Annales UMCS Informatica AI 2 (2004) 31-36

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Two modifications of Levenberg-Marquardt's method for fast batch neural network training

Viktor Sobetskyy[*], Stanisław Grzegórski

Department of Computer Science, Lublin University of Technology,
Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The problems of artificial neural networks learning and their parallelisation are taken up in this article.

The article shows comparison of the Levenberg-Marquardt's method (LMM) and its two modifications JWM (method with Jacobian matrices formed in each step) and BKM (Jacobian calculations only in the first step) for training artificial neural networks. These algorithms have the following properties: 1) simpler calculations; 2) they are partly parallelized. The experiments proved their efficiency. Experimental results demonstrate that neural network for training by them needs a similar number of epochs as the LMM and lesser time for training.

1. Introduction

The learning for the big feedforward artificial neural networks with a large training set takes a long time (in terms of days) [1,2], it also becomes imperative to look at train algorithms modifications and parallel implementation schemes to reduce this training time.

The experience of training feedforward artificial neural networks by Levenberg-Marquardt's method and its JWM and BKM modifications on single processor computer and realization for multiprocessors cluster are stated in the article.

LMM is the fastest and most popular of Newton's methods [3]. These methods use the batch training mode, rather than the pattern mode which is based on derivatives of instantaneous errors. We propose two modification schemes of it. These algorithms have lesser convergence than LMM, but calculation time lesser than LMM.

[*] Corresponding author: *e-mail address*: viktor@pluton.pol.lublin.pl

2. LMM

Levenberg-Marquardt algorithm originates from the Gauss-Newton method. It is the fastest and most popular of Newton's methods. These methods use the batch training mode, rather than the pattern mode which is based on derivatives of instantaneous errors [1,2].

For simplicity, we consider two layer perceptrons with weights matrice $\boldsymbol{W}^{\boldsymbol{h}}$, $\boldsymbol{W}^{\boldsymbol{y}}$, ***p***, ***L***, ***m*** – number neurons in input, hidden, output layers accordingly. The weight vector **w** is formed by scanning these matrices in rows.

The training set consists of ***N*** examples.

Let us assume that:

– The length of **w** is

$$K = L\boldsymbol{(}\, p + m\,\boldsymbol{)}\,.$$

– All weights have been arranged in one vector

$$\mathbf{w} = \left[\, w_1 ... w_j ... w_K \,\right].$$

– All (instantaneous) errors form a column vector

$$\varepsilon(\mathbf{w}, n) = \mathbf{d}(n) - \mathbf{y}(n) = \left[\varepsilon_1 ... \varepsilon_k ... \varepsilon_m\right]^T .$$

– The instantaneous performance index

$$E(\mathbf{w}, n) = \frac{1}{2}\sum_{k=1}^{m} \varepsilon_k^2(n) = \frac{1}{2}\varepsilon(n)\cdot\varepsilon^T(n).$$

– The total performance index

$$F(\mathbf{w}) = \frac{1}{M}\sum_{n=1}^{N} E(\mathbf{w}, n),$$

where $M = mN$ (m – number of outputs).

The symbol ***F*** is used instead of ***J*** to avoid confusion with the Jacobian matrix.

– The instantaneous Jacobian matrix, ***J(n)***, is $\boldsymbol{m\times K}$, one column per weight, and can be partitioned into two blocks related to the hidden and output weights, respectively:

$$\boldsymbol{J}(\boldsymbol{n}) = \left[\boldsymbol{J}^{\boldsymbol{h}}(\boldsymbol{n}) \;\; \boldsymbol{J}^{\boldsymbol{y}}(\boldsymbol{n})\right].$$

– The batch weight update is the linear equations system solving

$$\left(\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{w})\cdot\boldsymbol{J}(\boldsymbol{w}) + \mu\cdot\boldsymbol{I}\right)\cdot\Delta\mathbf{w}^{\boldsymbol{T}} = -\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{w})\cdot\boldsymbol{e}(\boldsymbol{w}),$$

where μ is a small constant, ***I*** is the identity matrix.

We propose to solve the system of linear equations using Householder's transformation [4]. This numeric method allows getting good solution accuracy. Its parallelization gains good decreasing training time results, as well.

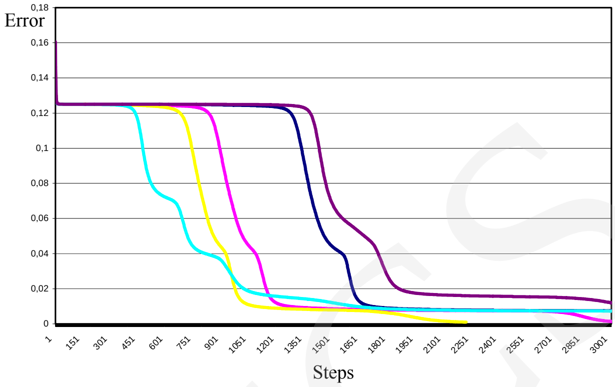

Fig. 1. Functional dependence of total perform index from the number of epochs demonstrate of LMM

LMM algorithm convergence is demonstrated in Figure 1. Every line in the figure corresponds to different weights vectors $\boldsymbol{w_0}$, but they have the same training set ***N.***

We attempt to parallelize it for the work on cluster. To solve this system we use Householder's transformation for each column of the system matrix. After those transformations, the matrix will gain triangle shape.

