ANNIIP 2009

Kurosh Madani (Ed.)

Artificial Neural Networks and Intelligent Information Processing

Proceedings of the
5th International Workshop on
Artificial Neural Networks and Intelligent Information Processing - ANNIIP 2009

INSTICC PRESS

In conjunction with ICINCO 2009
Milan - Italy, July 2009

Kurosh Madani (Ed.)

Artificial Neural Networks and Intelligent Information Processing

Proceedings of the
5th International Workshop on
Artificial Neural Networks and Intelligent Information Processing
Workshop ANNIIP 2009

In conjunction with ICINCO 2009
Milan, Italy, July 2009

INSTICC PRESS
Portugal

Volume Editor

Kurosh Madani
The University of PARIS XII
France

5th International Workshop on
Artificial Neural Networks and Intelligent Information Processing
Milan, Italy, July 2009

Printed in Portugal

ISBN: 978-989-674-002-3
Depósito Legal: 294692/09

Foreword

Five years have revolved since the first international workshop on Artificial Neural Networks and Intelligent Information Processing (ANNIIP) and it is a great pleasure to notice that the ever-increasing creativity, revealing the fantastic intellectual dynamics created around bio-inspired Artificial Intelligence, remains intact. If applicative and technological challenges, emanating from nowadays' industrial, socioeconomic or environment needs, open every day new dilemmas to solved, the escalating interest of both confirmed and young researchers on this relatively juvenile science, upholds a reach multidisciplinary synergy between a large number of scientific communities making conceivable a forthcoming emergence of viable solutions to these real-world complex challenges.

Since 2005, ANNIIP international workshop, within the frame of the prestigious ICINCO International Conference, takes part in the aforementioned appealing dynamics by offering a privileged space to refit and exchange the knowledge about further theoretical advances, new experimental discoveries and novel technological improvements in the promising area of the bio-inspired Artificial Intelligence (and related topics as: Artificial Neural Networks, Humanoid Robotics, Ambient Intelligence, etc...). The present book is the outcome of the fifth edition of this annual event.

Within this inveterate philosophy and around a deliberately limited number of papers, the objective of this fifth volume is to convene once more relevant recent works focusing this exciting topic, related fields and issued applications. Conformably to our values, the choice of publishing a restricted number of papers is persistently motivated on the one hand by the premeditated desire to give a large space to exchanges and discussions during the workshop, and on the other hand by the strong principle of the presentation of each accepted article by its authors. Our constant fondness for the two above-mentioned inseparable and complementary attitudes rolls up from our sturdy wish to respect the further engaging feature inherent to a workshop which is the opportunity of a constructive "viva discussion" around the published works. If "Bio-inspired Artificial Intelligence" and its real-world applications remain, as in the previous editions of this international workshop, the foremost premises of this 5th volume, a special attention has been devoted to the balance between theoretical and applicative aspects.

It is important to remind that scientific relevance and technical excellence of a collective volume emerge from quality of its contributors: those who have contributed by the high quality of their manuscripts and those who have taken part in reviewing of submitted papers ensuring, by their valuable expertise, the distinction of the present book. I would like to express again my acknowledgements to contributors of all accepted papers: You are the central reason of the nobles of this tome. Also, I would like to reedit my gratitude to Reviewing Board and Program Committee for the valuable work that they accomplished: My heartfelt recognition to those who already were members of ANNIIP Program Committee as well as my sincere thanks to those who kindly accepted to enlarge the Reviewing Board of this 5th workshop's edition.

It is also essential to be reminiscent that frequently, creative dynamics is the result of fruitful humans' contacts within a same scientific field or the consequence of humans' interactions from different scientific communities and since 2004, the date of the its first edition, the ICINCO multi-conference has been an outstanding bench of such creative synergies. For that, again, I would like to express my warm appreciation and my particular gratitude to my friend Prof. Joaquim Filipe, ICINCO 2009 Conference's Chair, for his faith in young science of "Bio-inspired Artificial Intelligence" and for his reliance on devoting once more this privileged space to the ANNIIP workshop within his valuable conference.

Finally, if ICINCO Organizing Committee's professionalism became an obvious skill of this international event's organization in so accurate way, it should never be forgotten that the organization of a prestigious conference remains a challenging undertake requiring a reliable and a solid team. So, I would like to acknowledge whole the organizing team, with a special word for Marina Carvalho from Workshops Secretariat who, during these five revolved years since 2005, has proved her irreplaceable merit as a key person in ANNIIP workshop's organization.

July 2009,

Kurosh Madani
The University of PARIS XII, France

Workshop Chair

Kurosh Madani
The University of PARIS XII
France

Program Committee

Ajith Abraham, Machine Intelligence research Labs (MIR Labs), U.S.A.
Veronique Amarger, PARIS-EST University, France
Gilles Bernard, PARIS 8 University, France
Ezzedine Ben Braiek, Ministery of High Studies, Tunisia
Abdennasser Chebira, LISSI, France
Amine Chohra, Paris Xii - Val De Marne University - Senart-fontainebleau Institute of Technology, France
Suash Deb, C. V. Raman College of Engineering, India
Khalifa Djemal, University of Evry Val d'Essonne, France
Peter Géczy, AIST, Japan
Vladimir Golovko, Brest State Technical University, Russian Federation
Robert Hiromoto, University of Idaho, U.S.A.
Dattatraya Kodavade, DKTE Society's Textile & Engineering Institute Ichalkaranji, India
Thomas Laengle, Fraunhofer Iitb, Germany
Hichem Maaref, Université D'evry Val D'essonne, France
Jean-Jacques Mariage, PARIS 8 University, France
M'sirdi K. Nacer, LSIS, France
Georgy Panev, SPIIRAN - Russian Academy of Science, Russian Federation
Leszek Rutkowski, Technical University of Czestochowa, Poland
Mariusz Rybnik, Finance & Management University, Poland
Christophe Sabourin, Laboratoire Images, Signaux, Et Systèmes Intelligents, France
Anatoly Sachenko, Research Institute of Intelligent Computer Systems / Ternopil National Economic University, Ukraine
Khalid Saeed, AGH University of Science and Technology, Poland
Lamine Thiaw, Ecole Supérieur Polytechnique de Dakar, Senegal

Table of Contents

Modelling, Learning and Design of ANNs

Evolving Gradient a New Approach to Perform Neural Network Training

César Daltoé Berci and Celso Pascoli Bottura

Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas
Av. Albert Einstein 400, Campinas, Brazil
cesardaltoe@hotmail.com, cpbottura@fee.unicamp.br

Abstract. The use of genetic algorithms in ANNs training is not a new subject, several works have already accomplished good results, however not competitive with procedural methods for problems where the gradient of the error is well defined. The present document proposes an alternative for ANNs training using GA(Genetic Algorithms) to evolve the training process itself and not to evolve directly the network parameters. This way we get quite superior results and obtain a method competitive with these, usually used to training ANNs.

