Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Quasi-Newton methods for training neural networks

B. Robitaille, B. Marcos, M. Veillette & G. Payre
Chemical Engineering Department, Université de Sherbrooke, Sherbrooke, Québec, Canada, J1K 2R1

ABSTRACT

The backpropagation algorithm is the most popular procedure to train self-learning feedforward neural networks. However, the rate of convergence of this algorithm is slow because the backpropagation algorithm is mainly a steepest descent method. Several researchers have proposed other approaches to improve the rate of convergence: conjugate gradient methods, dynamic modification of learning parameters, full quasi-Newton or Newton methods, stochastic methods, etc. Quasi-Newton methods were criticized because they require significant computation time and memory space to perform the update of the hessian matrix. This paper proposes a modification to the classical approach of the quasi-Newton method that takes into account the structure of the network. With this modification, the size of the problem is not proportional to the total number of weights but depends on the number of neurons of each level. The modified quasi-Newton method is tested on two examples and is compared to classical approaches. The numerical results show that this approach represents a clear gain in terms of computational time without increasing the requirement of memory space.

INTRODUCTION

Self-learning feedforward neural nets require a learning process to map an output set to an input set. The learning process consists in fixing a set of weights W that characterizes the connections between neurons, making the link between input

and output. The backpropagation algorithm as proposed by Rumelhart *et al* [1] is the most popular procedure. This is a relatively efficient procedure to estimate a set of weights and to get a satisfying relationship between input and output, as long as high precision is not required. However, the converging speed of this procedure is slow which is not surprising since the backpropagation algorithm is essentially a steepest descent method; in the optimization domain, theoretical and numerical works have shown that simple gradient methods have very slow convergence.

Several accelerating techniques were proposed to speed up the converging procedure. Fahlman [2], Jacobs [3], Tallenaere [4] proposed to modify dynamically some parameters such as the learning speed or the momentum; Rigler *et al* [5] suggested a scaling of the derivative as a function of successive levels. Leonard *et al* [6] improved the backpropagation algorithm with conjugated gradient techniques to adjust dynamically η and α with a unidirectional search at each optimization step. Van Ooyen *et al* [7] presented modifications in the error function used to measure the global net performance.

Theoretical and numerical results proved that Quasi-Newton algorithms are superior to gradient algorithms (Denis *et al* [8]). For this reason, several researchers proposed these techniques to train neural nets. Watrous [9] employed DFP and BFGS methods and compared them with the backpropagation algorithm; this comparison showed that DFP and BFGS methods need fewer iterations but each iteration requires the update of the hessian approximation and more calculation time. Parker [10] derived a formula for the update of the inverse hessian that suits a parallel implementation. Becker *et al* [11] and Kollias *et al* [12] proposed simple hessian approximations which, however, led to no significant time reduction in convergence. Because of these limitations, Barnard [13] proposed a stochastic method and noticed that his method is better than deterministic methods such as conjugated gradient and variable metrics. Bello [14] used a nonlinear least squares optimization algorithm enhanced with a Quasi-Newton algorithm for performing additional iterations of the "global-batch" optimization problem.

Many authors stated that Quasi-Newton methods are limited to middle-sized applications because of the computation time and the memory space needed to perform the update of the hessian approximation. This paper proposes a

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

modification to the classical approach of the Quasi-Newton method. It consists in a new hessian approximation that takes into account the structure of the network. More specifically, the hessian dimension is not the total number of weights to be calculated but is, instead, proportional to the number of neurons in each level. Since it limits the hessian dimension, our approach performs the training phase of a neural-net without the computation time and the memory space problem. The paper begins with the presentation of the mathematical formulation used for three classical approaches: the backpropagation, the steepest gradient and the Quasi-Newton method. It continues by outlining the hessian approximation and our two proposed modifications to this approximation. It compares our method with the classical approaches, applying them to two testing nets. It concludes with an analysis and discussion of results.

2 FORMULATION

2.1 Notation

The following models are developed for fully connected feedforward neural networks with a single hidden layer. The purpose of using a single hidden layer is for notation simplicity and the extension of each model to several layers is straightforward.

