American Journal of Physics and Applications
2014; 2(4): 88-94
Published online August 10, 2014 (http://www.sciencepublishinggroup.com/j/ajpa)
doi: 10.11648/j.ajpa.20140204.11
ISSN: 2330-4286 (Print); ISSN: 2330-4308 (Online)

Optimizing back-propagation gradient for classification by an artificial neural network

Said El Yamani, Samir Zeriouh, Mustapha Boutahri, Ahmed Roukhe

Optronic and Information Treatment Team, Atomic, Mechanical, Photonic and Energy Laboratory, Faculty of Science, Moulay Ismail University, B. P. 11201 Zitoune, Meknès, Morocco

Email address:
said.elyamani@yahoo.fr (S. E. Yamani)

To cite this article:
Said El Yamani, Samir Zeriouh, Mustapha Boutahri, Ahmed Roukhe. Optimizing Back-Propagation Gradient for Classification by an Artificial Neural Network. *American Journal of Physics and Applications.* Vol. 2, No. 4, 2014, pp. 88-94.
doi: 10.11648/j.ajpa.20140204.11

Abstract: In a complex and changing a remote sensing system, which requires taking quick and informed decisions environment, connectionist methods have shown their great contribution in particular the reduction and classification of spectral data. In this context, this paper proposes to study the parameters that optimize the results of an artificial neural network ANN multilayer perceptron based, for classification of chemical agents on multi-spectral images. The mean squared error cost function remains one of the major parameters of the network convergence at its learning phase and a challenge that will face our approach to improve the gradient descent by the conjugate gradient method that seems fast and efficient.

Keywords: Optimizing, Artificial Neural Networks, Classification, Identification, Conjugate Gradient, Multi-Layer Perceptron, Back Propagation of the Gradient

1. Introduction

Recent years have been marked by an extraordinary emergence of Artificial Neural Networks (ANNs) that were found very suitable for solving complex and unstructured classifications. The strength of these actually lies in their ability to adapt thanks to a learning phase through which they develop and establish relationships between variable from a dataset. In the field of remote sensing, ANN proved of great interest among researchers especially in case of non-linear phenomena.

Nowadays, several threats proliferate and the most important remain chemical, biological or nuclear. These threats can occur in several scenarios and require immediate action. It should be noted that, facing such an eventuality, a large number of key elements turns essentially necessary to determine the particular nature or agents, their rates, their propagation speeds and especially the surfaces they could occupy.

This involves the deployment of resources and identification of reliable and efficient classification. In this context the ANN appear to be a highly suitable tool [1].

Their diverse neuronal architectures allow justifying the choice of the type of neural networks to use for each case that may arise. In addition, during emergency cases, many optimization methods are proposed to reduce processing time and improve network performance, considering these constraints and characteristics related to the use of the type of network selected.

Thus, the work of this paper focuses on this subject to the optimization of the back propagation learning in the network.

2. Multi-Layer Perceptron Network

2.1. MLP Network Architecture

The model used is a Multi-layer Perception based (MLP) network, comprising a P layer of n inputs, the hidden layers and a Y layer of m outputs [2]. Adjacent layers are fully connected (Fig. 1).

The input layer comprises a number of cells equal to the dimension of the sample vector.

$$P = \begin{bmatrix} P_{11} & P_{12} & . & . & P_{1r} \\ P_{21} & P_{22} & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ P_{s1} & . & . & . & P_{sr} \end{bmatrix}$$

With r = 18, number of values in each sample.
s =30, number of samples.

The number of cells in each hidden layer may vary depending on the behavior of the network. However, the number of cells of the output layer corresponds to the dimension of projection which is 3.

Activation functions adopted in this ANN model are sigmoid and linear for the output function.

Diagram 1 shows a simplified diagram of the network.

2.2. Learning Algorithm

2.2.1. Back Propagation of Gradient

The proposed algorithm is the back propagation gradient that helps learning the multi-layer network characteristics by updating and adjusting the weights of different layers to minimize a cost function [3].

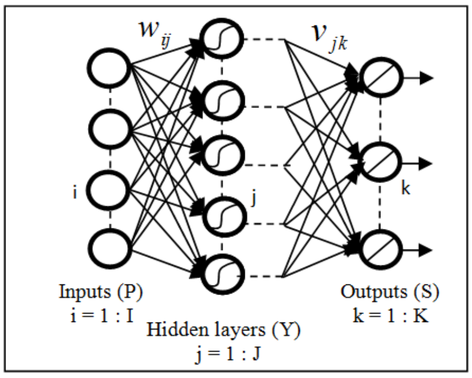
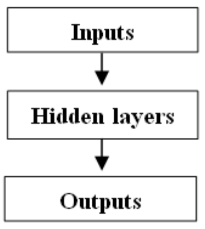


Figure 1. MLP network architecture.



x_i

$$y_j = f\left(\sum_i x_i w_{ij}\right) = f(b_j) \tag{1}$$

$$s_k = f\left(\sum_j y_j v_{jk}\right) = f(a_k) \tag{2}$$

Diag 1. Simplified model of MLP network architecture.