1 Introduction

Artificial Neural Networks is a computational paradigm inspired in the operation of the biological brain, especially in the human brain, and seeks to explore certain properties present in the human neural processing, that are very attractive from the computational view point. Among the principal characteristics of the biological information processing, the following can be mentioned [9]:

- Robustness and fault Tolerance. The human brain possesses a great number of neurons and even losing thousands of them, the brain may continues in operation without losing its capacities.
- Flexibility. There is no need to reprogram the system when exposed to new unknown situations. In these cases the brain has the capacity to assimilate the new scenery and to adapt to it.
- Possibility of working with fuzzy, probabilistic, noisy and inconsistent information. The neural computation has the intrinsic ability to work with uncertainties, which conventionally requires a high sophistication level to be treated by more conventional computational paradigms.
- Parallelism. Neural computers, as the human brain, are parallel in their essence, what turns them highly efficient for treating certain problems.

Another computational method, also bio-inspired, are the genetic algorithms, based on *Charles Darwin's work*, more precisely in his book *The Origin of Species* [6] where the author idealizes the natural selection mechanism. The philosopher *Daniel Dennett* great defender of Darwin's theories, presents in his book: *Darwin's Dangerous Idea: Evolution and the Meanings of Life* [7], a engineering vision of the evolution theory, being this one of the most influential works on the subject today.

Genetic algorithms are computational devices, based on biological evolutionary processes, designed to find optimum and sub-optimum solutions for computational problems. An usual application case of this tool, is in continuous function optimization, where the genetic algorithms can be viewed as a multi objective global optimization tool, that finds the function extreme points through a blind search mechanism, based on the evolution of previous solutions.

The optimization problem extends to several areas of the exact sciences, including the optimization of the ANNs parameters vector[1], also known as ANNs learning or training, because this process trains the network to recognize a certain pattern relating its inputs to its outputs.

That training usually occur through iterative optimization methods, based on the gradient descent of the ANN error surface, which is calculated through the *backpropagation* algorithm [14],[15]. Among the more efficient methods known today for ANNs training can be mentioned, the quasi-Newton method: *BFGS* [1] and the conjugated directions method: *Scaled Conjugate Gradient* [11],[12].

An alternative way to find desired the ANNs parameters vector, is the use of meta-heuristic methods, as the genetic algorithms. Several researchers are using those algorithms directly in the optimization of the error surface with respect with parameters vector[16],[13],[17]. There are others works using genetic algorithms not only to optimize the network with respect to its parameters vector, but also other parameters as its topology [4],[8].

In spite of genetic algorithms represent a tool of great computational power to train ANNs, it does not have the same efficiency as procedural optimization methods, that uses more information about the problem (first and/or second order information), and usually produce better results.

The present document, introduces a new optimization concept for using genetic algorithms for ANN training, where an sub-optimum gradient is used, and steps are taken in the direction of this gradient. This proposal makes use of the full exploratory capacity of the genetic algorithms, united with the efficiency of gradient descent methods, reaching very superior results to those obtained by both techniques when applied separately.

2 Neural Networks Training

Artificial Neural Networks are devices with the universal approach capacity, and are in general applied to assimilate mappings, using for this a chain of interconnected artificial neurons, that interact each other in a similar way of the natural neural information processing.

This devices have also a formal mathematical representation, given by a mapping with inputs and outputs, which are expressed as a nonlinear function of its input, as described following:

$$y = \mathcal{F}(x, \theta) \tag{1}$$

[1] In this work, is considered the ANN parameters vector, a vector $\omega \in R^N$ containing the values of all network weighs and bias

where θ is the ANN parameters vector.

A priori it is not possible to determine an appropriated parameters vector θ such that the network expresses correctly any desired mapping $y = \mathcal{H}(x)$, therefore it is necessary to train the ANN to find a parameters vector, which produces the desired behavior.

In general, is not possible to find the parameters vector analytically, than, the network learning process is iterative, and seeks to increase the network adaptation to the target mapping at each iteration (also known as epoc). To make possible this task, an error is defined for the network output, that expresses the difference between the current behavior and the desired one. A possible definition for the network error, which is adopted in this work, can be the following:

$$e = \left(\sum_i (\mathcal{H}_i(x) - \mathcal{F}_i(x, \theta))^2 \right) / n_o \qquad (2)$$

where $\mathcal{H}$ is the target mapping that one wants to assimilate by the ANN, and n_o is the number of neurons on the output layer.

This error is a functional and creates a smooth surface in the space R^N, where N is the dimension of the vector θ, therefore the process of ANN training can be seen as the minimization of the error surface with respect to its parameter: θ.

3 The Evolving Gradient

There are some works that try to optimize the weight vector of the ANN using a genetic algorithm. This process is in general more onerous from the computational cost[2] viewpoint, and shows poor results when compared to conventional optimization methods based on gradient descent.

The present document, presents an alternative solution inspired in the work of Chalmers *The Evolution of Learning: An Experiment in Genetic Connectionism*[5], that applied evolutionary processes to evolve the learning process itself.

In the EG: Evolutionary Gradient method, the genetic algorithms are not used to optimize the weights vector, but to optimize the process by evolving the gradient vector.

3.1 Codification

The genetic algorithm implemented in the proposed EG algorithm, uses a population of n_p individuals, with a real codification, described as follows:

- Chromosome: Vectors containing real values belonging to the space R^N
- Fitness: $\exp(-\alpha e(x, \theta - p))$ where α it is a parameter to be adjusted, and p is a chromosome.
- Selection: Roulette and Elitist.

[2] here computational cost of a procedure is understood, the number of sum and multiplication operations necessary for accomplish this

– Reproduction: Matrix method [2] designed with based on the subspaces generated by parents chromosomes.
– Mutation: A Gaussian mutation modifying all components of the chromosome by adding a random noise.

In this code n_{rep} pairs are chosen for reproduction through a roulette mechanism, where $n_{rep} \in [1, n_p]$ is a random number. The remaining of the population $n_p - n_{rep}$ is chosen with an elitist selection procedure, to avoid loosing of the best solution and also to preserve the population diversity.

3.2 The EG Algorithm

This algorithm is based on gradient descent algorithm, where steps are taken in the direction of the gradient vector, however, here steps are taken in the direction of a sub-optimum gradient evolved by the genetic algorithm previously discussed.

To each iteration of the algorithm, an initial gradient vector is calculated using the *Backpropagation* method, or simple taken equals to the origin. Later a new population is created, and this gradient is introduced in the population, that will be evolved by the genetic algorithm during n_i generations, and then, an unit step is taken in this new evolved direction.

The use of the initial gradient seeks to accelerate the convergence of the evolutionary process, giving to him a reference point. It is also possible the method execution calculation of the initial gradient, as can be seen in [3]. This procedure is useful when it is not possible to accomplish the ANN retro-propagation phase, impeding the gradient vector construction, what enlarge the application range of the EG.

The EG algorithm for ANNs training is describe following.

Algorithm 1: Evolving Gradient.

```
Determine: $n_p$, $n_i$, $n_{max}$;
Initialize: $\theta$;
for $i$=1 to $n_{max}$ do
    Calculate $g_0$ by the retro-propagation of the error, or just set $g_0 = 0$;
    Evolve the gradient: $g = AG(n_p, n_i, g_0)$ ;
    Unit step: $\theta_{i+1} = \theta_i - grad$
end
```

where AG represents a genetic algorithm.

The algorithm 1, is the result a exhaustive study of the training process, and the functional analysis of the relations between the quantities of interest, taking into account the dimensionality of the involved spaces and the characteristics a priori known about the problem.

These studies converged to a method where the genetic algorithm is applied in a space of same dimension of the parameters vector. However, this choice brings a great advantage with respect to the cost function to be optimized.