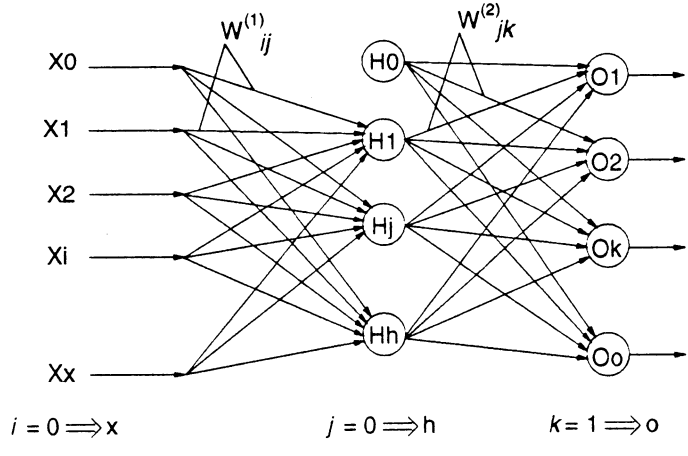


Figure 2.1 Schematic of a feedforward neural network

The input, hidden and output layers are noted $[X,H,O]$ respectively while the output training patterns are noted Y. $[x,h,o]$ are the total number of neurons in each layer $[X,H,O]$. $W^{L)}$ are the weights in level L and $f^{(L)}$ is the activation function for that level. With one hidden layer, L is equal to (1) or (2). X^0 and H^0, the bias for the corresponding layer, are always equal to one. The neurons H_j and O_k are calculated via activation function of the forms:

$$H_{pj} = f^{(1)}\left(\sum_{i=0}^{x} W_{ij}^{(1)} X_{pi}\right) \tag{1}$$

$$O_{pk} = f^{(2)}\left(\sum_{j=0}^{h} W_{jk}^{(2)} H_{pj}\right) \tag{2}$$

The training is done by presenting p presentations of input-output vectors $[X_p,Y_p]$ to the neural network and by minimizing a function of Y and O of the form:

$$E = \sum_{p=1}^{P} \sum_{k=1}^{o} Ep(Y_{pk}, O_{pk}) \tag{3}$$

where Ep is an error function.

2.1.1 Gradient evaluation Let i, j and k be the elements of the corresponding layer $[X,H,O]$ and i', j' be the element of two successive layers. Define an error function as:

$$Ep = \frac{1}{2}\sum_{k}(Y_{pk} - O_{pk})^2 \tag{4}$$

for one presentation p, so that:

$$E = \sum_{p} Ep \tag{5}$$

Define

$$\text{sum}_{pj} = \sum_{i} W_{ij}^{(1)} X_{pi} \tag{6}$$

and

$$\text{sum}_{pk} = \sum_{j} W_{jk}^{(2)} H_{pj} \tag{7}$$

The gradient is then expressed for the final layer by:

$$\frac{\partial E}{\partial W_{jk}^{(2)}} = -\sum_p f'^{(2)}(sum_{pk}) \quad (Y_{pk} - O_{pk}) \quad H_{pj} \tag{8}$$

and for the hidden layer by:

$$\frac{\partial E}{\partial W_{ij}^{(1)}} = -\sum_p f'^{(1)}(sum_{pj}) \quad \sum_k [f'^{(2)}(sum_{pk})(Y_{pk} - O_{pk})W_{jk}^2] \quad X_{pi} \tag{9}$$

Rewriting the gradients in a compact notation with an iteration index (n) gives for the gradient associated to one weight and for one presentation:

$$Sp_{i'j'}^{(L)}(n) = \frac{\partial Ep}{\partial W_{i'j'}^{L}(n)} \tag{10}$$

and for the sum of gradients for one weight:

$$S_{i'j'}^{(L)}(n) = \sum_p \frac{\partial Ep}{\partial W_{i'j'}^{L}(n)} \tag{11}$$

We define $S(n)$ as the gradient vector whose elements $S_m(n)$ are all the $S_{i'j'}^{(L)}(n)$ (the **bold** character refers to a vector).