Where b_j is the total input of the output layer s_k and a_k is the total input of the hidden layer y_j.

It goes through three phases: A first one that allows forward propagation of the cells state. The second step is a back propagation process that minimizes the error, and the third step is an adjustment of the weights connection [4].

Mathematical expressions of the different phases of the algorithm are expressed by the following equations.

First, the cost function "E" must be defined:

$$E\left(v_{jk}\right) = \frac{1}{2}\sum_k \left(s_{d_k} - s_k\right)^2 \tag{3}$$

Where, S_{dk} is the desired output; S_k is the effective

To minimize E, we calculate its gradient with respect to the weights v, then we change the weight in the opposite direction of the gradient. In other words, we update the weights of the gradient of the highest slope.

$$\Delta v_{jk} = -\eta_1 \frac{\partial E\left(v_{jk}\right)}{\partial v_{jk}} \tag{4}$$

$$v_{jk}^{n} = v_{jk}^{n+1} - \Delta v_{jk}^{n} \tag{5}$$

Where, η (0< η <1) is the gradient step, used to control the fast convergence of the algorithm.

For a two-layer network:

$$s_k = f\left(\sum_j y_j v_{jk}\right) = f\left(\sum_j f\left(\sum_i x_i w_{ij}\right) v_{jk}\right) \tag{6}$$

After partial derivations, respectively to v_{hl} and w_{hl}, we obtain:

$$\frac{\partial E\left(v_{jk}\right)}{\partial v_{hl}} = -\sum_k \left(sd_k - s_k\right)\frac{\partial s_k}{\partial v_{hl}} \tag{7}$$

$$\frac{\partial s_k}{\partial v_{hk}} = \frac{\partial f\left(a_k\right)}{\partial v_{hk}} = \frac{\partial f\left(a_k\right)}{\partial a_k}\cdot\frac{\partial a_k}{\partial v_{hk}}$$
$$= \frac{\partial f\left(a_k\right)}{\partial a_k}\cdot\frac{\partial\left(\sum_j y_j v_{jk}\right)}{\partial v_{hk}} = f'\left(\sum_j y_j v_{jk}\right) y_h \tag{8}$$

$$\frac{\partial s_k}{\partial w_{hl}} = \frac{\partial f\left(a_k\right)}{\partial w_{hl}} = \frac{\partial f\left(a_k\right)}{\partial a_k}\cdot\frac{\partial a_k}{\partial w_{hl}}$$
$$= \frac{\partial f\left(a_k\right)}{\partial a_k}\cdot\frac{\partial\left(\sum_j f(b_j) v_{jk}\right)}{\partial b_j}\cdot\frac{\partial b_j}{\partial w_{hl}}$$
$$= \frac{\partial f\left(\sum_j y_j v_{jk}\right)}{\partial\left(\sum_j y_j v_{jk}\right)}\cdot\frac{\partial\left(\sum_j f\left(\sum_i x_i w_{ij}\right) v_{jk}\right)}{\partial\left(\sum_i x_i w_{ij}\right)}\cdot\frac{\partial\left(\sum_i x_i w_{il}\right)}{\partial w_{hl}} \tag{9}$$
$$= f'\left(\sum_j y_j v_{jk}\right) f'\left(\sum_i x_i w_{il}\right) x_h$$

Substituting the derivatives in the weights updating equations, we get:

$$\Delta v_{hl} = \eta_1 \sum_l \left(sd_l - s_l\right) f'\left(\sum_j y_j w_{jl}\right) y_h \tag{10}$$

$$\Delta w_{hl} = \eta_2 f'\left(\sum_j y_j w_{jl}\right) f'\left(\sum_i x_i w_{il}\right) x_h \tag{11}$$

2.2.2. Back Propagation of Gradient (Method of Newton)

According to the Taylor expansion of n-th order v^n (nth iteration), E may be approximated using the following equation [5]:

$$E\left(v_{jk}^{n+1}\right) = E\left(v_{jk}^{n}\right) + \sum_{m} \frac{\partial E\left(v_{jk}^{n}\right)}{\partial v^{m}} \cdot \Delta\left(v_{jk}^{n}\right) + \frac{1}{2}\left[\Delta\left(v_{jk}^{n}\right)\right]^{T} \sum_{m}\sum_{p} \frac{\partial^{2} E\left(v_{jk}^{n}\right)}{\partial v^{m} \partial v^{p}} \cdot \Delta\left(v_{jk}^{n}\right) \tag{12}$$