We can parallelize some of the algorithm parts [2].

The matrix columns are shared between the processors and that way processor p_i gets columns $\{c_i, c_{N+i}, c_{2N+i}, ...\}$, N – mean number of processors, p_i – processor number, c_{ij} – column number.

Then one by one we define Householder's transformation of each column. The vector we get is sent to the other processors [4]. The next step is based on each processor usage of the vector it gets to transform its own vectors.

The parallelization of LMM algorithm for different training sets and the same weights vectors $\boldsymbol{w_0}$ is demonstrated in Figure 2. The number of columns in Jacobian is equal to the vector $\boldsymbol{w}$ size. One epoch is the time for one batch weight update calculation.

3. JWM and BKM

The JWM is a method of training artificial neural network. It is LMM modification for $\mu = 0$ case. Then weights update system linear equation becomes

$$\boldsymbol{J}(\boldsymbol{w}) \cdot \Delta \mathbf{w}^{\boldsymbol{T}} = \boldsymbol{e}(\boldsymbol{w}).$$

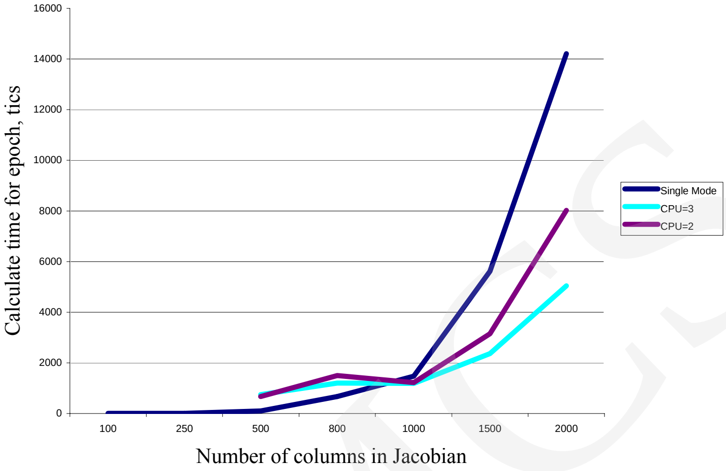

Fig. 2. Dependence of training time on the number of columns in Jacobian in single computer and 2,3 processor cluster

This is a standard Gauss-Newton method. One problem of the Gauss-Newton method is that the system of linear equations does not often have solutions. For solving this problem we update the weights in the $\boldsymbol{k}+1$ iteration by the next rule:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \boldsymbol{t}\cdot\Delta\boldsymbol{w} + \alpha$$

where $\boldsymbol{w_k}$ is the previous weights value, $\Delta\boldsymbol{w}$ is the batch weights update, $\boldsymbol{t}$ is the learning rate variation, α is the noise (small random value).

We use conjugate gradient algorithm use to find learning rate variation $\boldsymbol{t}$ [3,5].

The results of the artificial neuron network learning by means of JWM for test patterns are demonstrated by diagrams in Fig. 3.

The other BKM method is the BFGS algorithm modification. It does not need to calculate Jacobian matrix for each iteration. Instead we calculate $\boldsymbol{B_k}$ matrix. It is defined as:

$$\boldsymbol{B}_0 = \boldsymbol{J}\left(\boldsymbol{w}\right),$$

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{r}_k\boldsymbol{s}_k^T}{\boldsymbol{s}_k^T\boldsymbol{s}_k},$$

$$\boldsymbol{r}_k = \boldsymbol{B}_k\boldsymbol{s}_k - \boldsymbol{y}_k,$$

$$\boldsymbol{s}_k = \boldsymbol{w}_{k+1} - \boldsymbol{w}_k,$$

$$\boldsymbol{y}_k = \boldsymbol{e}\left(\boldsymbol{w}_{k+1}\right) - \boldsymbol{e}\left(\boldsymbol{w}_k\right),$$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \boldsymbol{t}\cdot\Delta\boldsymbol{w} + \alpha,$$

where $\boldsymbol{w_k}$ is the previous weights value, $\Delta \boldsymbol{w}$ is the batch weights update, $\boldsymbol{t}$ is the variation of learning rate, α is the noise (small random value). We use conjugate gradient algorithm to find variation learning rate.

