ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 *2011. No. 1(107)*
ELEKTRONIKA IR ELEKTROTECHNIKA

SYSTEM ENGINEERING, COMPUTER TECHNOLOGY
T 120
SISTEMŲ INŽINERIJA, KOMPIUTERINĖS TECHNOLOGIJOS

Tuning Fuzzy Perceptron using Parallelized Evolutionary Algorithms

Gh. Radu, I. Balan, I. Ungurean
Electrical Engineering and Computer Science Faculty, "Stefan cel Mare" University of Suceava, Universitatii str. 13, 720229 Suceava, Romania, phone: +40 723 604 084, e-mail: gh.radu@gmail.com

Introduction

Dumitrescu [1–5] proposed two fuzzy training procedures (Fuzzy Perceptron and the Fuzzy Relaxation). In [3] some evolutionary algorithms for tuning these training procedures are given. The aim of this paper is to review the approach from [3] and to give some parallelized algorithms concerning this approach, using the processing capabilities of a high performance processor cluster.

The paper is structured in three main sections. The first section presents the two training procedures. The second one is a short presentation of our approach on evolutionary algorithms. In the third section the results obtained by running the parallel algorithms are presented. Conclusions are drawn in the final section.

Two fuzzy training procedures

Problem: find a separation vector for fuzzy classes A_1, A_2.
Solution: Fuzzy Perceptron Algorithm.
Minimize distance of misclassified points to the (separation) hyperplane

$$(H): v^T z = 0. \tag{1}$$

Criterion function

$$J(v) = \sum_{i=1}^{2} \sum_{z \in E_i(v)} A_i(z)\left(-v^T z\right), \tag{2}$$

where

$$E_j(v) = \left\{ z \mid v^T z < 0,\ A_j(z) > 0.5 \right\},\ \text{j=1,2.} \tag{3}$$

Optimization problem: minimize $J(v),\ v \in R^{s+1}$.

Modifications: Consider a training sequence $\left(z^n\right)$, obtained by considering cyclically the vectors $z^1, z^2, \ldots z^p$. At the k-th step of the algorithm, consider only *one* training vector. Then, the *Fuzzy Perceptron Training* rule is

$$v^{k+1} = \begin{cases} v^k + cA_i\left(z^k\right) z^k, if A_i\left(z^k\right) > 0.5 and \left(v^k\right)^T z^k \le 0. \\ v^k, otherwise. \end{cases} \tag{4}$$

Remark: We have considered points *on* the separation hyperplane that as misclassified (i.e. condition $v^{kT} z^k < 0$ becomes $v^{kT} z^k \le 0$).

Now, for a better classification, a stronger separation condition will be considered

$$v^T z \ge b,\ b>0. \tag{5}$$

We can define the criterion function as the sum of squared distances with respect to the fuzzy training classes of the misclassified points to the separating hyperplane

$$J'(v) = \sum_{i=1}^{2} \sum_{z \in E_i'(v)} \left(A_i(z)\right)^2 \frac{(v^T z - b)^2}{\|z\|^2}, \tag{6}$$

where $\|v\|^2 = v^T v$ and

$$E_j'(v) = \left\{ z \mid v^T z < b,\ A_j(z) > 0.5 \right\},\ j = 1,2. \tag{7}$$

We will make the following assumptions: consider at the step k only the sample z^k; consider 0<c<2 and a correction is also made for a sample z^k for which

$$\left(v^k\right)^T z^k = b. \tag{8}$$

Thus the correction rule becomes:

$$v^{k+1} = \begin{cases} v^k + c\left(A_i\left(z^k\right)\right)^2 \frac{b - \left(v^k\right)^T z^k}{\|z^k\|^2} z^k, \\ if \left(v^k\right)^T z^k \le b \ and \ A_i\left(z^k\right) > 0.5, \\ v^k, otherwise. \end{cases} \tag{9}$$

The iterative procedure which uses this correction rule is called *Fuzzy Relaxation Training Algorithm*.