Where $\Delta\left(v_{jk}^{n}\right) = v_{jk}^{n+1} - v_{jk}^{n}$

To simplify,

$$E\left(v_{jk}^{n+1}\right) = E\left(v_{jk}^{n}\right) + E'\left(v_{jk}^{n}\right) \cdot \Delta\left(v_{jk}^{n}\right) + \frac{1}{2}\left[\Delta\left(v_{jk}^{n}\right)\right]^{T} \cdot E''\left(v_{jk}^{n}\right)\Delta\left(v_{jk}^{n}\right) \tag{13}$$

Where the E' matrix composed of partial derivatives of 1st order is a jacobian matrix:

$$G^{n} = E'\left(v_{jk}^{n}\right) = \sum_{m} \frac{\partial E\left(v_{jk}^{n}\right)}{\partial v_{jk}^{m}} \tag{14}$$

And E" is a matrix composed of second order partial derivatives, which is a Hessian symmetric positive definite matrix:

$$H^{n} = E''\left(v_{jk}^{n}\right) = \sum_{m}\sum_{p} \frac{\partial^{2} E\left(v_{jk}^{n}\right)}{\partial v_{jk}^{m} \partial v_{jk}^{p}} \tag{15}$$

Deriving from (13)

$$E'\left(v_{jk}^{n+1}\right) = E'\left(v_{jk}^{n}\right) + E''\left(v_{jk}^{n}\right)\Delta\left(v_{jk}^{n}\right) \tag{16}$$

We assume E' and E" are continuous and exist for values of higher order derivatives, very negligible for this approximation to be minimized.

From (16), the weights updating can be expressed as:

$$\Delta\left(v_{jk}^{n}\right) = -E'\left(v_{jk}^{n}\right) \cdot \left[E''\left(v_{jk}^{n}\right)\right]^{-1} \tag{17}$$

$$v_{jk}^{n+1} = v_{jk}^{n} - \left[E'\left(v_{jk}^{n}\right)\right] \cdot \left[E''\left(v_{jk}^{n}\right)\right]^{-1} \tag{18}$$

$$v_{jk}^{n+1} = v_{jk}^{n} - \left[G^{n}\right] \cdot \left[H^{m}\right]^{-1} \tag{19}$$

From (17), the gradient can be expressed as:

$$G^{n} = -H^{n} \cdot \left(v^{n+1} - v^{n}\right) \tag{20}$$

To improve this method, we try to make G^{n+1} orthogonal to the line of the previous value of G^{n}. So, we pose:

$$\left(G^{n+1}\right)^{T} G^{n} = 0 \tag{21}$$

After weight adjustments according to (5)

$$v^{n+1} - v^{n} = -\eta^{n} G^{n} \tag{22}$$

And after substitution of (16)

$$G^{n+1} = G^{n} - \eta^{n} H^{n} G^{n} \tag{23}$$

Finally, multiplying this result by $(G^{n})^{T}$ leads to:

$$0 = \left(G^{n}\right)^{T} G^{n} - \eta^{n}\left(G^{n}\right)^{T} H^{n} G^{n} \tag{24}$$

Hence, from this last one, we obtain

$$\eta^{n} = \frac{\left(G^{n}\right)^{T} G^{n}}{\left(G^{n}\right)^{T} H^{n} G^{n}} \tag{25}$$

Note that H^{n} are constant:

$$\text{For } n=0 \qquad G^{1} = G^{0} + H^{0} \cdot \left(v^{1} - v^{0}\right) \tag{26}$$

Where the minimal error is obtained for $G^{1}=0$. That means:

$$v^{1} - v^{0} = -\left(H^{0}\right)^{-1} G^{0} \tag{27}$$

This corresponds to the results of the Newton's method, which is for updating weights:

$$v^{n} - v^{0} = -\sum_{n} G^{n}\left(H^{n}\right)^{-1} \tag{28}$$

Thus, the weights are updated in the direction of negative gradient following form:

$$v^{n} - v^{0} = -\alpha^{n} G^{n} \tag{29}$$

However the major disadvantage of this method is the calculation of the storage and the inversion of the Hessian matrix during iterations. This is why the conjugate gradient method, a quasi-Newton method likely to reduce these limitations, reveals more efficient.

