Available online at http://scik.org

Commun. Math. Biol. Neurosci. 2022, 2022:37

https://doi.org/10.28919/cmbn/7355

ISSN: 2052-2541

A COMPARATIVE STUDY OF BACK-PROPAGATION ALGORITHMS: LEVENBERG-MARQUART AND BFGS FOR THE FORMATION OF MULTILAYER NEURAL NETWORKS FOR ESTIMATION OF FLUORIDE

RACHID EL CHAAL*, MOULAY OTHMAN ABOUTAFAIL

Engineering Sciences Laboratory, Data Analysis, Mathematical Modeling, and Optimization Team,

Ibn Tofail University National School of Applied Sciences ENSA, Kenitra 14 000 Morocco

Abstract. This paper compares and contrasts two back-propagation algorithms: the Levenberg-Marquardt (LM) and the Broyden Fletcher Goldfarb Shanno (BFGS). The concentrations of sixteen physicochemical factors were used to predict Fluoride in the Inaouène basin using artificial neural networks (ANN) of the multilayer perceptron type (MLP). We created many models based on the evolution of activation functions and the number of neurons in the hidden layer. The mean square error (MSE) and correlation coefficient were used to assess the effectiveness of the various ANN model training procedures (R). The LM training algorithms outperform the BFGS training algorithm, according to the results. The statistical indicators (R = 0.99 and MSE = 0.135 for LM and R = 0.95 and MSE = 41.22 for BFGS) highlight the efficacy of the LM algorithm for Fluoride prediction when compared to the BFGS method utilizing MLP type neural networks.

Keywords: back-propagation algorithm; multilayer perceptron; prediction; optimization; statistical indicators.

2010 AMS Subject Classification: 92B20.

*Corresponding author

E-mail address: rachid.elchaal@uit.ac.ma

Received March 16, 2022

1. INTRODUCTION

The most popular learning algorithm is Back-propagation (BP) [1], and it is the most common and widely used supervised training algorithm for solving approximation problems, recognition of shapes, classifying and discovering patterns, and making predictions from data, and other well-known issues. Based on statistics, data mining, pattern recognition, and predictive analyzes. The BP algorithm(BPA) is the most widely used example of supervised learning because of the media coverage of some spectacular applications, such as the demonstration of Sejnowski and Rosenberg (1987) and Adamson and Damper(1996), in which BPA is used in a system that learns to pronounce a text in English [2], [3]. Another success was the prediction of stock market prices [4] and, the Comparative study of different artificial neural network (ANN) training algorithms for atmospheric temperature forecasting in Tabuk, Saudi Arabia [5] and, more recently, a study on cumulative hazards evaluation for the water environment [6].

The gradient BP technique is a method that calculates the error gradient for each Neuron in the network, from the last layer to the first. The publication history shows that BPA has been discovered independently by different authors but under different names. The principle of BP can be described in three basic steps: routing information through the network; BP of sensitivities and calculation of the gradient; and adjust the parameters by the approximate gradient rule. It is important to note that BPA suffers from the inherent limitations of the gradient technique because of the risk of being trapped in a local minimum. If the gradients or their derivatives are zero, the network is trapped in a local minimum. Add to this the slowness of convergence, especially when dealing with large networks [7] (i.e., for which the number of connection weights to be determined is essential). To make the optimization more efficient, we can use second-order methods such as the so-called Quasi-Newton or modified Newton methods.

2. SECOND-DEGREE OPTIMIZATION METHOD (QUASI-NEWTONIAN METHODS)

2.1. Levenberg-Marquardt Back-Propagation Method (LM):

The LM algorithm is a variation of Newton's method [8], which was designed for minimizing functions that are sums of squares of non-linear functions [9], [10]. This is ideal for training

artificial neural networks (ANN). this is known to be very efficient when applied to ANN [11], [12], where the root mean square error is the performance index. Newton's; update for optimizing a performance index F(x) is $x_{k+1} = x_k - A_k^{-1} g_k$ where $A_k \equiv \nabla^2 F(x)\big|_{x=x_k}$ and $g_k \equiv \nabla F(x)\big|_{x=x_k}$

supposing that F(X) is a sum of the square function

$$F(x) = \sum_{i=1}^{N} v^T(x) v(x) \tag{1}$$

therefore, the j th element of the gradient is

$$[\nabla F(\mathbf{x})]_j = \frac{\partial F(\mathbf{x})}{\partial x_j} = 2 \sum_{i=1}^{N} v_i(\mathbf{x}) \frac{\partial v_i(\mathbf{x})}{\partial x_j} \tag{2}$$