In the application of a blind search method, as in case, the efficiency of the optimization process may be considerably increased if the method is initialized near the neighborhood of a local optimum point, which represents a good solution to the problem. However, this points are not known a priori, nor their neighborhoods.

The same happens in the ANNs training, the error surface are not complete known, so nothing can be done to increase the training algorithm efficiency. Moreover, it is known that a sufficient small step in a descent direction is ways a minimizing step. Therefore, we conclude that a R^N vector, representing a descent direction, will be certainly found in a neighborhood of the origin.

This information is the main goal of the EG, given to blind search method what it needs, a good initial condition. This approach gives to genetic algorithm a considerable efficiency increase, so the algorithm EG here proposed, presents a significantly higher convergence rate when compared with others meta-heuristic methods in the same context.

Another important feature of the EG, is the intrinsic parallelism of the algorithm. Its implementation in a sequential machine, as a digital computer, will generates a process computationally very onerous, however, in a completely parallel machine, still hypothetical, is obtained a very faster and efficient process.

4 Examples and Comparisons

With the intention of determining the relative efficiency of the EG compared with others founded in the literature, some ANN application examples are used, and the efficiency of the learning process is evaluated when the network is trained by different methods.

In this document will be considered, as comparison bases, two quasi-Newton methods, two conjugated directions methods and one simple gradient descent method, all described as follows:

- GRAD: Optimium Gradient [10]: Gradient descent method with super-linear convergence, the fastest among methods with linear convergence rate.
- DFP: Davidson Fletcher Powell [10]: quasi-Newton Method with quadratic convergence.
- BFGS:Broyden Fletcher Goldfarb Shanno [1]: quasi-Newton Method with quadratic convergence, and with smaller sensibility to the bad numerical conditioning than the DFP method.
- FR: Fletcher Reeves [1]: Conjugated directions method, with N-steps quadratic convergence.
- SCG: Scaled Conjugated Gradient[12]: Conjugated directions method that do not use unidimensional searches. It possesses N-steps quadratic convergence, and it is the fastest among these methods from the computational cost viewpoint.

The process of unidimensional search used in the algorithms GRAD, DFP, BFGS and FR is the golden section method, applied by 30 iterations on the initial interval.

4.1 Motor Currents

In theory, the current of a three-phase induction motor can be easily calculated on the basis of motor voltage and power, as shown in equation (3).

$$I = \frac{P}{\sqrt{3}V\eta} \tag{3}$$

where P and V represent the motor power and tension respectively. The variable η takes into account the power factor and efficiency of the motor, that are based on construction factors, the mechanical load and the rotation of the motor. Thus, it is clear difficult to specify the variable η and so, the motor current.

The problem in question uses a neural estimator for the current calculation, based on motor power, voltage and rotation, through a MLP network containing 3 neurons in its sensorial layer, and with 1 neuron in its output layer.

The set used in the training consist on 300 samples obtained from manufacturers catalogs, including motors that meet the following values ranges:

- Power: 0.1 a 330 KW.
- Rotation: 600,900,1200,1800 e 3600 rpm.
- Tension: 220, 380 e 440 V.
- Current: 0.3 a 580 A.

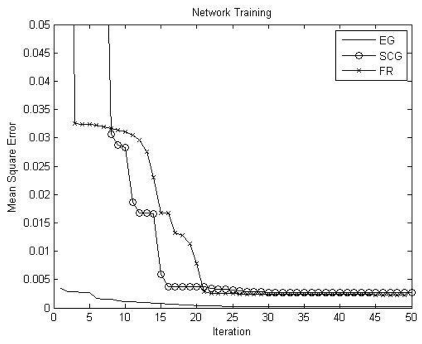


Fig. 1. EG SCG FR.

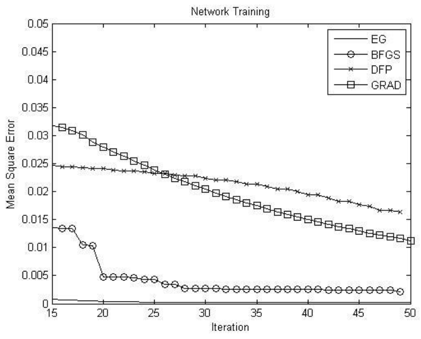


Fig. 2. EG BFGS DFP GRAD.

In a first test, seeking to compare the efficiency of the EG compared to the previously mentioned methods, a ANN containing 3 neurons in its hidden layer was used, having the configuration: 3-3-1. The result of the network training can be visualized in the Fig.1 and Fig.2.

Given the stochastic characteristic of the EG, these figures show its average behavior for a total of 20 repetitions of the training process. For this example the EG presented quite superior results when compared with the procedural methods.

Another important analysis is to compare the result of the ANN training by evolving gradient the method with the using the genetic algorithm directly for obtaining the optimum vector θ by minimizing the error surface.

For accomplish this analysis, let us consider a genetic algorithm, GA, with the same number of individuals of the EG. This algorithm is applied to search for parameters vector through 100 iterations while the EG algorithm had accomplished 5 epoc with 20 iterations per epoc, what gives for both methods the same number of iterations. The optimization process was repeated 20 times, and the average values for the ANN error as shown in Figure 3.

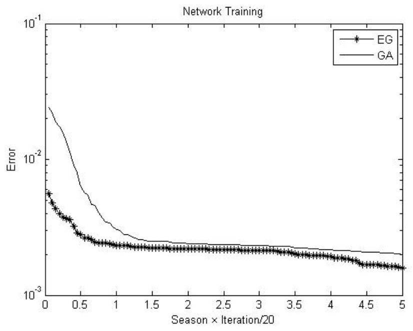


Fig. 3. Network Learning: EG $\times$ GA.

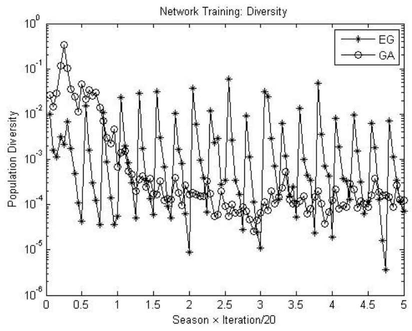


Fig. 4. Network Learning Diversity.

Figure 3 shows the superiors results obtained by the EG.

An analysis also relevant in this study, is to verify the populations diversity in both methods, what together with the results above, gives an more accurate understanding about the search mechanism. The metric here chosen to measure the diversity value, is the variance of the individuals fitness. Figure (5) shows the average value of diversity for both methods in each iteration[3] of the training process.

Is clearly in Figure 3 the superior diversity preservation present by the EG. Differing from the GA algorithm, the EG one do not present a significant lost of diversity after some iterations.

In spite of to providing a significant reduction of the network mean square error at each iteration, the EG is quite onerous from the computational cost viewpoint, given that an evolution process should be completed at each epoc. In that way, the execution of the algorithm may become too *slow*, depending of its configuration.

In [12], the author proves the superiority of the method SCG about the other methods here analyzed, due to the need of unidimensional search required by most methods, that has computational cost $O(N^2)$ per iteration. The SCG method presents a computational cost: $O(2N^2)$ per iteration, what is very inferior to the ones of the methods GRAD, DFP, BFGS and FR that possess computational costs:[4] $O(31N^2)$.