2.2.3. Methods of Improving Back-Propagation Gradient

2.2.3.1. Method of BGFS (Quasi-Newton)

This algorithm is the Broyden-Fletcher-Goldfarb-Shanno. The weight update is made according to the equation [6]:

$$\Delta v^{n} = -\left(H^{n}\right)^{T} G^{n} \tag{30}$$

2.2.3.2. Method of Levenberg-Marquardt (LM)

This is a similar method to the BFGS algorithm with $\left(H^n\right)^T = \left(\nabla E\right)^T \cdot \left(\nabla E\right)$ et $G = \left(\nabla G\right)^T \cdot \varepsilon$ where ∇G is the jacobian matrix of 1st order, and ε is the errors vector [7].

2.2.3.3. Method of Gradient Descent with Momentum

To prevent learning phases causes oscillations, the weight difference comprises varying the previous weight by adding a kinetic energy term [8].

$$\Delta v^n = -\alpha^n G^n + \rho \Delta v^{n-1} \tag{31}$$

2.2.3.4. Method of the Conjugate Gradient

The conjugate gradient method is similar to the descent of the steepest approach, but includes an inertial term which is calculated during iterations steps to search for the conjugate of the directions, the update process has been described by Fletcher -Reeves of the Polak-Ribiere [9].

According to Fletcher-Reeves, we operate on it, firstly by looking for the most important downward slope, and then by combining its direction with the previous one.

$$-G^0 = g^0 \quad , \quad \Delta v_{jk}^n = -\alpha^n g^n$$

$$g^n = -G^n + \beta^n g^{n-1} \quad , \beta^n = \frac{\left(G^n\right)^T + G^n}{\left(G^{n-1}\right)^T + G^{n-1}} \tag{32}$$

Or $\beta^n = \dfrac{\Delta\left(G^{n-1}\right)^T + G^n}{\left(G^{n-1}\right)^T + G^{n-1}}$ Polak-Ribiere according

3. Application in Chemicals Agents Identification and Classification

3.1. Classification Using Multi-Layer Perceptron

In order to implement the followed theoretical process, a program has been written using Matlab and applies for a database with sample of the three agents (Table 1).