The gradient is written in matrix form

$$\nabla F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{v}(\mathbf{x}) \tag{3}$$

where[13]

$$\begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{4}$$

it is the Jacobian matrix. to determine the Hesse matrix, the element k, j of this matrix would be

$$[\nabla^2 F(\mathbf{x})]_{k,j} = \frac{\partial^2 F(\mathbf{x})}{\partial x_k \partial x_j} = 2 \sum_{i=1}^{N} \left\{ \frac{\partial v_i(\mathbf{x})}{\partial x_k} \frac{\partial v_i(\mathbf{x})}{\partial x_j} + v_i(\mathbf{x}) \frac{\partial^2 v_i(\mathbf{x})}{\partial x_k \partial x_j} \right\} \tag{5}$$

The Hessian matrix can then be expressed in matrix form

$$\nabla^2 F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + 2\mathbf{S}(\mathbf{x}) \tag{6}$$

Where:

$$\mathbf{S}(\mathbf{x}) = \sum_{i=1}^{N} v_i(\mathbf{x}) \nabla^2 v_i(\mathbf{x}) \tag{7}$$

If **S(x)** it is assumed to be minor, the Hessian matrix can be approximated as

$$\nabla^2 F(\mathbf{x}) \cong 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) \tag{8}$$

Substituting (8) and (3) into $x_{k+1} = x_k - A_k^{-1} g_k$, the Gauss-Newton method is obtained[14]

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k - [2\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1}2\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) \\ &= \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)\end{aligned} \tag{9}$$

The Gauss-Newton approach has the advantage of not requiring the computation of second-order derivatives, as does Newton's conventional approach. The Gauss-Newton approach has the disadvantage of requiring the matrix: $H = J^T J$. It could be that it isn't invertible. To overcome this, make the following adjustments to the predicted Hessian matrix: $G = H + \mu I$. In order to invert this matrix, we first assumed that the eigenvalues and the eigenvectors of **H** are respectively the following:$\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ and $\{z_1, z_2, \dots, z_n\}$.

Then

$$\mathbf{G}_i = [\mathbf{H} + \mu\mathbf{I}]\mathbf{z} = \mathbf{H}\mathbf{u} + \mu\mathbf{z}_i = \lambda\mathbf{z}_i + \mu\mathbf{z}_i = (\lambda_i + \mu)\mathbf{z}_i \tag{10}$$

As a result, the eigenvectors of G and H are the same., and the eigenvalues of G are $(\lambda_j + \mu) \cdot G$ can be made definite in a positive way by raising μ until $(\lambda_j + \mu) \succ 0$ for all is,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) \tag{11}$$

Or

$$\Delta\mathbf{x}_k = -[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) \tag{12}$$

This algorithm has the benefit of approaching the steepest descent algorithm with a low learning rate as μ_k is increased.

$$\mathbf{x}_{k+1} \cong \mathbf{x}_k - \frac{1}{\mu_k}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) = \mathbf{x}_k \frac{1}{2\mu_k}\nabla F(\mathbf{x}) \tag{13}$$

The algorithm becomes Gauss-Newton when μ_k is high and when μ_k is reduced to zero. μ_k is set to a tiny number (e.g., $\mu_k = 0.01$) before the start of the process. If a step does not result in a lower $\mathbf{F}(\mathbf{x})$ value, it is repeated with μ_k multiplied by some factor $\vartheta \succ 1 (eg \cdot \vartheta = 10)$. As a little step in the direction of the sharpest decline is taken, $\mathbf{F}(\mathbf{x})$ should decrease with time. If the step yields a smaller value for $\mathbf{F}(\mathbf{x})$, the next step divides μ_k by ϑ, bringing the algorithm closer to Gauss-Newton and allowing for faster convergence.

2.2. Back-propagation method Broyden-Fletcher-Goldfarb-Shanno (BFGS):

Newton's technique uses a quadratic approximation of the function **F(x)** rather than a linear approximation. At a point where the quadratic function is minimized, the following approximate

solution is obtained:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) = F(\mathbf{x}_k) + \mathbf{g}_k^T\mathbf{A}\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k\mathbf{A}_k\Delta\mathbf{x}_k \quad (14)$$

The sequence obtained is $x_{k+1} = x_k - A_k^{-1}g_k$

The principal advance of Newton's approach is that it converges at a quadratic rate. The fastest descent method, on the other hand, has a much more slowly linear convergency rate. But Newton's method requires a significant amount of computation at each step. Suppose the problem has N dimensions an $O(N^*)$To calculate the research direction d^k, a floating-point operation is required. The quasi-Newton approach computes the search direction by using an approximation [15] Hessian matrix. Let H_k be a N*N symmetric matrix that approximates the Hessian matrix A_k; then, by minimizing the quadratic function, the search direction for the quasi-Newton method is obtained:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) = F(\mathbf{x}_k) + \mathbf{g}_k^T\Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k^T\mathbf{H}_k\Delta\mathbf{x}_k \quad (15)$$

If H_k is invertible, the quadratic program's solution yields the following descent direction:

$$\mathrm{d}^k = \mathbf{x} - \mathbf{x}_k = -(\mathbf{H}_k^{-1})\nabla F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}_1} \quad (16)$$