The EG method, presents a total cost $O(n_i n_p N^2)$ (using the same analysis of [12]), that can become quite superior to the ones of the other methods depending on the choice of n_i and n_p. However, the fast convergence of the method, compensate this high computational cost. Figure 5 illustrates a simple comparison of the temporary evolution of the SGC method with the evolving gradient method, for the network configuration: 3-9-9-1 containing two hidden layers. For the SCG method 500 iterations were made while

[3] Here iteration is used to describe the intermediate steps of training processes, and one iteration of main process, of the EG, is referred as epoc

[4] This value is due to the fact that the unidimensional search to makes 30 iterations for each iteration make for the training method

for the EG method only 5 iterations were made, to compensate the difference between the computational costs.

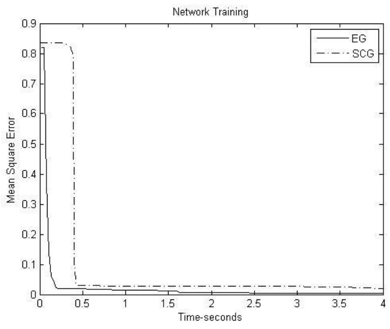


Fig. 5. Network Learning.

The EG, in spite of being computationally onerous, has a quite fast convergence, and in some cases this characteristic compensates its high computational cost, as in the situation shown in the Figure 5. Due the stochastic characteristic present in the EG and also in the training initialization, is not possible to conclude that the EG algorithm is faster than the SCG method in all cases, however, the result in Figure5 let us to state that the EG method is competitive with the other methods, from the learning efficiency viewpoint.

4.2 Curve Fitting

In this example a group of 100 test samples of input-output pairs was used for a quadratic function $y = x^2$, where a white noise of width 10^{-4} was inserted in both signs (input and output). The training was accomplished for various networks configurations, using the EG and the algorithms previously mentioned. The results can be observed in Tables 1 and 2.

Table 1. Learning Results: EG SCG FR.

Architecture	EG	SCG	FR
1-3-1	0.00019	0.00066	0.00030
1-6-1	0.00022	0.00069	0.00031
1-9-1	0.00020	0.00078	0.00033
1-18-1	0.00024	0.00082	0.00035
1-6-6-1	0.00029	0.01065	0.01395
1-12-12-1	0.00019	0.01135	0.01735
1-6-12-6-1	0.00023	0.00667	0.05779
1-12-18-12-1	0.00014	0.01764	0.06142

Table 2. Learning Results: BFGS DFP GRAD.

Architecture	BFGS	DPF	GRAD
1-3-1	0.00032	0.00079	0.01012
1-6-1	0.00037	0.00529	0.01197
1-9-1	0.00039	0.00617	0.01207
1-18-1	0.00045	0.00657	0.01287
1-6-6-1	0.01161	0.00959	0.01342
1-12-12-1	0.00943	0.00993	0.01377
1-6-12-6-1	0.00070	0.01060	0.01398
1-12-18-12-1	0.00420	0.01254	0.01485

For this problem it is also possible to notice that the final errors for the EG, was also quite inferior to the ones of the others tested algorithms. Another outstanding characteristic observed in the exposed results is the robustness of the EG method with respect to variations in the ANN topology. Due the stochasticity of the learning process, it is possible to infer that the EG method has presented the same final errors results for the several tested configurations.

5 Conclusions

The method proposed in this paper represents a new approach for MLP artificial ANNs training using meta-heuristic methods, presenting some new features with respect to others similar methods.

The use of genetic algorithms in ANN training was until now, not competitive given the inferior performance of these methods when compared to procedural optimization techniques. This new approach, however, is competitive in this scenery, reaching results comparable with the ones of the usual methods of ANN training, and still preserving some characteristics of the heuristic methods.

One of the main advantages of the evolving gradient method, is the possibility to train ANNs with the same efficiency of methods as BFGS and SCG, without the error gradient, enlarging its application to another several problems, as the one proposed in [3],[2].

The high computational cost, characteristic of heuristic methods as the genetic algorithms, also is present in the EG that is more onerous that the other methods here discussed. However, it is clear in the shown examples, that this high computational cost is compensated by the accelerated convergence rate of EG method, turning its temporary evolution, comparable to any one of the others training algorithms here discussed.

Moreover, the computational cost here analysed is related to a digital machine, which is the tool today available, however, the intrinsic parallelism of the EG, allows its an implementation in a hypothetical parallel machine, that can be several orders of magnitude faster than the procedural methods here discussed, inherently sequential.

So we may conclude that the Evolutionary Gradient method here presented, represents a viable alternative solution for artificial ANNs training in several situations, especially in more complex applications, mainly when the construction of the gradient vector is difficulty or even impossible.

References

1. R. Battiti and F. Masulli. Bfgs optimization for faster and automated supervised learning. *INNC 90 Paris, International Neural Network Conference,*, pages 757–760, 1990.
2. Csar Dalto Berci. *Observadores Inteligentes de Estado: Propostas.* Tese de Mestrado, LCSI/FEEC/UNICAMP, Campinas, Brasil, 2008.
3. Csar Dalto Berci and Celso Pascoli Bottura. Observador inteligente adaptativo neural no baseado em modelo para sistemas no lineares. *Proceedings of 7th Brazilian Conference on Dynamics, Control and Applications. Presidente Prudente, Brasil*, 7:209–215, 2008.
4. Jürgen Branke. Evolutionary algorithms for neural network design and training. In *1st Nordic Workshop on Genetic Algorithms and its Applications*, 1995. Vaasa, Finland, January 1995.
5. D.J. Chalmers. The evolution of learning: An experiment in genetic connectionism. *Proceedings of the 1990 Connectionist Summer School*, pages 81–90, 1990.
6. Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* John Murray, London, 1859.
7. D.C. Dennett. *Darwin's Dangerous Idea: Evolution and the Meanings of Life.* Penguim Books, 1995.
8. A. Fiszelew, P. Britos, A. Ochoa, H. Merlino, E. Fernndez, and R. Garca-Martnez. Finding optimal neural network architecture using genetic algorithms. *Software & Knowledge Engineering Center. Buenos Aires Institute of Technology. Intelligent Systems Laboratory. School of Engineering. University of Buenos Aires.*, 2004.
9. C. Fyfe. *Artificial Neural Networks.* Department of Computing and Information Systems, The University of Paisley, Edition 1.1, 1996.
10. D.G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, 2nd edition, 1984.
11. M.F. Mller. Learning by conjugate gradients. *The 6th International Meeting of Young Computer Scientists*, 1990.
12. M.F. Mller. A scaled conjugate gradient algorithm for fast supervised learning. *Computer Science Department, University of Aarhus Denmark*, 6:525–533, 1990.
13. D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 762–767, 1989.
14. D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin. *Backpropagation: The basic theory.* Lawrence Erlbaum Associates, Inc., 1995.
15. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation, in: Parallel distributed processing: Exploration in the microstructure of cognition. *Eds. D.E. Rumelhart, J.L. McClelland, MIT Press, Cambridge, MA.*, pages 318–362, 1986.
16. Udo Seiffert. Multiple layer perceptron training using genetic algorithms. *ESANN'2001 proceedings - European Symposium on Artificial Neural Networks*, pages 159–164, 2001.
17. Zhi-Hua Zhou, Jian-Xin Wu, Yuan Jiang, and Shi-Fu Chen. Genetic algorithm based selective neural network ensemble. *Proceedings of the 17th International Joint Conference on Artificial Intelligence.*, 2:797–802, 2001.