Table 1. *Three agents gas database.*

H2S-NO2 mixture																			
Ech01	=	0.01	0.13	0.05	0.03	0.03	0.02	0.02	0.27	0.08	0.05	0.03	0.04	0.00	0.00	0.00	0.00	0.01	0.00;
Ech02	=	0.01	0.12	0.04	0.03	0.03	0.02	0.01	0.31	0.08	0.05	0.03	0.04	0.00	0.01	0.00	0.00	0.00	0.00;
Ech03	=	0.01	0.12	0.04	0.02	0.03	0.02	0.02	0.19	0.07	0.04	0.03	0.03	0.00	0.01	0.00	0.00	0.00	0.00;
Ech04	=	0.01	0.12	0.04	0.03	0.03	0.02	0.10	0.31	0.09	0.07	0.04	0.05	0.00	0.01	0.01	0.01	0.00	0.00;
Ech05	=	0.01	0.15	0.06	0.04	0.03	0.02	0.02	0.21	0.07	0.05	0.03	0.04	0.00	0.00	0.00	0.00	0.00	0.00;
Ech06	=	0.01	0.14	0.05	0.03	0.03	0.02	0.02	0.26	0.08	0.05	0.03	0.04	0.00	0.01	0.00	0.00	0.00	0.00;
Ech07	=	0.01	0.14	0.05	0.03	0.03	0.02	0.01	0.29	0.08	0.05	0.03	0.03	0.00	0.01	0.00	0.01	0.01	0.00;
Ech08	=	0.01	0.12	0.05	0.02	0.03	0.02	0.01	0.30	0.08	0.06	0.03	0.04	0.00	0.01	0.00	0.00	0.00	0.00;
Ech09	=	0.01	0.13	0.04	0.02	0.02	0.02	0.02	0.22	0.07	0.04	0.03	0.03	0.00	0.01	0.00	0.00	0.00	0.00;
Ech10	=	0.01	0.12	0.05	0.03	0.03	0.02	0.10	0.25	0.09	0.06	0.04	0.05	0.00	0.01	0.01	0.01	0.01	0.00;
Gaz H2S																			
Ech11	=	0.01	0.22	0.05	0.04	0.06	0.02	0.02	0.24	0.07	0.05	0.06	0.03	0.00	0.01	0.00	0.00	0.00	0.00;
Ech12	=	0.01	0.18	0.04	0.04	0.05	0.02	0.01	0.23	0.07	0.05	0.05	0.03	0.00	0.00	0.00	0.00	0.00	0.00;
Ech13	=	0.01	0.17	0.03	0.03	0.05	0.02	0.05	0.27	0.09	0.07	0.06	0.04	0.00	0.01	0.00	0.00	0.00	0.00;
Ech14	=	0.01	0.17	0.04	0.04	0.05	0.02	0.11	0.37	0.11	0.09	0.06	0.06	0.00	0.02	0.01	0.01	0.00	0.00;
Ech15	=	0.02	0.17	0.04	0.04	0.05	0.02	0.17	0.44	0.11	0.10	0.07	0.07	0.01	0.02	0.01	0.01	0.00	0.00;
Ech16	=	0.02	0.17	0.05	0.04	0.05	0.02	0.24	0.55	0.16	0.15	0.10	0.09	0.04	0.07	0.02	0.02	0.01	0.02;
Ech17	=	0.09	0.19	0.04	0.05	0.05	0.02	0.12	0.30	0.07	0.07	0.05	0.04	0.00	0.01	0.00	0.00	0.00	0.00;
Ech18	=	0.08	0.16	0.03	0.04	0.04	0.02	0.16	0.40	0.09	0.09	0.06	0.05	0.01	0.01	0.00	0.00	0.00	0.00;
Ech19	=	0.08	0.15	0.03	0.04	0.04	0.02	0.28	0.67	0.14	0.16	0.08	0.08	0.01	0.01	0.01	0.01	0.00	0.00;
Ech20	=	0.09	0.16	0.04	0.04	0.04	0.02	0.47	0.86	0.17	0.20	0.09	0.10	0.01	0.03	0.01	0.01	0.00	0.00;
Gaz NO2																			
Ech21	=	0.01	0.21	0.04	0.04	0.05	0.02	0.01	0.21	0.05	0.05	0.05	0.02	0.00	0.04	0.00	0.00	0.01	0.01;
Ech22	=	0.01	0.19	0.04	0.04	0.04	0.02	0.01	0.19	0.04	0.04	0.04	0.02	0.00	0.01	0.00	0.01	0.01	0.00;
Ech23	=	0.01	0.18	0.03	0.03	0.04	0.02	0.01	0.18	0.03	0.03	0.04	0.02	0.00	0.00	0.00	0.00	0.00	0.00;
Ech24	=	0.01	0.16	0.03	0.03	0.04	0.01	0.00	0.16	0.03	0.03	0.04	0.01	0.00	0.00	0.00	0.00	0.00	0.00;
Ech25	=	0.01	0.21	0.04	0.04	0.05	0.02	0.01	0.21	0.05	0.05	0.05	0.02	0.00	0.04	0.00	0.00	0.00	0.01;
Ech26	=	0.01	0.19	0.04	0.05	0.04	0.02	0.00	0.18	0.04	0.04	0.04	0.02	0.00	0.00	0.00	0.00	0.00	0.00;
Ech27	=	0.01	0.17	0.04	0.04	0.04	0.02	0.00	0.17	0.03	0.04	0.04	0.02	0.00	0.00	0.00	0.00	0.00	0.00;
Ech28	=	0.01	0.16	0.03	0.04	0.04	0.02	0.00	0.14	0.02	0.02	0.04	0.01	0.00	0.00	0.00	0.00	0.00	0.00;
Ech29	=	0.00	0.16	0.03	0.03	0.04	0.02	0.00	0.11	0.01	0.01	0.04	0.01	0.00	0.00	0.00	0.00	0.00	0.00;
Ech30	=	0.01	0.15	0.03	0.03	0.04	0.02	0.00	0.10	0.01	0.01	0.04	0.01	0.00	0.00	0.00	0.00	0.00	0.00;

The followed methodology during this computing phase in Matlab is based on fundamental steps, especially the creation of the network where

- The Samples matrix that contains the input values of the network are r × s = 18 × 30 size;
- The matrix with the desired results. Its size is 3× 30 with the following vectors [-1 -1 +1]’, [-1 +1 -1]’ and [+1 -1 -1]’ as expected result per type of gas.

Several networks models are tested using Mean Square Error criterion [10] (Fig. 4).

The outputs of the networks at each iteration are equivalent to the expected values in the samples where the total learning path length is associated to a gradient and E (Fig. 5).

The initial weights have values near zero.