$$S_m(n) = S_{i'j'}^{(L)}(n) \tag{12}$$

$$m = (j'-1)(x+1)+i' \qquad m \le (x+1)h$$

$$i' \in [0,x]$$

$$j' \in [1,h]$$

$$m = (j'-1)(h+1)+i' + (x+1)h \qquad m > (x+1)h$$

$$i' \in [0,h]$$

$$j' \in [1,o]$$

2.2 The classical methods

2.2.1 Steepest descent The simplest minimizing technique, when the gradient is available, is to select a descent direction opposed to the gradient and to apply a unidirectional search along this direction. This steepest descent optimization technique is defined by:

$$\Delta \boldsymbol{W}(n) = -\lambda \boldsymbol{S}(n) \tag{13}$$

where λ is a sufficient step for minimizing the function in the descent direction. It has been shown that there is no advantage in finding the exact minimum on the search direction at each iteration (Denis *et al* [8]).

2.2.2 Backpropagation The Backpropagation algorithm proposed by Werbos [15] and popularized by Rumelhart *et al* [1] is based on a variation of the steepest descent technique. There is no unidirectional search used but a fixed descent step, η, called learning rate which is added to a fraction of the last variation, α, called momentum. This algorithm is defined by:

$$\Delta_p W_{ij}^{(L)}(n) = -\eta S p_{ij}^{(L)}(n) + \alpha \Delta_p W_{ij}^{(L)}(n-1) \tag{14}$$

2.2.3 Quasi-Newton method By calculating the second derivative of the objective function, there is a better understanding of the function topology, which leads in turn to choosing a more efficient descent direction. Let:

$$\Delta \boldsymbol{W}(n) = \lambda \boldsymbol{G}(n) \tag{15}$$

where the descent direction $\boldsymbol{G}$ is defined by:

$$\boldsymbol{G}(n) = -[\boldsymbol{H}(n)]^{-1} \boldsymbol{S}(n) \tag{16}$$

and where $\boldsymbol{H}(n)$ is the Hessian matrix. The main difficulty with this approach is that finding the solution of this system at every iteration is very tedious. The variable metric methods, also called quasi-newton methods, bypass this difficulty by directly approximating the inverse Hessian matrix, $[\tilde{\boldsymbol{H}}(n)]^{-1}$, from the first derivative, $\boldsymbol{S}(n)$. These methods are the most popular unconstrained optimization techniques and **BFGS** is the most widely used update method, which calculates $[\tilde{\boldsymbol{H}}(n)]^{-1}$ by:

$$\Delta[\tilde{\boldsymbol{H}}(n)]^{-1} = [\tilde{\boldsymbol{H}}(n+1)]^{-1} - [\tilde{\boldsymbol{H}}(n)]^{-1} \tag{17}$$

$$\Delta[\tilde{\boldsymbol{H}}(n)]^{-1} = + \frac{[\Delta \boldsymbol{W}(n) - [\tilde{\boldsymbol{H}}(n)]^{-1}\,\Delta \boldsymbol{S}(n)]\,[\Delta \boldsymbol{W}(n)]^T + [\Delta \boldsymbol{W}(n)]\,[\Delta \boldsymbol{W}(n) - [\tilde{\boldsymbol{H}}(n)]^{-1}\,\Delta \boldsymbol{S}(n)]^T}{[\Delta \boldsymbol{S}(n)]^T \Delta \boldsymbol{W}(n)} \quad (18)$$

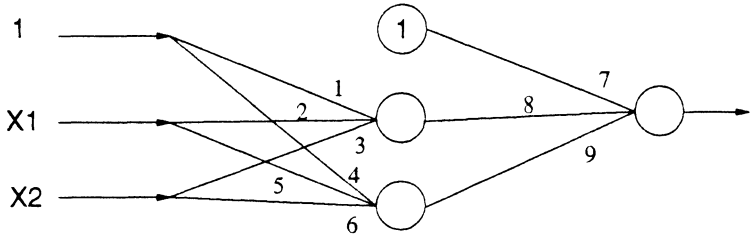
$$- \frac{[\Delta \boldsymbol{W}(n) - [\tilde{\boldsymbol{H}}(n)]^{-1}\,\Delta \boldsymbol{S}(n)]^T \Delta \boldsymbol{S}(n) \Delta \boldsymbol{W}(n)\,[\Delta \boldsymbol{W}(n)]^T}{[[\Delta \boldsymbol{S}(n)]^T \Delta \boldsymbol{W}(n)]\,[[\Delta \boldsymbol{S}(n)]^T \Delta \boldsymbol{W}(n)]}$$