H_k as the matrix, must be updated from iteration to iteration by including the most recent gradient information in order to estimate the Hessian of the function **F(x)** at $x = x_k$

The matrix H_k is updated using the equation below [16], [17]:

$$H_{k+1} = H_k + \frac{y_k y_k^\top}{y_k^\top s_k} - \frac{H_k s_k s_k^\top H_k}{s_k^\top H_k s_k} \quad (17)$$

Where

$$s_k = x_{k+1} - x_k; y_k = g_{k+1} - g_k \quad (18)$$

2.3. Collection of Data:

TAZA is a city located in the northeast of Morocco in the Taza corridor, a pass where the Rif and Middle Atlas Mountains meet. Apart from the "corridor" formed by the valley of the Wadi Inaouen and the plain of Guercif, the rest of the province is dominated by mountains. Indeed, the province occupies the area which connects the Rif to the Middle Atlas, the two mountain ranges

narrowing at the level of the Touaher pass (559 m above sea level). The Inaouen is a Moroccan river, that forms near the city of TAZA, by the confluence of the Boulejraf and Larbaâ wadis, and borrows from east to west the breach of Taza, which marks the limit between the Rif and the Middle Atlas.

Our database contains 100 surface water samples (notes) collected between 2014 and 2015 in the governorate of Taza (the Inaouen watershed). The water samples are being gathered, shipped and stocked following the protocol established by the National Potable Water Office. Part of the survey was carried out on site (temperature, dissolved oxygen, etc.). The rest was carried out at the Regional Centre CURI in Fez, which is supported by the USMBA (Sidi Mohamed Ben Abdellah University)[18].

2.4 Reduction and Preprocessing of Data:

2.4.1 Inputs Selection:

Use The 16 physico-chemical properties measured in the samples are the explanatory variables (independent):

pH, Temperature: (T°C), dissolved oxygen: (DO), Magnesium: (Mg), Bicarbonate: (HCO3), Chlorides: (Cl), total dissolved solids: (TDS), Total Alkalinity: (CaCO3), Sodium: (Na), Potassium: (K), Calcium: (Ca), Sulfates: (SO4), Phosphorus: (P), Nitrate: (NO3), conductivity: (Cond) and Ammoniacal nitrogen: (NH4),

Fluoride (F) is the dependent variable (predicted).

The database is distributed as follows: For the training phase of a predictive pattern of the dependent variable, 70% of the samples were randomly selected from the entire data set. The other 30% of the data have been used to check the performance of the network and to avoid overlearning. This is to see if the predictions of these models are accurate and valid (15% for testing and 15% for validity).

2.4.2 Formatting of Data

Standardization is a preprocessing approach for data that aids in the reduction of model complexity. Raw, untransformed values comprise the input data (16 independent variables). Their

magnitudes are vastly different. These data are turned into standardized variables to help standardize the measurement scales. In fact, according to the relationship, the values of each independent variable I have been standardized with respect to their means and standard deviations:$X(v_i)$ [19][20][21]

$$\mathrm{X_s}(v_i) = \left(\frac{\mathrm{X}(v_i) - \bar{X}(v_i)}{\sigma(v_i)}\right) \qquad i \in \{1.....16\} \tag{19}$$

With:

$\mathrm{X_s}(v_i)$: Standardized value

$\mathrm{X}(v_i)$: Observed value

$\bar{X}(v_i)$: Average value

$$\bar{X}(v_i) = \frac{1}{100}\sum_{k=1}^{100} X_k(v_i) \tag{20}$$

$\sigma(v_i)$: standard deviation

$$\sigma(v_i) = \sqrt{\frac{1}{100}\sum_{k=1}^{100}(X_k(v_i) - \bar{X}(v_i))^2} \tag{21}$$

The goal of normalising the values of all parameters is to eliminate very high or low-level exponential values and to prevent the variation from increasing too much with the mean.

To meet the needs of the neural network transfer function, the values for the dependent variables were normalized in the interval [0;1]. This normalization was done in accordance with the relationship:

$$\mathbf{Y}_\mathrm{n} = \left(\frac{\mathrm{Y} - \mathrm{Y_{min}}}{\mathrm{Y_{max}} - \mathrm{Y_{min}}}\right) \tag{22}$$

With:

$\mathrm{Y_n}$: Normalized value

Y : Original value

$\mathrm{Y_{min}}$: Minimum value

$\mathrm{Y_{max}}$: Maximum value

2.4.3 Neural Network Implementation with MATLAB:

Simple single-layer networks are substantially less effective than multilayer perceptron (MLP) networks. There are three layers in the network employed in this study: an input layer, a hidden

layer, and an output layer. The hidden layer's number of neurons isn't predetermined. This is decided during the learning process. The MATLAB 2018 software was used to run the neural network simulations.