3.2. Matlab Simulations

We conducted various simulations in Matlab for testing and the results are plotted as follow:

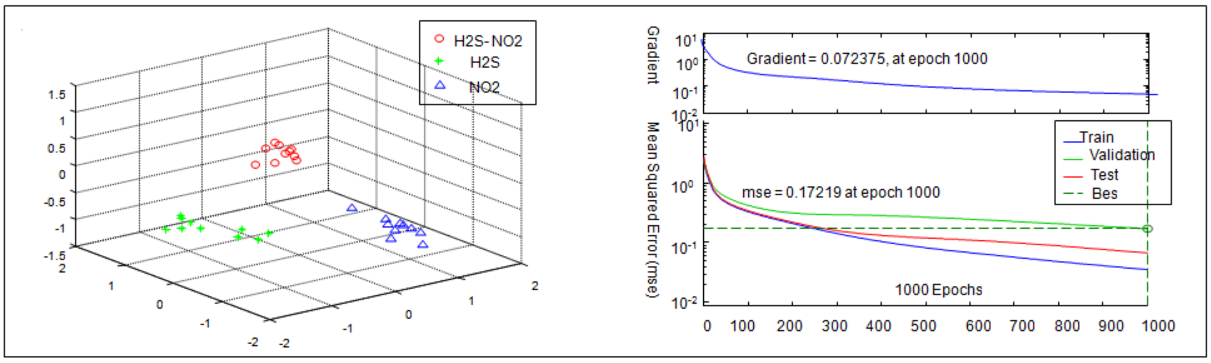


Figure 2. *3D classification of the three agents and performances in learning with Gradient Descent (GD) (traingd)*

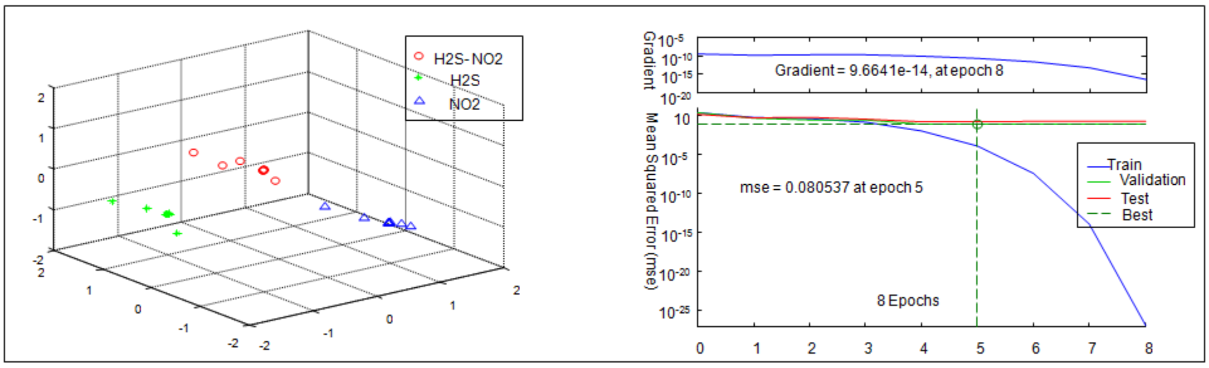


Figure 3. *3D classification of the three agents and performances in learning with Levenberg-Marquardt (LM) (trainlm)*

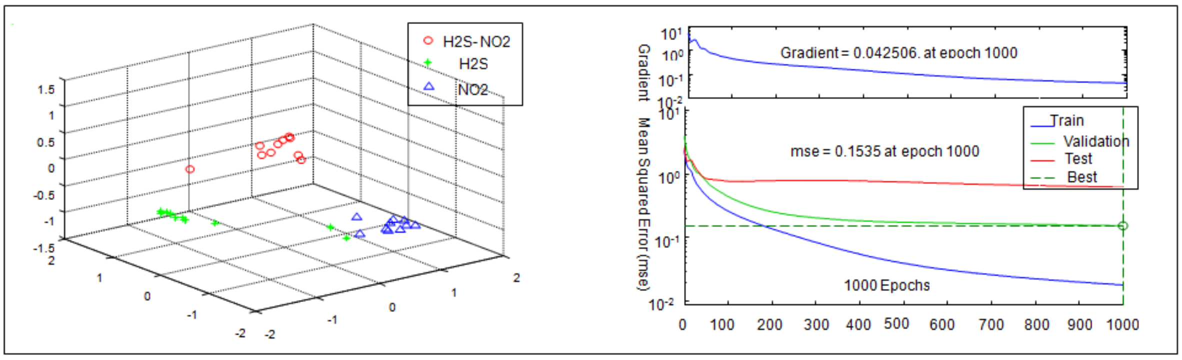


Figure 4. *3D classification of the three agents and performances in learning with GD with Momentum (GDM) (traingdm)*