Topologies and Meaning Generating Capacities of Neural Networks

Jürgen Klüver and Christina Klüver

Department of Economy - COBASC Research Group, University of Duisburg-Essen
Universitätsstr. 12, 45117 Essen, Germany
{juergen.kluever,c.stoica-kluever}@uni-due.de

Abstract. The paper introduces the concept of meaning generating capacity (MC) of neural nets, i.e. a measure of information processing, depending on the size of basins of attraction. It can be shown that there is a significant relation between the variance values of the weight matrix of a network and its MC-values. By the concept of MC network characteristics like robustness and generalizing capability can be explained.

1 Introduction

The analysis of topological characteristics of complex dynamical systems frequently enables important insights into the behavior, i.e. the dynamics of such systems. By "topology" we here mean that set of system's rules that determine, which elements of the respective systems interact with which other elements. In the classical mathematical meaning of topology these rules define the neighborhood relations of the respective elements, which are at the core of, e.g., the fundamental Hausdorff axioms of topology. In the case of neural networks the topology is usually defined by the according weight matrix, which determines the degree of interaction between the different elements, including the limiting case of interaction degree equal to zero.

In [1] we introduced the concept of the *meaning processing capacity* (MC) of a complex dynamical system. This definition was motivated by some informal remarks of Wolfram [3] about the "information processing capacity" of complex dynamical systems. With this term Wolfram described the fact that frequently different initial states of a system generate the same final attractor state; other systems in contrast generate different final states if the initial states are different. In other words, the information processing capacity refers to the different sizes of the "basins of attraction" of a system, i.e. the sets of initial states that generate the same final attractor state.

In [1] we defined the concept of the "meaning" of a message by the final attractor state a system generates when receiving this message; in other words, a system processes a message and generates an according meaning. Therefore, we now use the term of *meaning generating capacity* (MC), i.e. the capacity to generate more or less different meanings when receiving different inputs.

The MC-value of a complex dynamical system is now defined as the proportion between the size m of the set of all final attractor states and the size n of the set of all initial states of a system, i.e., $MC = m/n$. Obviously $0 < MC \leq 1$: $MC = 0$ is impossible because each complex system has at least one final state, even if it is an attractor state with a very large period. The according limiting case hence is $MC = 1/n$. If MC is very small then many different initial states will generate the same final states – the according attractors are characterized by large basins of attraction. If $MC = 1$ then each different initial state will generate a different final attractor state. This is the other limiting case, where the basins of attraction all are of size 1. In other words, small values of MC mean large basins of attractions and vice versa. It must be noted that we refer only to discrete systems, i.e. systems with only a finite number of initial states.

There are at least three main reasons why this concept is important: On the one hand it is possible via the usage of MC to analyze complex dynamical systems like neural networks with respect to their informational complexity. In this sense MC allows for new approaches in the theory of computability. On the other hand an important and frequently mentioned characteristic of neural networks can be understood in a new and more differentiated way: In all textbooks on neural networks there are statements like "one of the main advantages of neural networks is their robustness, i.e. their tolerance with respect to faulty inputs" or something equivalent. We shall show that via the definition of MC not only a theoretical explanation of this advantage can be given but also a measurement of this robustness; in particular by the variation of MC specific neural networks can be generated that are either very robust, less robust or not at all robust in the sense of error tolerance.

Last but not least it is possible to give by the usage of MC an explanation for phenomena known from the field of human information processing. It is well known that different humans react in a significant different way to the same messages. This can be illustrated by the examples of fanatics who refer all messages to the same cause, e.g. the enmity of Western Capitalism to religious movements. The psychiatrist Sacks [2] for another example describes a rather intelligent and well-educated man who is unable to distinguish little children from fire hydrants. The definition of MC can be a useful approach to construct mathematical models for the explanation of such phenomena.

In contrast to dynamical systems like, e.g., cellular automata and Boolean networks neural networks are not often analyzed in terms of complex dynamical systems. Therefore, it is necessary to clarify what we understand by "initial states" and "final attractor states" when speaking of neural networks.

In a strict systems theoretical sense all initial states of neural networks are the same, i.e. the activation values of all neurons are equal to zero, and regardless to which layer(s) they belong. Because this fact would make the definition of different initial states quite useless we define the initial state of a neural net as the state where the neurons of the input layer have been externally activated with certain input values and where the activation values of all other neurons still are equal to zero, in particular those of the output layer. An initial state S_i of a neural net, hence, is formally defined by $S_i = ((A_i), (0))$, if (A_i) is the input vector and (0) is the output vector, i.e. it denotes the fact that the values of the output neurons are still equal to zero. If there is no specific input layer then the definition must be understood that some neurons are externally activated and the others are not.

The external activation of the input neurons causes via the different functions the "spread of information", determined by the respective weight values. In the case of simple feed forward networks the final activation values of an output layer are immediately generated; in the case of feed back networks or recurrent ones the output is generated in a more complex manner; yet in the end in all cases a certain output vector is generated, i.e., each neuron of the output layer, if there is any, has obtained a certain activation value. If there is no distinction between different layers as for example it is the case with a Hopfield network or an interactive network the output vector will consist of the final activation values of all neurons. Note that except in the case of feed forward networks the output vector may be an attractor with a period $p > 1$. The network will then oscillate between different vectors, i.e. between different states of the attractor. For theoretical and practical purposes neural networks are mainly analyzed with respect to the input-output relation. Therefore, we define the final state S_f of a neural network as $S_f = ((A_i), (A_f))$, if (A_i) is again the input vector and (A_f) the final output vector. If (A_f) is an attractor with period $p > 1$, then the components of (A_f) consists of ordered sets, i.e. the set of all different activation values the output neurons obtain in the attractor.

Because in the experiments described below we investigate only the behavior of feed forward networks with respect to different MC-values, for practical purposes we just define the final state as the values of the output vector after the external activation via the input vector. Hence we speak of a large basin of attraction if many different input vectors generate the same output vector and vice versa. The limiting case MC = 1 for example defines a network where each different input vector generates a different output vector. Accordingly the case M = 1/n defines a network where practically all n different input vectors generate the same output vector.

With these definitions it is easy to explain and measure in a formal manner the characteristics of neural networks with respect to robustness. A robust network, i.e. a network that is tolerant of faulty inputs, has necessarily a MC-value significantly smaller than 1. Robustness means that different inputs, i.e. inputs that differ from the correct one, still will generate the "correct" output, i.e. that output that is generated by the correct input. That is possible only if some faulty inputs belong to the same basin of attraction as the correct input; these and only these inputs from this basin of attraction will generate the correct output. All other faulty inputs transcend the limits of tolerance with respect to the correct output and will accordingly generate another output. If MC = 1 or near 1 then the network will not be robust at all for the respective reasons.