Evolutionary algorithms for tuning training procedures

The components of the evolutionary algorithms:

a) Representation and initialization. A chromosome corresponds to a solution (*weight* or *separation*) vector *v*

$$v^j \quad (v_1^j, v_2^j, ..., v_{s+1}^j), \tag{10}$$

where each gene v_i^j of a chromosome is a real number.

Besides the $s+1$ genes of a chromozome, another one will be added, b; this coresponds to the H' hyperplane's margin from the Fuzzy Relaxation Training Algorithm. Thus the real codification of the chromosome is

$$(v_1^j, v_2^j, ..., v_{s+1}^j, b). \tag{11}$$

b) Fitness function. Since genetic algorithms are designed to maximize functions, we will consider the fitness function f defined as $f(v) = Max - J(v)$ (or $f(v) = Max - J'(v)$ for Fuzzy Relaxation Training Algorithm), where *Max* is a constant used to ensure that $f(v) \geq 0, \forall v$. *Remark:* Maximizing f $\Rightarrow$ minimizing J. Fitness of the chromosome v is $eval(v) = J(v)$.

c) Selection. For selecting the parents, two types of selection are implemented:

- *proportional* selection, where the probability of a chromosome v^i is

$$p_i = \frac{f(v^i)}{\sum_{v^j \in P(t)} f(v^j)}, \tag{12}$$

- *tournament* selection, two versions: binary tournament and *q*-tournament, where *q* is an integer number between 6 and 10.

Survivor selection is an elitist selection: all parents and offspring compete for becoming members of a new generation (evolution-based strategy).

We will take two generational models:

- *steady-state* model: in each generation an old individual is replaced with a new one,
- *custom* model: allows the combination of different types of variation operators and selection strategies.

d) Mutation. Consider 2 options:

- mutate a single gene of selected chromosome; a normal perturbation is useful (uniform repartition),
- mutate each gene of selected chromosome.

In both cases the creep mutation is used, the gene, or genes, being "altered" by adding a constant quantity. The initial value of the mutation constant is 0.5.

e) Recombination. Consider 2 options:

- from the pair of parents (a, b), we obtain *two offspring, c* and *d*, by the following formulas:

$$\begin{cases} c_i = \alpha_i a_i + (1-\alpha_i) b_i, \ i = 1, \ldots, s+1, \\ d_i = (1-\alpha_i) a_i + \alpha_i b_i, \ i = 1, \ldots, s+1, \end{cases} \tag{13}$$

where a_i is a random number with uniform distribution on $[0, 1]$.

- from the pair of parents (a,b), we obtain *one offspring, c*, by the same formula.

Parallelizing evolutionary algorithms

The evolutionary algorithms used for solving real problems consume a large amount of resources, a fact that prolongs the time needed for obtaining the results.

For testing the evolutionary training algorithms, we used a parallelized version of the algorithms by using the OpenMPI library. The parallelization of the algorithms was made on its data; the population is equally shared between the MPI processes that run simultaneously. The algorithm has been implemented in two different versions. In the first version, the population is actualized in each MPI process, at each step each process does the recombinations and mutations using chromosomes from all the population. In the second version, each MPI process has a diffrent populaton than the other processes, each MPI process does the recombinations and mutation using chromosomes from the local population. The tests have been conducted on the RedPOWER HPC cluster within the high performance computing laboratory from „Stefan cel Mare" University of Suceava. This cluster is made of 28 nods and each node is made by 2 Xeon Quad Core E5345 80W 2.33GHz/1333MHz/8MB L2 processors. The communication is done by 2 Gigabit networks, one used for administration purposes and the other for intensive calculus. Each node can deploy 8 MPI proceses (2 processors * 4 cores = 8 cores) at a time [5].