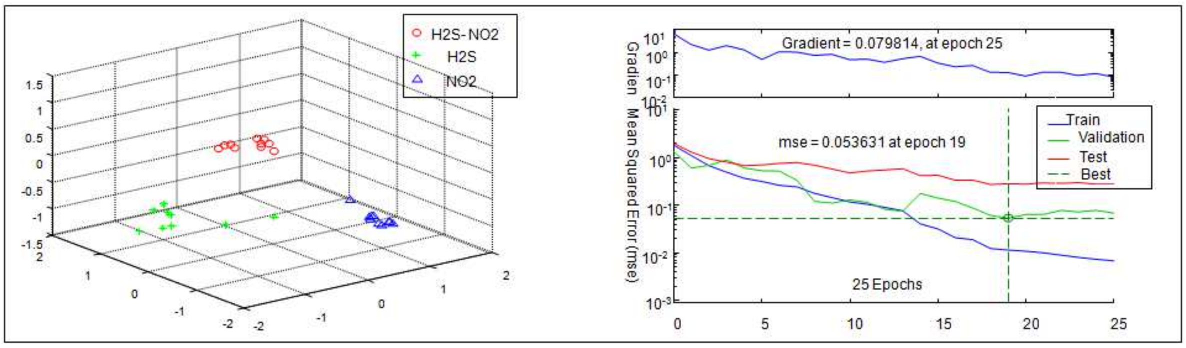


Figure 5. *3D classification of the three agents and performances in learning with Conjugate Gradient Descent (trainscg)*

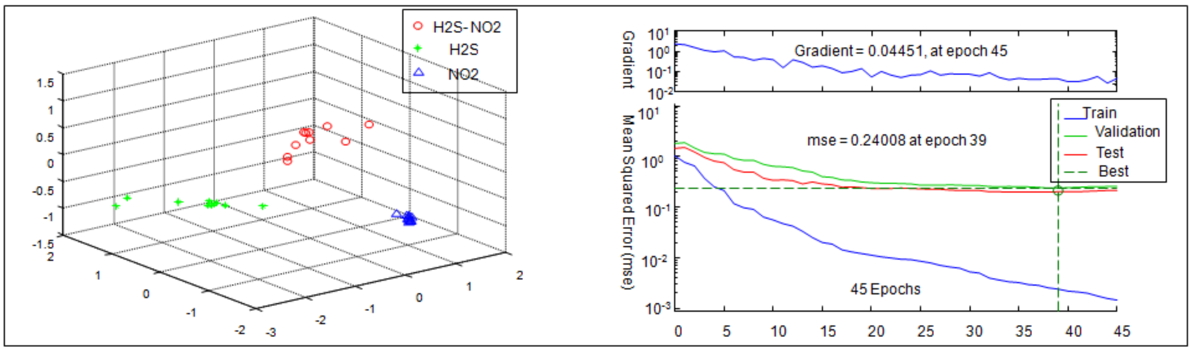


Figure 6. *3D classification of the three agents and performances in learning with CGD Fletcher-Reeves (traincgf)*

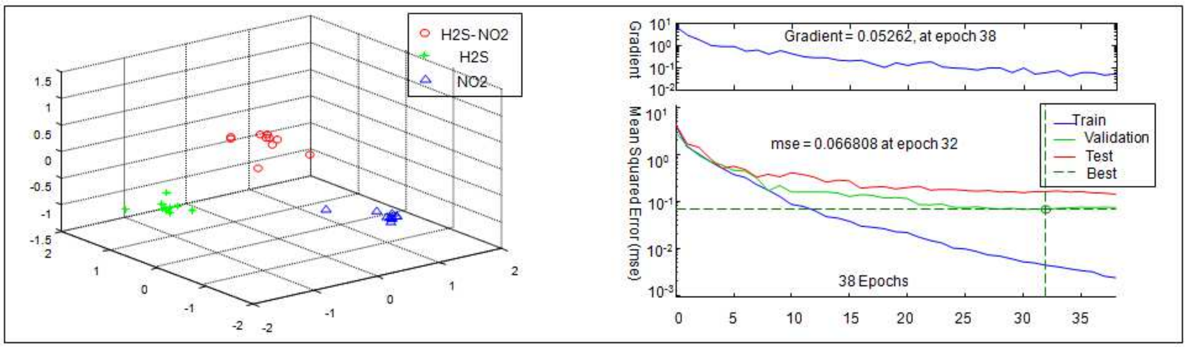


Figure 7. *3D classification of the three agents and performances in learning with CGD Polak-Ribière (traincgp)*