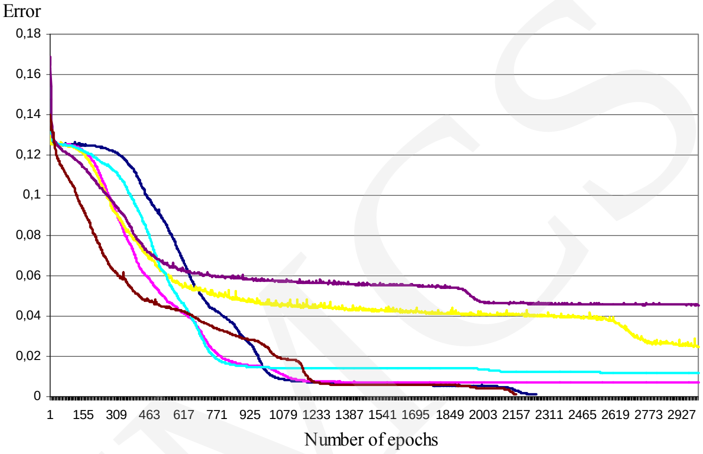

Fig. 3. Functional dependence of total perform index on the number of epochs demonstrate of JWM

The results of the artificial neuron network learning by means of BKM for test patterns are demonstrated by diagrams in Fig. 4.

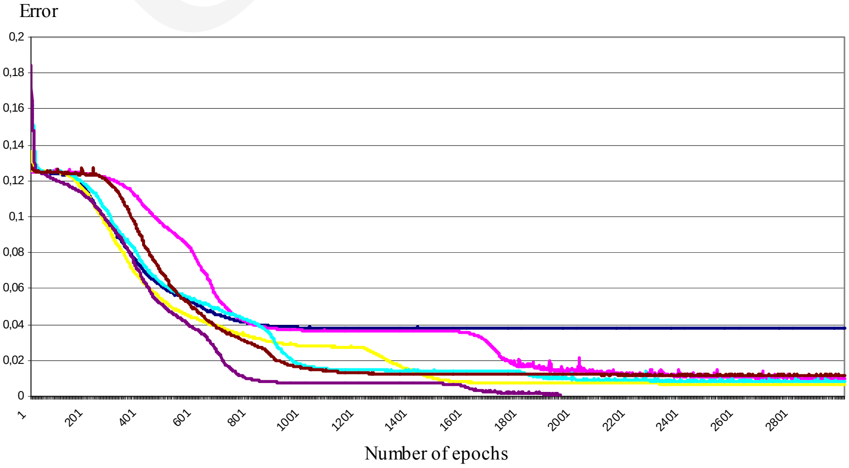

Fig. 4. Functional dependence of total perform index on the number of epochs demonstrated for BKM

Training time per epoch is demonstrated in Figure 5. The upper line is for the LMM time, the lower line is for JWM and BKM. The number of columns in Jacobian is equal to the vector ***w*** size. One epoch is the time for one batch weight update calculation.

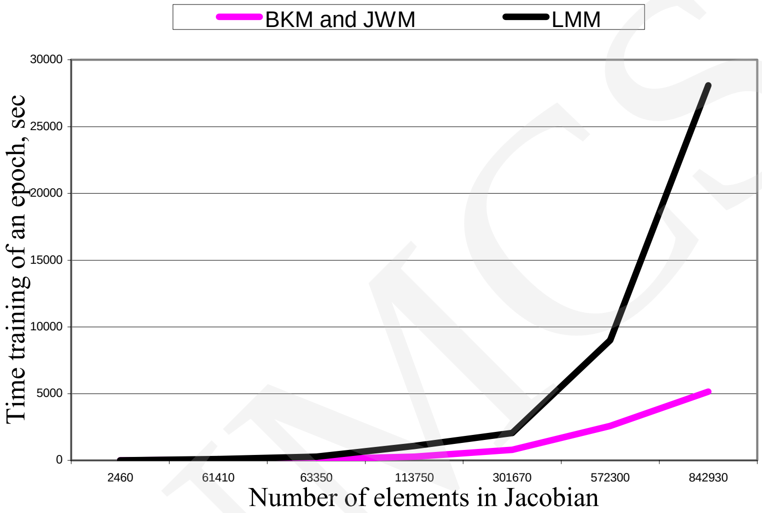

Fig. 5. Calculating time of one epoch

4. Conclusions

This paper presents two algorithms for artificial networks learning. The main advantage of the proposed algorithms is that they allow to train neural network faster than the standard Levenberg-Marquardt's algorithm. BKM method does not need calculating Jacobian matrices of each step. Algorithms give good results for recognition task accomplishing.

JWM and BKM are interesting with respect to parallelisation. Both methods are partially parallel. The next experiment series could show which of them is faster.

References

[1] Paplinski A.P., *NNets*, L7, (2002).

[4] Srinivasan A., *Givens and Householder Reduction for Linear Last Squares on Cluster Workstations*, University of California at Santa Barbara.

[2] Sobetskyy V., Grzegorski S., *Use clusters for training neural networks in tasks detection and recognition of persons*, Proceedings of IASTED international conference Artificial Intelligence and Applications, ISBN: 0-88986-375-X, (2004).

[3] Vojevodin V.V., Vojevodin V.L., *Parallels computing*, BXV-Petersburg, (2002).

[5] Jezhov A., Shumski S., *Neurocomputing and it implementations in economic*, MIFI, (1998).