Pattern Recognition with Single Layer Neural Network

Oriol Domingo Roig, Roger Creus Castanyer

April 2019

1 Introduction

In this work we aim to classify set of numbers using a **Data Mining** technique, such as **Neural Networks**. 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 links 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.

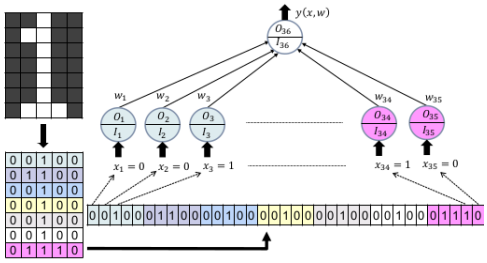


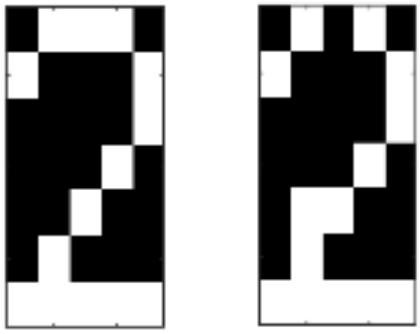
Figure 1: Training Model [1]

2 Materials and Methods

This project is divided into three components which ease the understanding of global workflow. Firstly, we develop 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 report 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 a file *"dataSets"* which requires two seeds, number of samples in the training data, target number data set, frequency of target in data, 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 vector of Boolean values representing whether such sample belongs to the target number set or not. Finally, 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 a 0 or 1 value, representing each one a neuron. Then, some weights are applied to each neuron and link 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.

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

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

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.

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

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.

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

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 their main goal are stationary points. In fact, we will be working with three different algorithms:

- Gradient Method
- Conjugate Gradient Method
- 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 a descents directions since we are searching a minimum. This method needs the maximum number of iterations, as well as the maximum step length that it is modified at each iteration automatically, and then two values following this rule: $0 < c_1 < c_2 < 1$. This last values could be better assigned with further knowledge on FODM's convergence.

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.

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}	1000	1	30	10^{-3}	0.1	0.9	0.002	0.003	1000	100	100
8	0.5	0.1	500	5000	123	1234	G	0	10^{-6}	1000	1	30	10^{-3}	0.1	0.9	1.02	13.84	1000	97.4	94.4
4	0.5	0.1	10	100	123	1234	CG	0	10^{-6}	1000	1	30	10^{-3}	0.1	0.45	0.001	0.003	1000	100	100
8	0.5	0.1	500	5000	123	1234	CG	0	10^{-6}	1000	1	30	10^{-3}	0.1	0.45	0.064	4.83	1000	99.2	94.8
4	0.5	0.1	10	100	123	1234	BFGS	0	10^{-6}	1000	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}	1000	2	30	10^{-3}	0.1	0.9	$9.8e^{-7}$	4.0	46	99.2	93.6

Table 1: Experiment conditions and Results

Second experiment consists of predicting each number, individually, using only BFGS method.

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}	1000	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}	1000	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}	1000	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}	1000	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}	1000	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}	1000	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}	1000	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}	1000	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}	1000	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}	1000	1	30	10^{-3}	0.1	0.9	$7.8e^{-7}$	$1.1e^{-7}$	30	100	92.5

Table 2: Experiment conditions and Results

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].