The design of multilayer neural networks consists of two parts: establishing the network's architecture and performing numerical optimization calculations. This computation involves determining synaptic coefficients and using a supervised learning technique to update these coefficients. The LM and BFGS algorithms were chosen for our research. These two techniques use non-linear optimization approaches to reduce the cost function (the mean squared error (MSE)), which is a measure of the difference between the network's actual and desired responses. This optimization is done iteratively by altering the weights as a function of the cost function's gradient: the gradient is computed using the BP method, which is a specialized method for neural networks. Algorithm optimization makes advantage of it. Before learning, the weights are set at random.

2.4.4 Evaluation of Performances:

MATLAB 2018 uses the cost function, which is most commonly used in statistics and is called the least-squares criterion, and consists of minimizing the sum of the squares of the residuals, in this case, the network will learn a discriminant function, for evaluating the quality of our predictive model and judging these performances. The total of the discrepancies between the target values and the predicted outputs provided for the training set gives the mean squared error (MSE). The evaluation's outcome can be expressed in two ways: statistical indicators and graph analysis. The correlation coefficient (R) and mean squared error (MSE) are the indicators employed in this study, and they are defined as follows: The correlation coefficient:

$$\mathbf{R} = \frac{\sum_{i=1}^{100}\left(Y_i^{\text{obs}} - \overline{Y^{\text{obs}}}\right)\left(Y_i^{\text{predi}} - \overline{Y^{\text{predi}}}\right)}{\sqrt{\sum_{i=1}^{100}\left(Y_i^{\text{obs}} - \overline{Y^{\text{obs}}}\right)^2\left(Y_i^{\text{predi}} - \overline{Y^{\text{predi}}}\right)^2}} \tag{23}$$

The mean squared error (MSE)

$$\text{MSE} = \frac{1}{100}\sum_{i=1}^{100}\left(Y_i^{\text{obs}} - Y_i^{\text{predi}}\right)^2 \tag{24}$$

With Y_i^{obs} is the observed (actual) value of the studied metal, Y_i^{predi} is the estimated value of the metal by the model at observation i, $\bar{Y}$ is the mean value. The best prediction is when $|R|$ on the one hand and SSE, on the other hand, tends towards 1 and 0, respectively.

3. RESULTS AND DISCUSSION

Tests have demonstrated that changing the network's design, the number of hidden layers, the number of hidden neurons, and the length of training cycles can increase the performance of a model built with MLP (Multilayer Perceptron) type neural networks (number of iterations). We did this by gradually increasing the number of hidden neurons (NHN = 1, 2, 3, ..., 15). We used two learning algorithms in this study, referred to as high-performance algorithms LM and BFGS. We varied the number of neurons in the hidden layer and the pairs of transfer functions for each learning technique. The mean square error (MSE) and the correlation coefficient were used to evaluate performance (R). The methods are implemented and developed using the MATLAB 2018 platform on a computer. The computer's processor is an Intel Core i5-7200U CPU 2.50GHz with 4 GB RAM.

3.1 Training ANN with LM Algorithm

Table 1 represents the best performance found for the different combinations of transfer function pairs for the LM algorithm; they converge quickly and result in low values of the mean square error MSE and high values of the correlation coefficient R in a time of no more than a few seconds.

Hidden layer	Output layer	R	MSE	Architecture	Number of iterations
Tansig	Tansig	0,980	33.96	[16-3-1]	11
Tansig	Logsig	0,994	26.14	[16-4-1]	15
Tansig	Purelin	0,910	95.12	[16-5-1]	9
Logsig	Logsig	0,911	92.31	[16-6-1]	12
Logsig	Purelin	0,970	41.4	[16-7-1]	18
Logsig	Purelin	0,999	0.135	[16-8-1]	**22**

Table1: Recap of the best architectures offered by Matlab for prediction fluoride with algorithm LM.

From this table, we note that:

* The [16-8-1] architecture, with a Logsig function for the input layer and a Purelin function for the output layer, gave the best performance for the LM algorithm: R=0.99 and MSE=0.135.

* The LM algorithm converges with the minimum number of iterations (22iterations) for all combinations of transfer functions; this algorithm is reputed to be very efficient in the approximation of functions, mainly when the network contains less than a hundredweight to be updated, which is the case here.

After 22 cycles, the network had hit overtraining; it would be worthwhile to keep on learning until this point for the test to minimize the gradient and improve our model (Figure.3). The following are the different values of training parameters found in this investigation, as shown in Figures 2; 3; and 4:

- Max. numbers of iteration (Epochs) = 22
- Determination coefficients = 0.99
- Root mean squared error (MSE) = 0.135
- Rate of learning (Mu) = 0.0001
- Minimal gradient = 0.875