3 A SIMPLIFIED FORM OF THE HESSIAN MATRIX

Our first approach neglects the second order interactions between weights of different levels and considers a separate matrix $\boldsymbol{H}$ for each level. The second approach assumes that only the weights connected to the same neuron have important second order interaction and it associates a matrix $\boldsymbol{H}$ to every output and hidden neuron. The major advantage in associating the Hessian matrix to level or neuron is that it reduces considerably the total size of the matrix to be calculated.

Figure 3.1 shows the size of the matrix applied to a typical neural network that solves the XOR problem for the following configurations: global, by level and by neuron. For example, the second derivative $\frac{\partial^2 E}{\partial w_5 \partial w_9}$ does not need to be calculated in configuration b and c. For the illustrated network, the matrix $\boldsymbol{H}$ with the global configuration (a) holds $(9)^2 = 81$ elements while the level configuration (b) holds $(6)^2 + (3)^2 = 45$ and the neuron configuration (c) holds $(3)^2 + (3)^2 + (3)^2 = 27$ elements.

The formulation of the variation of weights as a function of our simplified approach is very similar to the formulation of Quasi-Newton methods. They differ in the elements selected to form the vector gradient. In particular, for the weight variation by level, let $S^{(L)}(n)$ be the vectors whose elements are all the $S_{ij}^{(L)}(n)$ of a single level where:



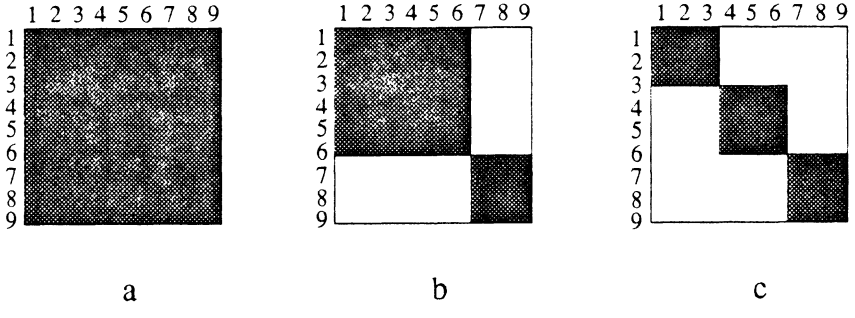


Figure 3.1 Schematic of second order interaction. The gray zones represent the dimensions of the Hessian matrix for different configurations: a) global, b) by level and c) by neuron.

$$S = [S^{(1)}, S^{(2)}] \tag{19}$$

and where $S^{(1)}$ holds $(x+1)h$ elements and $S^{(2)}$ holds $(h+1)o$ elements. The weight variation for each level will then be defined by:

$$\Delta W^{(L)}(n) = \lambda G^{(L)}(n) \tag{20}$$

with a descent direction for each level defined by:

$$G^{(L)}(n) = -[H^{(L)}(n)]^{-1} \quad S^{(L)}(n) \tag{21}$$

In the same way, we rewrite the weight variation as a function of neurons. Let $S_{j'}^{(L)}(n)$ be the vector whose elements are all the $S_{i'j'}^{(L)}(n)$ connected to the same neuron and where:

$$S^{(1)} = [S_1^{(1)}, S_2^{(1)}, \ldots, S_{j'}^{(1)}, \ldots, S_h^{(1)}] \quad (22)$$

$$S^{(2)} = [S_1^{(2)}, S_2^{(2)}, \ldots, S_{j'}^{(2)}, \ldots, S_o^{(2)}] \quad (23)$$