The same explanation can be given for the also frequently quoted capability of neural networks to "generalize": In a formal sense the generalizing capability is just the same as robustness, only looked upon from another perspective. A new input can be perceived as "similar" or as "nearly the same" as an input that the net has already learned if and only if the similar input belongs to the same basin of attraction as the input the network has been trained to remember. In other words, the training process with respect to a certain vector automatically is also a training process with respect to the elements of the according basin of attraction. The capability of generalization, hence, can be understood as the result of the construction of a certain basin of attraction. Accordingly the generalization capability is again dependent on the MC-values: if these are small, i.e. if the basins of attraction are rather large, then the network has a comparatively great generalizing capability and vice versa. Because one network can have only one MC-value it is obvious that systems like the human brain must have for one and the same perceiving task at least two different networks, namely one with a great generalization capability, i.e., a small MC-value, and one with a large MC-value to perceive different inputs as different.

Robustness and generalizing capability of a network, hence, can be "globally" defined by the according MC-value. Yet there is a *caveat*: it is always possible to generate networks via according training methods that are characterized by different basins of attractions with different sizes. Therefore, the MC-value is not necessarily a unique measure for the size of all basins of attraction of a particular network. The term "basin of attraction" refers always only to a certain "equivalence class" of input vectors, namely a set of input vectors that are equivalent in the sense that they generate the same attractor. The size of these sets may be quite different for specific attractors. Hence, the MC-value gives just an average measure with respect to the different basins of attraction. With respect to some attractors and their generating inputs the networks may be robust and with respect to others not. Considering that possibility the concept of MC could also be defined as the difference in size of all basins of attraction of the networks. Fortunately the results of our present experiments hint at the fact that in most cases the basins of attraction of a certain networks differ not much in size. The *caveat* is necessary for theoretical and methodical reasons but seems not to be very important in practical contexts.

Concepts like "size of basins of attraction" and "values of meaning generation capacity" obviously are very useful for the explanation of important characteristics like robustness or generalizing capability. Yet in a strict sense they are too general concepts because they only explain the behavior of certain neural networks from very general characteristics of complex dynamical systems. They do not explain, which *structural* characteristics of neural networks may be the reason for specific MC-values. Hence, these concepts remain, so to speak, on a phenomenological level.

In the beginning of our article we mentioned the fact that frequently certain topological characte-ristics of complex dynamical explain the behavior of such systems. The topology of a neural network is mainly expressed in the weight matrix. Hence the thought suggests itself to look for features of the weight matrix that could explain the size of basins of attraction and MC-values. In anticipation of our results we may say that we were successful in the sense that we found some general trends although no deterministic relations.

2 Two Experimental Series

In the first experimental analysis we used a standard three-layered feed forward network; we chose this type because it is very frequently used for tasks of pattern recognition and related problems. Because, as is well known, two layers are not enough to solve problems of non-linear separableness we took three layers in order to get results for networks with sufficient efficiency. The input layer consists of 10 units, the hidden layer of 5 and the output layer of 10 units. Input and output neurons are binary coded, which results in $2^{10} = 1024$ possible input patterns. To keep the experiments as clearly as possible we defined "equivalence classes" of input patterns:

all input patterns with the same number of zeroes are members of the same class. By choosing at random one pattern from each class we obtained 11 different input patterns. The activation function respectively is the sigmoid function; because of the three layers we chose as learning rule the standard Back Propagation rule.

The training design was the following: In each step the network was trained to associate different input patterns with one target pattern; the target pattern was again chosen at random from the 11 input patterns. In the first step the tasks was to associate each input pattern with one different target pattern; the according basins of attraction all were of size one and the MC-value of this network after the training process is 1:1. In the next steps the sizes of the basins of attraction were gradually increased to 2, 3, 4, 5, and 6; in the last step the size of the only basin of attraction finally was 11, i.e. all input layers had to be associated with one and the same target pattern and the according MC-value is MC = 1/11. We did not investigate basins of attraction with sizes 7 or 10 because in the according experiments the other basins would become too small; for example, one basin of attraction with the size of 8 would force the network to take into regard also at least one basin of attraction of size 3. Hence we only investigated networks with basins of attraction of maximum size 5, 6, and 11. By taking into regard different combinations of basins of attraction we obtained 11 different networks.

The according weight matrices were analyzed with respect to the variance of their weight values. This variance analysis was separately performed for the weight matrix between the input layer and the hidden layer and the matrix between the hidden layer and the output one. The results are shown in figure 1:

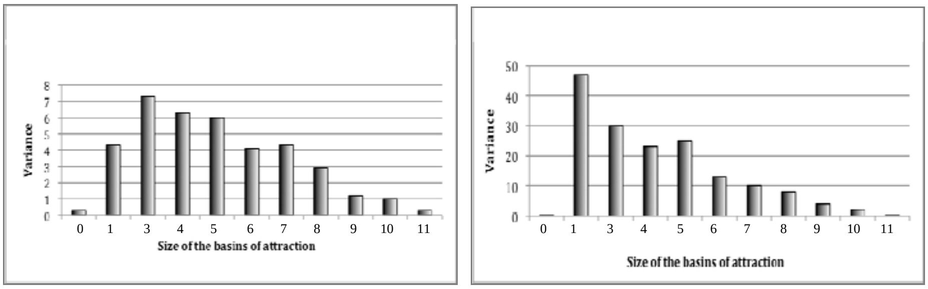


Fig. 1. Variance of the first part of matrix (left figure) and the second part (right figure) in relation to the size of the basins of attraction.

The order of the different networks in both figures is according to increasing size of the basins of attraction. No 1 is the case with MC = 1:1, no 11 is the network with MC = 1:11.

The left figure obviously gives no unambiguous result with respect to possible relations between variance values and the size of basins of attraction but it suggest a certain trend, namely the decreasing of the variance by increasing the basins sizes. The right figure confirms this and even shows an unambiguous result: The variance values indeed gradually decrease with the increasing of the size of the basins of attraction. We then combined the two matrices by summing up the variance values of both matrices and obtained the final result shown in figure 2:

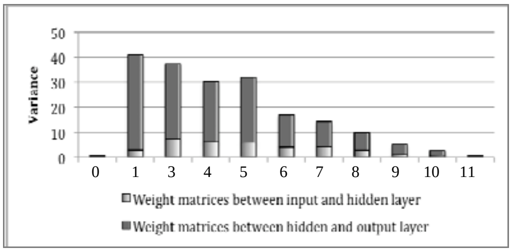


Fig. 2. Variance and size of basins of attraction for the whole network; the networks are ordered as in figure 1.

This final result completely confirms the trend shown in figure 1, left side, and the clear results of figure 1, right side: the larger the size of the basins of attraction are, i.e., the smaller the MC-values are, the smaller are the variance values and vice versa. By the way, the difference between the variance of the upper matrix and that of the lower one is probably due to the fact that the Back Propagation rule does not operate in exactly the same way on both matrices: The lower half of the whole matrix is changed by directly taking into account the error, i.e. the distance between the output neurons and those of the target vector. The changing of the upper half of the matrix is done by computing a certain proportion of the error and thus “dispersing” the changed weight values with respect to those of the lower half. Yet these just force the variance of the whole matrix to the shown result. If our networks had contained only two layers the whole result would have been like that of figure 2. We shall come back to this effect of a certain learning rule in the next section.

We did not expect such unambiguous results yet on hindsight they are quite plausible and comprehensible: Low variance values mean dispersion of information or of the differences between different information respectively because of the near equality of the weight values. If on the other hand the weight values are significantly different, i.e. a high variance, then differences between different messages can be preserved. As small or large sizes respectively of basins of attraction have exactly that effect on the performing of messages it is no wonder that we obtained that clear and unambiguous relation between variance and size of basins of attraction.