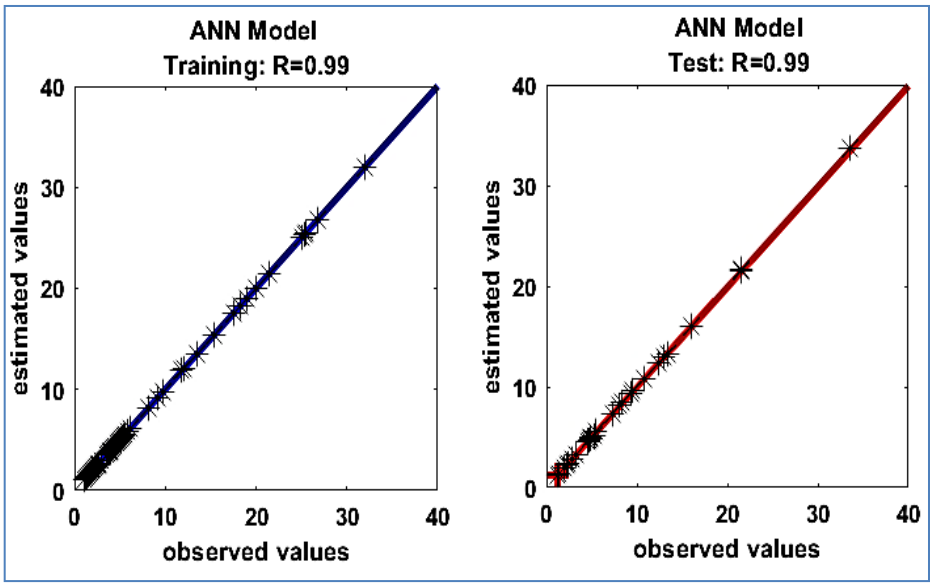

Figure 1: Trend line showing the relationship between the observed values and values estimated by the MLP model with algorithm ML for fluoride for the training and test phase

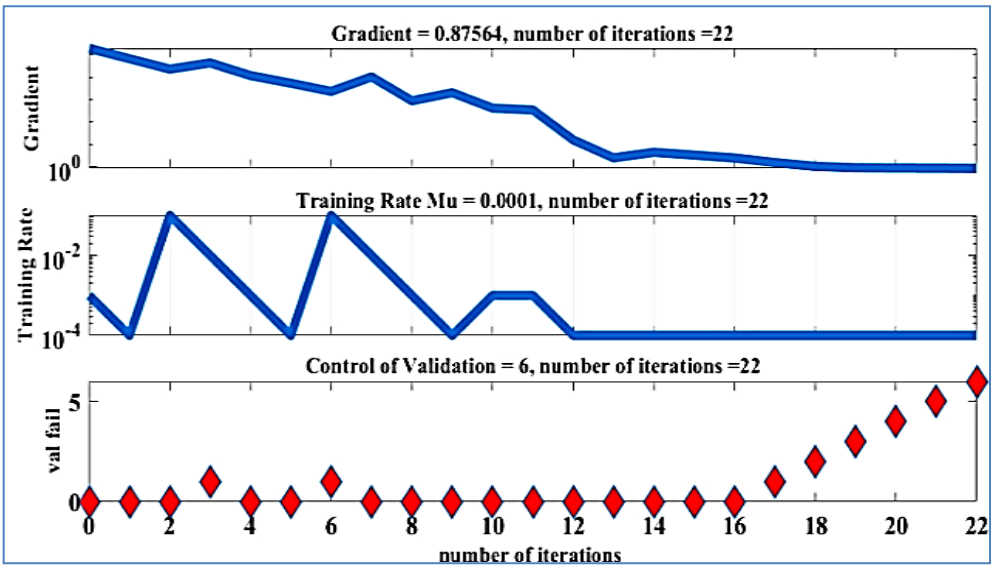

Figure 2: As a function of the number of iterations, the error gradient, learning rate, and validation error (for fluoride) evolve.

Network training is depicted in Figure 3. It demonstrates that the desired outcome is reached after sixteen iterations. The three curves relating to the evolution of the mean square error of the three phases converge appropriately to the least mean square error with eight hidden neurons (MSE)