$S_{j'}^{(1)}$ holds $(x+1)$ elements and $S_{j'}^{(2)}$ holds $(h+1)$ elements. The weight variation is then defined by:

$$\Delta W_{j'}^{(L)}(n) = \lambda G_{j'}^{(L)}(n) \quad (24)$$

$$G_{j'}^{(L)}(n) = -[H_{j'}^{(L)}(n)]^{-1} \quad S_{j'}^{(L)}(n) \quad (25)$$

4 CASE STUDY

The performance of each of five algorithms is compared through two examples. The five algorithms are backpropagation (**BP**), steepest descent (**SD**), Quasi-Newton with **BFGS** update, **BFGS-L** which is the update proposed as a function of level in the network and **BFGS-N**, the update proposed as a function of neuron. For **BP**, the values of η and α are fixed to 0.7 and 0.9 respectively. For all the other algorithms, λ is evaluated by a unidirectional search method, which multiplies or divide the steps by two and is bounded by $[1x10^{-11},64]$. A sigmoid function is used as an activation function throughout the algorithms. Minimization stops when the objective function (equation 4) is smaller than $1x10^{-5}$. Both tests underwent 10 trials according to each of the five methods, in each trial different random initial weights were used; for each method, the initial weights used were the same.

The first test is the standard XOR test with 2 input, 2 neurons on the hidden layer and one output. The training set has four presentations where the input values, X, are 0 or 1 and the output values, Y, are 0.1 or 0.9.

The second test is the simulation of the reaction dynamics of three CSTR reactors in series with a second order reaction of the type $\mathbf{2A} \Rightarrow \mathbf{2B}$. The three input variables are the initial concentrations of reactant **A**, the temperature of the reactors and the volume flow rate, while the output variable is the residual concentration of reactant **A** at the outlet of the reactors. The network layout is made of an input layer with three neurons, a hidden layer with ten neurons and an output layer with one neuron. The three CSTR reactors in series are a highly non-linear system, particularly regarding temperature, and, for the modeling of

the reactor dynamics, it is necessary to have a training set of 45 presentations made up with the combination of three initial concentrations, three volume flow rates and five reactor temperatures.

5 RESULTS AND DISCUSSION

The comparison is done by considering the number of iterations and the time required to reach a solution. The number of cases reaching a solution is also indicated. For **BP**, the number of iterations is calculated at the end of the presentation of the entire training set. Table 5.1 presents the results for the XOR test problem. Cases where the method failed to converge in less than 2000 iterations were excluded from the statistics.

The two proposed methods, **BFGS-L** and **BFGS-N**, need much fewer iterations than the other algorithms. One may find it surprising that these two methods require less iteration than the standard **BFGS** algorithm. We have to note, however, that all the **BFGS** algorithms programmed here do not apply any correction to the Hessian matrix when these are non-definite positive, a process that is very time consuming. Instead, the Hessian matrix is replaced with the identity matrix and the method is then identical to the **SD** method for that iteration. For the XOR test problem, the Hessian matrix of the **BFGS** method is almost always non-definite positive, which is translated into a number of iterations very similar to the **SD** method. The **BP** method is last in terms of iterations required but comes second in terms of time and it reached a solution in every case. The **BFGS-N** method is the fastest of all and reached a solution in seven cases.

Table 5.2 presents the results for the three CSTR reactor problem. For this test, the maximum number of iterations to reach a solution was set at 10^6. This large number of iterations is comparable to the number used by Bello [14]. The **SD** and **BFGS** methods were incapable of meeting the stopping criterion in that number of iterations and the values of the error function at the end of calculations were around 1×10^{-4}. The comparison between **BP** and **BFGS-N** shows that **BFGS-N** requires one tenth of the iterations and is twice as fast, the number of converged solutions being similar for both methods.