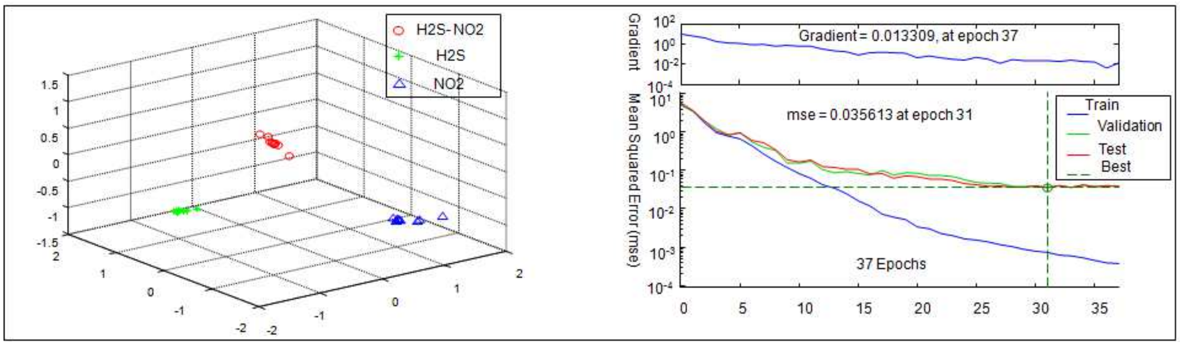


Figure 8. *3D classification of the three agents and performances in learning with Quasi-Newton (BFGS) (trainbfg)*

Table 2. *Comparison of parameters of the different back-propagation networks*

Number of neurons for a hidden layer	Epochs	Train time (s)	Performances	Gradient	Mu
Gradient Descent (GD) (traingd)	1000	07	0.0354	0.0724	-
Levenberg-Marquardt (LM) (trainlm)	08	01	1.77e-27	9.66e-14	1.00e-11
GD with Momentum (GDM) (traingdm)	1000	04	0.0181	0.0425	-
Conjugate Gradient Descent (trainscg)	25	01	0.00680	0.0798	-
CGD Fletcher-Reeves (traincgf)	45	01	0.00145	0.0445	-
CGD Polak-Ribière (traincgp)	38	01	0.00237	0.0526	-
Quasi-Newton (BFGS) (trainbfg)	37	31	0.000372	0.0133	-

3.3. Results Analysis

The simulation results obtained show that, the back-propagation of the gradient, which operates an iterative approximation along the line of the steepest slope, could be improved by the conjugate gradient algorithm with the search for a minimum which produces a faster convergence, and could also be improved by the Quasi-Newton method. However, in this case, this remains less efficient than the Levenberg-Marquardt. Compared to other back-propagation algorithms, the latter is much more effective in view of its high speed and high precision, considering the network does not reach saturation.

4. Conclusions

In this article, learning methods presented dealt with the back-propagation of the gradient for a neural network multilayer perceptron based, in order to establish a classification of chemical agents. This classification is applied to a database which test statistic is the mean square error to be minimized to threshold value. From this study, it appears that multiple solutions are able to optimize the performance of learning, thereby creating special interest in functional classification.

So if back propagation algorithms adjust the weights in the direction of greatest slope, direction in which the performance function decreases more rapidly, it is still possible that convergence can be made faster with the conjugate gradient algorithm or the Quasi-Newton method for fast optimization algorithm or Levenberg-Marquardt. The latter was designed for second order training approaches without having to calculate the Hessian matrix. Thanks to the implementation done in MATLAB, this algorithm seems to be very effective in classification problems of moderate size of this kind, in spite of its very high calculations at iterations.

References

[1] M. JANATI IDRISS and All, Reducing the number of channels of multi-spectral images by connectionist approach, Signal Processing 17, 2000, pp 491-500.

[2] A. Guerin, J.H. Crasy, Reconfigurable computing architecture for simulating networks of neurons, Review Signal Processing 5 (3), 1988, pp 178-186.

[3] J. Proriol, MLP: Network program multi-layer neurons, Journal of Modulad, 1996, pp 24-28.

[4] S. Lahmiri, A comparative study of back-propagation algorithms in financial prediction, International Journal of Computer Science, Engineering and Applications (IJCSEA), Vol.1, No.4, 2011, pp 15-21.

[5] L. N. M. Tawfiq Improving Gradient Descent Method for Training Feed Forward Neural Networks, International Journal of Modern Computer Science & Engineering, 2(1), 2013, pp 12-24.

[6] M. F. MEILLER, A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning, Neural Networks, Vol.6, 1993, pp. 525-533.

[7] M. Wilamowski, and Hao Yu Improved Computation for Levenberg–Marquardt Training, IEEE transactions on neural networks, vol. 21, no. 6, 2010, pp 930-93.

[8] N. Qian1, On the momentum term in gradient descent learning algorithms, Neural Networks, 1999, pp 145-151.

[9] R. S. Ransing and N. M. Nawi1, An improved conjugate gradient based learning algorithm for back propagation neural networks, World Academy of Science, Engineering and Technology, Vol 2, 2008, pp 06-26.

[10] E.P. van Someren, L.F.A. Wessels, E. Backer, M.J.T. Reinders, Multi-criterion optimization for genetic network modeling, Signal Processing 83, 2003, pp 763-775.