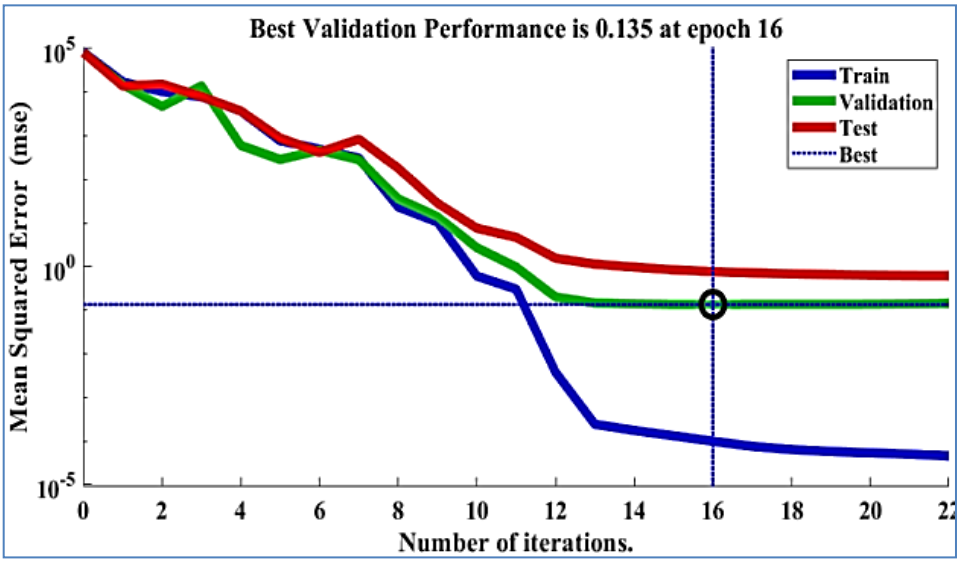

Figure 3: the representative graph concerning the development of the mean square error for a network architecture [16-8-1].

3.2 Training ANN with BFGS Algorithm:

Table 2 represents the best architecture found for the different combinations of transfer function pairs for the BFGS algorithm.

From this table, we note that:

- The [16-6-1] architecture, with a Logsig function for the input layer and a Purelin function for the output layer, gave the best performance for the Broyden-Fletcher-Goldfarb-Shanno algorithm and her indicator statistical (R=0.95 and MSE=41.22)

Hidden layer	Output layer	R	MSE	Architecture	Number of iterations
Tansig	Tansig	0,920	82.16	[16-3-1]	180
Tansig	Logsig	0,916	100.85	[16-4-1]	215
Tansig	Purelin	0,901	111.12	[16-5-1]	149
Logsig	*Logsig*	*0,953*	*41.22*	*[16-6-1]*	*222*
Logsig	Purelin	0,932	63.12	[16-7-1]	482
Logsig	Purelin	0,945	48.45	[16-8-1]	237

Table2: Recap of the best architectures offered by Matlab for prediction Fluoride with algorithm BFGS.

The network reached overtraining after 222 iterations; it would be worthwhile to keep training up to this point for testing to minimize the gradient and improve our model (Figure 6). Figures 5, 6 and 7 illustrate the different values of the training parameters found in this experiment:

- Max. numbers of iteration (Epochs) = 222
- Determination coefficients = 0.95
- Root mean squared error (MSE) = 41.22
- Minimal gradient = 66.96

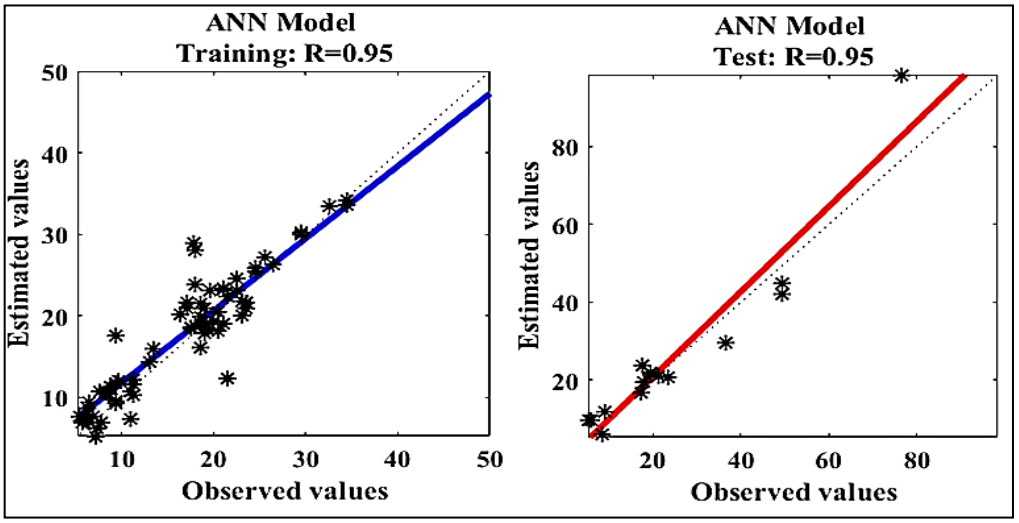

Figure 4: Trend line showing the relationship between the observed values and values estimated by the MLP model with algorithm BFGS for fluoride for the training and test phase.