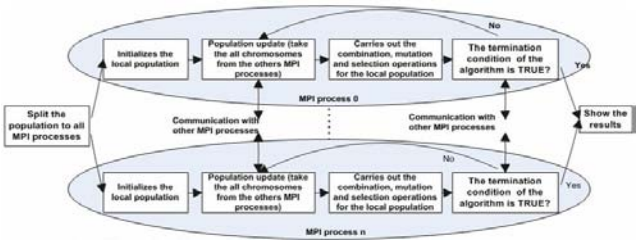

Fig. 1. The algorithm's parallel execution mode is shown

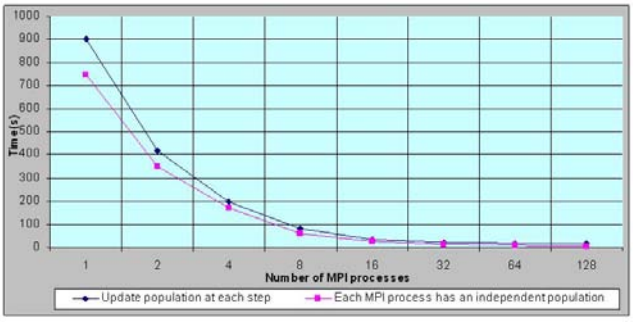

Fig. 2. Obtained results

The results shown in Fig. 2 were obtained by running the algorithm with the Pima Indians Diabetes Data Set [4], a set composed by 8 attributes where the identifier of the class has a number of 768 forms. In order to obtain the corresponding fuzzy training sets we used Generalized Fuzzy n-Means (GFNM) Algorithm [2] setting n=2. To be able to use the advantages provided by the parallel

architectures we need a quite large volume of data, therefore the time needed for communication between the nodes is insignificant against the time needed for processing the data. In order to prove the efficiency of using a multi-processor architecture, we have used a population of 12800 chromosomes that will evolve in several generations. We can notice in Fig. 1 how the processing time is decreasing with the increase of the number of cores, up to a certain limit, when the time is starting to increase. This increase is especially determined by the increased number of communications between the nodes and decreased volume of data for each core. The cores are than, not used to their maximum capacity but waiting for the communication.

Also from Fig. 1, assuming that certain delays due to different reasons are neglected, we can state that he difference between the two evolutions represents the time needed for communication.

Table 1. The results obtained for different versions of evolutionary algorithms

GA type	CA2Q*		S**		CA1B***	
Nr.	Steps	Execution time (s)	Steps	Execution time (s)	Steps	Execution time (s)
1	78	12,551414	31	132,86864	69	11,919009
2	78	12,223929	29	126,359432	62	10,776089
3	70	10,617825	29	131,983591	74	13,499302
4	62	9,371246	31	134,68343	72	13,633883
5	69	10,438041	31	133,769158	64	11,133868
Avg	71,4	11,040491	30,2	131,93285	68,2	12,19243
Best	62	9,371246	29	126,359432	62	10,776089

*Note: * custom evolutionary algorithm with mutations on all genes, recombinations 2 parents ⇒ 2 children, 6-tournament.*

*** steady-state generational model.*

**** custom evolutionary algorithm with mutations on all genes, recombinations 2 parents ⇒ 1 child, binary selection.*

In Table 1 the results obtained by running the different versions of evolutionary algorithms on the same data set (Prima Indians Diabetes Data Set) are shown, but having at our disposal a population of 100 chromosomes and a sequential structure. The running execution times in CA2Q and CA1B cases are close to the executions times of the algorithm with 12800 chromosomes on 128 cores. The CA2Q version consists in running the algorithm when all mutations affect all genes of the chromosomes composing the population, thus the recombinations being made allows the obtaining of two offspring from two parents, and the selection is of tournament type, therefore six by six chromosomes are faced with each other, in random order, out of which only one is chosen, the one that survives. In the CA1B case, compared to the first version of evolutionary algorithm, the recombinations determine the obtaining of only one offspring from 2 parent chromosomes, while the selection is binary, therefore 2 by 2 chromosomes are faced with each other, taking one chromosome from the population and the other random generated from the same population, the best of them will survive.