Yet although these clear results are quite satisfactory we knew very well that they must be treated with a great methodical *caveat*: the behaviour of neural networks, as that of practically all complex dynamical systems, depends on many parameters, in this case for example on specific propagation, activation and output functions, number of layers, special learning rules and so on. The results shown above were obtained with a specific type of neural network, although a standard and frequently used one with a standard learning rule. To make sure that our results are not only valid for this special methodical procedure we undertook another experimental series.

In these experiments we did not use one of the standard learning rules for neural networks but a Genetic Algorithm (GA). The combination of a GA with neural networks has frequently been done since the systematic analysis of neural networks in the eighties. Usually a GA or another evolutionary algorithm is used in addition to a certain learning rule in order to improve structural aspects of a network that are not changed by the learning rule, e.g. number of layers, number of neurons in a particular layer, threshold values and so on. In our experiments we used the GA as a substitute for a learning rule like the Back Propagation rule in the first experimental series. The according weight matrices of the different networks are, when using a GA, written as a vector and the GA operates on these vectors by the usual “genetic operators”, i.e. mutation and recombination (crossover).

We chose this procedure for two reasons: On the one hand the operational logic of a GA or any other evolutionary algorithm is very different from that of the standard learning rules. A learning rule modifies usually just one network; in this sense it is a simulation of ontogenetic learning. In contrast an evolutionary algorithm always operates on a certain *population* of objects and optimizes the single objects by selecting the best ones from this population at time t. This is a model of phylogenetic evolution. In addition learning rules like the Back Propagation rule or its simpler form, namely the Delta Rule, represent the type of supervised learning. Evolutionary algorithms represent another type of learning, i.e. the enforcing learning. In contrast to supervised learning enforcing learning systems get no feed back in form of numerical values that represent the size of the error. The systems just get the information if new results after an optimization step are better or worse than the old ones or if there is no change at all in the improvement process. Therefore, the training procedure in the second series is as different from that of the first one as one can imagine.

We assumed that by choosing such different procedures similar results from both experiments would be a very strong indicator for our working hypothesis, namely the relation between MC-values or size of the basins of attraction respectively and the mentioned characteristics of the according weight matrices. To be sure, that would not be a final proof but at least a “circumstantial evidence” that the results of the first series are no artifacts, i.e., that they are not only effects from the chosen procedure.

On the other hand we were in addition interested in the question if networks with certain MC-values are better or worse suited to adapt to changing environmental conditions. It is evident that *per se* high or low MC-values are not good or bad. It always depends on the situation if a network performs better with high or low capabilities to generate different meanings. Sometimes it is better to process a message in a rather general fashion and sometimes it is necessary to perceive even small differences. Yet from a perspective of evolutionary adaptation it is quite sensible to ask if systems with higher or lower MC can adjust better. That is why we used an evolutionary algorithm to investigate this problem although it is another question than that of a relation between the variance of the weight matrix and the according MC-values.

Because a GA can be constructed with using many different parameters like size of the mutation rate, size of the sub vectors in crossover, selection schemas, schemas of crossover (“wedding schemas”), keeping the best “parents” or not and so on it is rather difficult to obtain results that are representative for all possible GA-versions. We used a standard GA with a mutation rate of 10%, a population of 20 networks, initially generated at random, crossover segments of 5, and a selection schema according to the fitness of the respective networks. Because the networks were optimized with respect to the same association tasks as in the first series those networks are “fitter” than others that successfully have learned more association tasks than others. If for example a network is optimized with respect to the task to operate according to two basins of attraction of size 8 then a network is better that correctly associates 6 vectors of each basin to the target vector than a network that does this only for 5 vectors.

The population consists again of three-layered feed forward networks with binary coding for input and output layers; the input and output vectors consist of four neurons and the hidden layer of three. As in the first series the networks operate with the sigmoid activation function. We simplified the networks a bit because, as mentioned, a GA has not one network to operate with but a whole population. The target vectors were chosen at random; the vectors for the respective basins of attraction were chosen according to their Hamming distance to those output vectors that define the basins of attraction. It is no surprise that the GA came faster to satisfactory results, i.e. the generation of networks that are able to solve the respective association tasks, if the MC-values of the networks should be large than in the cases when the MC should be small. The main results are shown in figure 3:

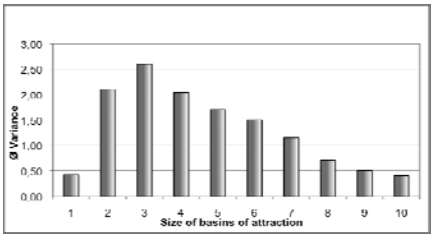


Fig. 3. Variance and size of basins of attraction in networks generated by a GA.

The figure obviously expresses a striking similarity to figure 1 of the first series. The trend is the same, namely a clear relation between the size of the variance and the increasing size of the basins of attraction or the decreasing size of the MC-values respectively. Like in figure 1 the exceptions from this trend occur in the cases of rather small basins of attraction, but only there. As we remarked in the preceding section these exceptions may be due to the fact that the GA even more disperses the weight values than does the Back Propagation rule for the upper half of the weight matrix. This fact clearly demonstrates that the relation between variance values and the sizes of the basins of attraction is “only” a statistical one, although the correlation is very clear. We omit for the sake of brevity he results of the evolutionary analysis.

As we mentioned in the beginning of this section, the fact that such totally different optimization algorithms like Back Propagation rule and GA, including the different types of learning, generate the same trend with respect to our working hypothesis is important evidence that the hypothesis may be valid in a general sense. Yet in both series we just considered “case studies”, i.e. we concentrated in both cases on one single network type and in the case of the GA-training on populations of the same type of networks. That is why we started a third series.

3 Third Series: Statistical Analysis of Large Samples

Experiments with large samples of neural networks are always difficult because of the large number of variables or parameters respectively that have to be taken into account. Besides the influence of different learning rules, activation and propagation functions and such parameters like learning rates and momentum the main problem is a "combinatorial explosion": if one takes into account the many different possible combinations of neurons in the different layers and in addition the possible variations of the number of layers one quickly gets such large samples that it is seldom possible to obtain meaningful results. That is why we chose another way in the preceding sections, namely the analysis of the two case studies in order to get a meaningful hypothesis at all.

Yet despite the great difference between our two case studies it is always rather problematic to draw general consequences from only several case studies. That is why we studied a larger sample of two-layered neural nets, i.e., ca. 400.000 different networks. We restricted the experiment to networks of two layers in order to keep the experiments as clear as possible. The number of neurons in the input and output vector are in all experiments the same and ranged from 3 to 10. The restriction to equal dimensions of the two vectors was introduced because networks with different sizes of the two vectors do not generate all MC-values with the same probability: If the input vector is larger than the output one then MC = 1 would not be possible at all because always more than one input vector will generate the same output vector. For example, a simple network that is trained to learn a certain Boolean function has an input vector of size 2 and an output vector of size 1. Its MC-value is 0.5. If conversely the output vector is larger than the input vector the probability for large MC-values will be greater than in networks with the same number of neurons in both vectors. To avoid such distortions we used only vectors of equal size.