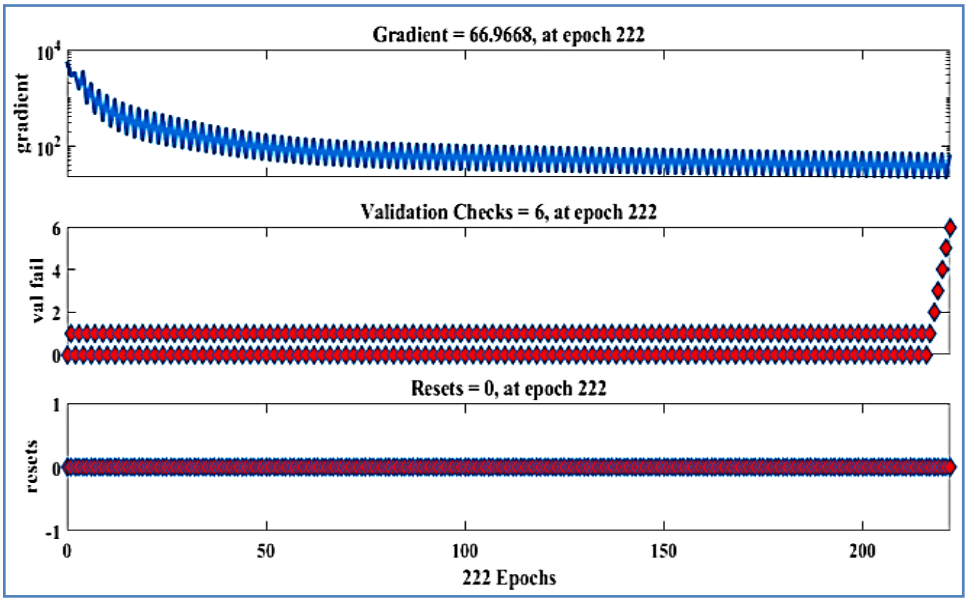

Figure 5: As a function of the number of iterations, the error gradient, learning rate, and validation error (for fluoride) evolve.

Network training is depicted in Figure 6. It demonstrates that the required result is obtained after 216 iterations. The three curves corresponding to the evolution of the mean square error of the three phases converge appropriately to the least mean square error (MSE=41.28) when six hidden neurons are used.

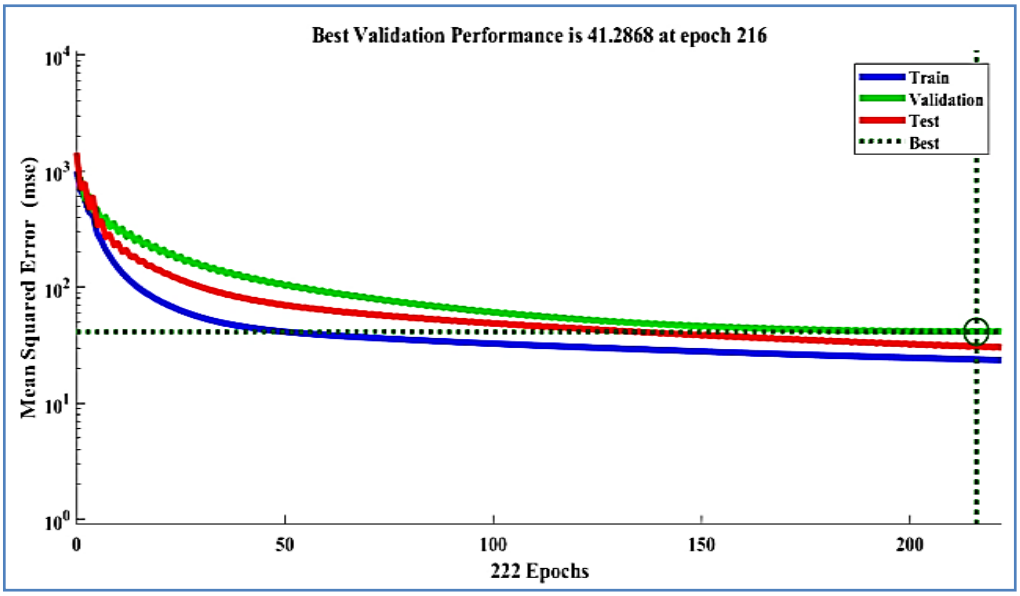

Figure 6: the representative graph concerning the development of the mean square error for a network architecture [16-6-1].

Figures 3 and 6 give the mean square error (MSE) values for LM algorithm of ANN. The LM algorithms showed a lowest MSE value at the point of convergence. However, the BFGS algorithm took more epochs (216) to converge to the smallest MSE (41.22). compared to the LM algorithms training. Nevertheless, the LM algorithm took only 16 epochs to reach 0.135 MSE.

4. CONCLUSION

Artificial neural networks are effective prediction tools. They are capable of dealing with non-linear issues. They do, however, have a considerable disadvantage in terms of network architecture selection, as this is generally left to the user. In this study, we constructed numerous high-performing models based on the two learning algorithms LM and BFGS. The findings reveal that the LM method has the best statistical indicators (R=0.99 and MSE=0.135) as well as the fastest convergence speed (22 iterations). Indeed, when compared to the BFGS algorithm, the model created by the ML algorithm allows for improvements of up to 4% in the explanation of

variation. The LM training algorithms outperform the BFGS training algorithm, according to the results.

As a result, the analytical findings show that the LM method is more efficient than the BFGS for Fluoride prediction in the Inaouen basin. The BFGS algorithm, on the other hand, can be considered a best substitute approach.

