
4	0.5	0.1	10	100	123	1234	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$7.3e^{-7}$	$9.7e^{-8}$	27	100	100
8	0.5	0.1	500	5000	123	1234	BFGS	0	10^{-6}	10^3	2	30	10^{-3}	0.1	0.9	$9.8e^{-7}$	4.0	46	99.2	93.6

Table 1: Experiment conditions and Results

This significant difference, between predicting both numbers, may be because its pixels distribution. This can lead to a confusion for the Neural Network, because training a number that differ from the rest is easier than if it is similar to them. This is the case of number 4, which some pixels makes it unique while some others are irrelevant to detect such number. Thus, its neural weights must have a non-homogeneous distribution. Instead, number 8 seems to be more similar to other numbers, hence, each pixel must be representative, leading to a more homogeneous weights distribution.


(a) Signature of the target 4

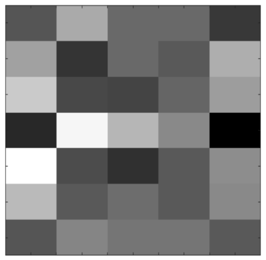

(b) Signature of the target 8

Figure 3: Gradient Method & Training Data = 500

In terms of global convergence, it seems that each method found a stationary point despite Gradient with target 8. Although Gradient method did not achieve great results with target 8, it was slightly better than BFGS predicting unseen data for such target. This algorithm, BFGS, seems to be the best regarding $||g^*||$ and L^*. Nevertheless, any method is good enough to perform a pattern detection with respect to 4.

Another metric considered is local convergence. We realise that BFGS was capable of searching for the right descent direction, hence, its iterations were the lowest by far. Once the other methods were close to the stationary point, they did not move enough to break stopping criteria $||g^*|| \leq \epsilon$. If we check descent direction $-d_k \nabla L$, it will be close to 0 since no moving is appreciate. Hence, CG and G are trapped in a zone nearby stationary point in 200 iterations more or less. One possible solution to reduce iterations could be applying $\lambda \neq 0.0$. This technique helps to find the optimum value faster due to penalization, however, it may present worse accuracy.

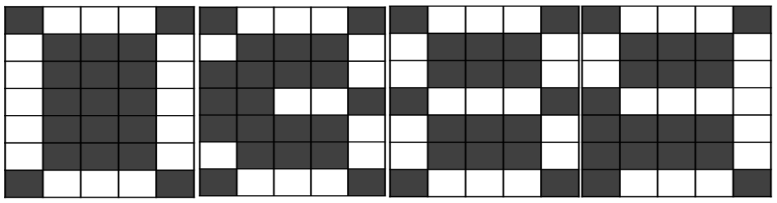
Second experiment consists of predicting each number, individually, using only BFGS method. We realise that some numbers were easier to detect than others. Consequently, we decided to classify numbers in two sets, depending on its test accuracy. We thought that test accuracy may be linked to ease of detection. For instance, the number set $C_1 = \{1, 2, 4, 5, 6, 7\}$ achieved around 97% test accuracy, whereas the number set $C_2 = \{0, 3, 8, 9\}$ attained around 90% test accuracy. Although C_1 was trained with less data than $C_2 - \{9\}$, it performed much better with unobserved samples. Surprisingly number 9 did not improve its accuracy even its training data increase, otherwise it tend to over-fit data.

Target	trFreq	noFreq	trSamples	teSamples	trSeed	teSeed	Method	λ	ϵ	k_{max}	α_{max}	k_{BLS}	ϵ_{BLS}	$c1$	$c2$	$\|\|g^*\|\|$	L^*	iter	trAcc	teAcc
0	0.5	0.2	500	5000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$9.6e^{-7}$	17.0	57	96.6	88.2
1	0.5	0.2	100	1000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$7.0e^{-7}$	$1.6e^{-7}$	28	100	98.8
2	0.5	0.2	100	1000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$9.3e^{-7}$	$1.3e^{-7}$	31	100	96.3
3	0.5	0.2	500	5000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$5.3e^{-7}$	7.0	53	98.6	91.9
4	0.5	0.2	100	1000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$6.8e^{-7}$	$1.0e^{-7}$	27	100	99.5
5	0.5	0.2	100	1000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$9.0e^{-7}$	$8.2e^{-7}$	25	100	97.5
6	0.5	0.2	100	1000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$7.7e^{-7}$	$1.0e^{-7}$	29	100	97.5
7	0.5	0.2	100	1000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$9.6e^{-7}$	$1.4e^{-7}$	30	100	97.6
8	0.5	0.2	500	5000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$5.3e^{-7}$	21.0	60	95.8	87.8
9	0.5	0.2	100	1000	65	12	BFGS	0	10^{-6}	10^3	1	30	10^{-3}	0.1	0.9	$7.8e^{-7}$	$1.1e^{-7}$	30	100	92.5

Table 2: Experiment conditions and Results

Experiment results showed that BFGS found a stationary point per each target. Moreover, iterations were no greater than 60, which means that this algorithm is robust. It can be seen that $L^*_{C_1} \simeq 0.0$. This results in very high training accuracy and it should be easy to perform well in test data. This is not the case of number 9 that seems to perform well in training but not in validation. It can be seen that $L^*_{C_2-\{9\}} \not\simeq 0.0$, hence, its training accuracy is not as good as C_1. This may be linked to the fact that it is more complicate to predict C_2 numbers. Not only is it hard to predict test data from C_2, but also training one.

This notable difference between C_1 and C_2 may be due to numbers similarities. It is very complicate to differentiate numbers in C_2, because they only differ at most in 6 pixels. Moreover, if we add 20% of noise frequency it is really hard to distinguish those numbers, as it can be seen in Figure 4 at detection phase. On the other hand, numbers in C_1 seem to have some unique traces, which ease its prediction.


(a) 0 (b) 3 (c) 8 (d) 9

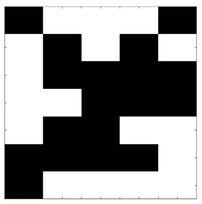

(e) 0 : 20% Noise

Figure 4: Detection phase

Thus, we could state that training Neural Networks to detect numbers in C_1 was easier than numbers in C_2.

References

[1] F.-Javier Heredia. Unconstrained optimization. University Lecture, 2019.

[2] Wikipedia contributors. Backtracking line search. `https://en.wikipedia.org/wiki/Backtracking_line_search`, 2019. [Online; accessed 20-April-2019].