In the case of the S version we deal with a steady evolutionary model. Therefore, in this case, we deal first with some recombinations of each of the population's chromosomes and a chromosome randomly generated, and then the obtained chromosome will be mutated, the last obtained chromosome will replace the least satisfying chromosome. We can see the number of generations needed for attaining the desired result for each different configuration. The steady model needs an average of only 30 steps, but the downside of this algorithm is the quite big execution time, while the other two (custom) versions, even if the number of generations to be traveled is much bigger, the executions time is quite short. We can notice this in Table 2, presented below. By the number of steps needed for touching the desired result we mean, basicaly, the number of generations that the evolution travels in order to obtain the final result.

Table 2. Number of steps on different nodes

No. of nodes	1	2	4	8	16
No. of steps	70	64	52	52	51

Conclusions

The execution times, as well as the number of steps needed for obtaining the final result have different values, from one execution to the other, because when dealing with these types of algorithms with evolutionary techniques the random process is of significant importance. That's why, significant for the present paper is the average value of the obtained results, while the Best value shows the results that can be obtained only in idealistic conditions. By paralelizing the evolutionary algorithms, besides the gain in the execution time area, we can also register some gainings in the number of steps required for touching the desired result.

References

1. **Dumitrescu D.** Fuzzy Training Procedures I // Fuzzy Sets and Systems. – Elsevier Science Publishers B.V., 1993. – Vol. 56 – P.155–169.
2. **Dumitrescu D., Lazzerini B., Jain L. C.** Fuzzy Sets and Their Application to Clustering and Training. – USA: CRC Press, 2000. – 622 p.
3. **Radu Gh**. Tuning Fuzzy Perceptron Using Evolutionary Algorithms // Proceedings of "Communication 2004" International Conference. – Bucharest, 2004. – Vol. 1. – P. 303-308.
4. **Sigillito V.** UCI Machine Learning Repository. - Research Center, RMI Group Leader, Applied Physics Laboratory, The Johns Hopkins University. Online: http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

5. **Komornicki A.**, **Mullen-Schultz G.**, **Landon D.** Roadrunner: Hardware and Software Overview. – USA: IBM Redbooks, 2009. – 50 p.

Received 2010 02 15

Gh. Radu, I. Balan, I. Ungurean. Tuning Fuzzy Perceptron using Parallelized Evolutionary Algorithms // Electronics and Electrical Engineering. – Kaunas: Technologija, 2011. – No. 1(107). – P. 51–54.

Evolutionary computation can be used as independent instrument to establish the neural network weights. We assume that the network architecture is known. Some evolutionary algorithms as training procedures of fuzzy perceptron have been proposed before. In this paper, we presented a new hybridization between evolutionary algorithms (used as training procedures of fuzzy perceptron) and parallel algorithms. Using a high performance processor cluster with 28 nodes we will try to get better results in much smaller intervals of time. The kernels used to solve the problem are of the same type, they are eight on each node and each of them is working on the same frequency. The computational results show the validity of new approach in terms runtime, accuracy and flexibility. Ill. 2, bibl. 5, tabl. 2 (in English; abstracts in English and Lithuanian).

Gh. Radu, I. Balan, I. Ungurean. Lygiagretaus atsinaujinančio algoritmo taikymas neraiškiuosiuose perceptrono tipo dirbtinio intelekto neuroniniuose tinkluose // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2011. – Nr. 1(107). – P. 51–54.

Neuroninių tinklų įverčiams nustatyti gali būti taikomi atsinaujinatys skaičiavimai. Keletas atsinaujinančių algoritmų jau buvo pateikti ir pasiūlyti anksčiau. Laikoma, kad tinklo struktūra yra žinoma. Siūloma nauja sąsaja tarp atsinaujinančiųjų ir lygiagrečiųjų algoritmų. Trumpesniuose laiko intervaluose, taikant labai našius procesorius su dvidešimt aštuoniais nodais, siekiama gauti geresnius rezultatus. Procesoriuose naudojami tokie pat ir to paties taktinio dažnio branduoliai. Gauti rezultatai rodo metodo lankstumą ir tikslumą. Il. 2, bibl. 5, lent. 2 (anglų kalba; santraukos anglų ir lietuvių k.).