In light of this, the following recommendations can help us continue to develop this topic:

- Use additional activation functions and work on various sorts of networks, such as recurring networks. The obtained results inspire us to reflect on the method that allows us to better the work we've done so far. Using other algorithms, for example, would be quite fascinating.
- Furthermore, the models are based on real-world data. As a result, they can be used to forecast Fluoride concentrations in the future based on physicochemical factors.
- Try out a different RNA architecture to see which one gives you the best results.

CONFLICT OF INTERESTS

The author(s) declare that there is no conflict of interests.

REFERENCES

[1] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature. 323 (1986), 533–536. https://doi.org/10.1038/323533a0.

[2] T.J. Sejnowski, C.R. Rosenberg, Parallel systems that learn to pronounce English text, Complex Syst. 1 (1987), 145–168.

[3] M.J. Adamson, R.I. Damper, A recurrent network that learns to pronounce English text, in: Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP ’96, IEEE, Philadelphia, PA, USA, 1996: pp. 1704–1707. https://doi.org/10.1109/ICSLP.1996.607955..

[4] A.N. Refenes, M. Azema-Barac, Neural network applications in financial asset management, Neural Comput. Appl. 2 (1994), 13–39. https://doi.org/10.1007/bf01423096.

[5] P. Anushka, A. Hazi Md., R. Upaka, Comparison of different artificial neural network (ANN) training algorithms

to predict the atmospheric temperature in Tabuk, Saudi Arabia, Mausam. 71 (2021), 233–244. https://doi.org/10.54302/mausam.v71i2.22.

[6] E. Shi, Y. Shang, Y. Li, M. Zhang, A cumulative-risk assessment method based on an artificial neural network model for the water environment, Environ. Sci. Pollut. Res. 28 (2021), 46176–46185. https://doi.org/10.1007/s11356-021-12540-6.

[7] R.S. Govindaraju, A.R. Rao, eds., Artificial neural networks in hydrology, Springer Netherlands, 2000. https://doi.org/10.1007/978-94-015-9341-0.

[8] S. Basterrech, S. Mohammed, G. Rubino, M. Soliman, Levenberg-Marquardt training algorithms for random neural networks, Computer J. 54 (2009), 125–135. https://doi.org/10.1093/comjnl/bxp101.

[9] C.M. Bishop, Pattern recognition and machine learning, Springer, New York, 2006.

[10] G. Dreyfus, Neural networks methodology and applications, Springer, Berlin, 2005.

[11] M.T. Hagan, M.B. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Trans. Neural Netw. 5 (1994), 989–993. https://doi.org/10.1109/72.329697.

[12] N. Ampazis, S.J. Perantonis, Levenberg-Marquardt algorithm with adaptive momentum for the efficient training of feedforward networks, in: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, IEEE, Como, Italy, 2000: pp. 126–131 vol.1. https://doi.org/10.1109/IJCNN.2000.857825.

[13] Y. Chen, S. Zhang, Research on EEG classification with neural networks based on the Levenberg-Marquardt algorithm, in: C. Liu, L. Wang, A. Yang (Eds.), Information Computing and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012: pp. 195–202. https://doi.org/10.1007/978-3-642-34041-3_29.

[14] L. E. Scales, Introduction to non-linear optimization, Macmillan Publishers, London, 1985.

[15] D. Goldfarb, A family of variable-metric methods derived by variational means, Math. Comp. 24 (1970), 23–26. https://doi.org/10.1090/S0025-5718-1970-0258249-6..

[16] R. Battiti, F. Masulli, BFGS optimization for faster and automated supervised learning, in: International Neural Network Conference, Springer Netherlands, Dordrecht, 1990: pp. 757–760. https://doi.org/10.1007/978-94-009-0643-3_68.

[17] R. Battiti, First- and second-order methods for learning: Between steepest descent and Newton's method, Neural

Comput. 4 (1992), 141–166. https://doi.org/10.1162/neco.1992.4.2.141.

[18] R. El Chaal, M.O. Aboutafail, Development of stochastic mathematical models for the prediction of heavy metal content in surface waters using artificial neural network and multiple linear regression, E3S Web Conf. 314 (2021), 02001. https://doi.org/10.1051/e3sconf/202131402001.

[19] A. Abdallaoui, H. El Badaoui, Comparative study of two stochastic models using the physicochemical characteristics of river sediment to predict the concentration of toxic metals, J. Mater. Environ. Sci. 6 (2015), 445–454.

[20] S.G.K. Patro, K.K. Sahu, Normalization: A preprocessing stage, Int. Adv. Res. J. Sci. Eng. Technol. 2 (2015), 20–22. https://doi.org/10.17148/IARJSET.2015.2305.

[21] Z. Bayatzadeh Fard, F. Ghadimi, H. Fattahi, Use of artificial intelligence techniques to predict distribution of heavy metals in groundwater of Lakan lead-zinc mine in Iran, J. Mining Environ. 8 (2017), 35-48.