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Table 5.1 Result of the XOR test problem

XOR	Iterations				Time		Conv.
	Ave.	Std.Dev.	Min.	Max.	Ave.	Std.Dev.	/10
BP	305	59.2	237	409	6.34	1.05	10
SD	245	62.1	153	319	10.61	2.62	9
BFGS	244	62.1	152	318	15.54	3.86	9
BFGS-L	66	26.6	47	105	9.07	2.86	4
BFGS-N	32	8.1	18	40	4.59	.99	7

Table 5.2 Result of the modeling of the three CSTR reactors.

CSTR	Iterations				Time		Conv.
	Ave.	Std.Dev.	Min.	Max.	Ave.	Std.Dev.	/10
BP	206974	281787	61765	948382	2188	3397	9
SD	-	-	-	-	-	-	0
BFGS	-	-	-	-	-	-	0
BFGS-L	75827	34501	35193	127064	3875	1823	8
BFGS-N	24211	20459	6991	63212	1092	910	8

CONCLUSION

Two modifications to the classical approach of the Quasi-Newton method have been presented. It was shown that the hypotheses supporting those methods are relevant and desirable in terms of convergence properties. The **BFGS-N** method, the proposed update as a function of neurons, is a very good alternative to the standard backpropagation algorithm. It represents a clear gain in terms of computational time without a major increase in memory space required, making the approach suitable for large scale problems. There is also no need to adjust parameters, as in the backpropagation algorithm, which makes our algorithm very easy to use.

ACKNOWLEDGMENT

This research project was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

1. Rumelhart, D.E., Hinton, G.E., Williams, R.J. 'Learning Internal Representation by Error Propagation' chapitre 8, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Rumelhart, D.E. and McClelland, J.L. editor, MIT Press, Cambridge, MA, 1986

2. Fahlman, S.E. 'An Empirical Study of Learning Speed in Back-Propagation Networks' internal report: CMU-CS-88-162, Carnegie Mellon University, Pittsburgh, Juin 1988

3. Jacob, R.A. 'Increased rates of convergences through learning rate adaptation' *Neural Networks*, Vol. 1, 29 p., 1988

4. Tallaneare, T. 'SuperSAB: Fast Adaptive backpropagation with good scaling properties' *Neural Network*, Vol. 3, pp. 561-573, 1990

5. Rigler, A.K., Irvine, J.M., Vogl, K. 'Rescaling of variables in backpropagation learning' *Neural Networks*, Vol. 4, pp. 225-229, 1991

6. Leonard, J.A., Kramer, M.A. 'Improvement of the BackPropagation algorithm for training neural networks' *Computer chem. Engng.*, Vol. 14, No.3, pp. 337-341, 1990

7. Van Ooyen, A., Nienhuis, B. 'Improving the Convergence of the Back-Propagation Algorithm' *Neural Networks*, Vol. 5, pp.465-471, 1992

8. Dennis, J.E., Schnabel, R.B. *Numerical Methods for Unconstrained Optimisation and Nonlinear Equations* Prentice-Hall, 1983

9. Waltrous, R.L. 'Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization' pp II-619-627, *IEEE Firs Int. Conf. Neural Networks*, San Diego, 1987

10. Parker, D.B. 'Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order Hebbian learning' pp II-593-600, *IEEE Firs Int. Conf. Neural Networks*, San Diego, 1987

11. Becker, S., Le Cun, Y. 'Improving the convergence of back-propagation learning with second order methods' (D. Touretzky, G. Hinton and T. Sejnowski, Eds), pp. 29-37, *Proc. Connectionist Model summer School*, Morgan-Kaufman, San Mateo, 1988

12. Kollias, S., Anastassiou, D. 'An adaptive least squares algorithm for the efficient training of artificial neural networks' *IEEE Transaction Circuits Systems*, Vol. 36, pp. 1092-1101, 1989

13. Barnard, E. 'Optimization for Training Neural Nets' *IEEE Transaction on Neural Networks*, Vol. 3, No. 2, pp. 232-240, 1992

14. Bello, M.G. 'Enhanced Training Algorithms, and Integrated Training/Architecture Selection for Multilayer Perceptron Networks' *IEEE Transaction on Neural Networks*, Vol. 3, No. 6, pp. 864-875, 1992

15. Werbos, P.J. *'Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences'*, applied mathematic thesis, Harvard University, 1974