The networks were, as in the two case studies, binary coded and operated with the sigmoid function. Thus we obtained ca. 400.000 pairs (MC, v), v being the variance. The general results are the following:

As we supposed from the results of the two case studies the relation between variance and MC-values is "only" a statistical one in the sense that there are always exceptions from the general rule. Yet we discovered very clearly that indeed there is a significant probability: the larger the variance is the smaller is the probability to obtain networks with small MC-values, that is with large basins of attraction, and vice versa. This result is in particular valid for variance values significantly large or small. Only in the "middle regions" of variance values the probability to obtain MC-values as a deviation from the general rule is a bit larger but not very much. This probability distribution is shown in figure 4:

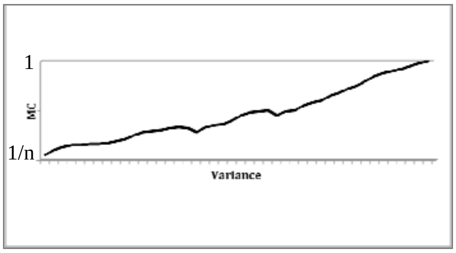


Fig. 4. Statistical relation between variance (x-axis) and MC-values (y-axis).

By the way, the deviations in the middle regions from the general trend may be a first explanation for the mentioned results from section 2 with respect to the evolutionary capability of networks with different MC-values. These networks adapt more easily to changing environments than those with very large or very small values.

The hypothesis of the relation between MC-values and variance values seems to be a valid one, at last as a statistical relation, Hence it is possible to predict the meaning generating capacity of a network and thus its practical behaviour for certain purposes with sufficient probability from a variance analysis of its weight matrix. Yet a *caveat* is in order: We analyzed only one type of networks and further experiments are necessary if our results are also valid for different types like, e.g. recurrent nets of Self Organized Maps.

4 Interpretations and Conclusions

The behavior of complex dynamical systems can practically never be explained or predicted by using only one numerical value (a scalar) as the decisive parameter. In a mathematical sense the problem for such a system is always the task of solving equations with a lot of variables, that is more variables than equations. It is well known that for such tasks there is always more than one solution. When considering neural networks by investigating the according weight matrix it is rather evident that for example large basins of attraction may be constructed by very different matrices. Hence, it is no wonder that the variance value, considered as a structural measure for the occurrence of certain MC-values and the according sizes of the basins of attraction, always allows for exceptions.

The knowledge about parameters that could predict the meaning generation capacity would not only give important insights into the logic of neural network operations; that would be an improvement of our theoretical understanding of these complex dynamical systems. It could also give practical advantages if one needs neural networks with certain capabilities – either if one needs robust networks with great generalizing capacity or if there is need for sensitive networks that react in a different manner to different inputs. To be sure, the relations we have discovered so far are of only statistical validity. Yet to know that with a high probability one gets the desired characteristics of a certain network if one constructs a weight matrix with a specific variance is frequently a practical advantage: one has not to train the networks and look afterwards if the network has the desired behavior but can construct the desired matrix and if necessary make the additionally needed improvements via learning rules and/or additional optimization algorithms like for example a GA. For these theoretical and practical reasons it will be worthwhile to investigate such relations as we have shown in this article further and deeper.

References

1. Klüver, J. and Klüver, C., 2007: On Communication. An Interdisciplinary and Mathematical Approach. Dordrecht (NL): Springer.
2. Sacks, O., 1985: The Man Who Mistook His Wife for a Hat. New York: Summit Books.
3. Wolfram, S., 2001: A new Kind of Science. Champagne (Ill.): Wolfram Media.

Models for Modular Neural Networks: A Comparison Study

Eva Volna

University of Ostrava
30th Dubna St. 22, 70103 Ostrava, Czech Republic
eva.volna@osu.cz

Abstract. There are mainly two approaches for machine learning: symbolic and sub-symbolic. Decision tree is a typical model for symbolic learning, and neural network is a model for sub-symbolic learning. For pattern recognition, decision trees are more efficient than neural networks for two reasons. First, the computations in making decisions are simpler. Second, important features can be selected automatically during the design process. This paper introduces models for modular neural network that are a neural network tree where each node being an expert neural network and modular neural architecture where interconnections between modules are reduced. In this paper, we will study adaptation processes of neural network trees, modular neural network and conventional neural network. Then, we will compare all these adaptation processes during experimental work with the Fisher's Iris data set that is the bench test database from the area of machine learning. Experimental results with a recognition problem show that both models (e.g. neural network tree and modular neural network) have better adaptation results than conventional multilayer neural network architecture but the time complexity for trained neural network trees increases exponentially with the number of inputs.

1 Introduction

There are mainly two approaches for machine learning. One is symbolic approach and another is sub-symbolic approach. Decision tree is a typical model for symbolic learning, and neural network is a model for subsymbolic learning. Generally speaking, symbolic approaches are good for producing comprehensible rules, but not good for incremental learning. Sub-symbolic approaches, on the other hand, are good for on-line incremental learning, but cannot provide comprehensible rules. Neural network tree is a hybrid-learning model. It is a decision tree with each non-terminal node being an expert neural network. Neural network tree is a learning model that may combine the advantages of both decision tree and neural network. To make the neural network tree model practically useful, we should make the following:

– to propose an efficient algorithm for incremental learning,
– to produce neural network trees as small as possible,
– to provide a method for on-line interpretation.

The first topic is studied in [1], the second topic is studied in [2], and the third topic is studied in [3].

A modular neural network can be characterized by a series of independent neural networks moderated by some intermediary. Each independent neural network serves as a module and operates on separate inputs to accomplish some subtask of the task the network hopes to perform [8]. The intermediary takes the outputs of each module and processes them to produce the output of the network as a whole. The intermediary only accepts the modules' outputs - it does not respond to, nor otherwise signal, the modules. As well, the modules do not interact with each other.

A multilayer artificial neural network is a net with one or more layers (or levels) of nodes (hidden units) between input units and the output units. They are often trained by backpropagation algorithm [9].

Next, we introduce all mentioned models in details. We will study adaptation processes of neural network trees, modular neural network and conventional neural network. Then, we will compare all these adaptation processes during experimental work with the Fisher's Iris data set that is the bench test database from the area of machine learning. Experimental results with a recognition problem show that both models (e.g. neural network tree and modular neural network) have better adaptation results than conventional multilayer neural network architecture but the time complexity for trained neural network trees increases exponentially with the number of inputs.

2 Binary Decision Tree

Since all kind of decision trees can be reduced to binary decision trees, then we can consider binary decision trees only. A binary decision tree can be defined as a list of 7-tuples. Each 7-tuple corresponds to a node. There are two kinds of nodes: *non-terminal node* and *terminal node*. Specifically, a node is defined by *node* = {*t, label, P, L, R, C, size*}, where

t	is the node number. The node with number $t = 0$ is called *root*.
label	is the class label of a terminal node, and it is meaningful only for terminal nodes.
P	is a pointer to the parent (for the root, $P = NULL$).
L, R	are the pointers to the left and the right children, respectively. For a terminal node, both pointers are *NULL*.
C	is a set of registers. For a non-terminal node, $n = C[0]$ and $a = C[1]$, and the classification is made using the following comparison: $feature_n < a$? If the result is YES, visit the left child; otherwise, visit the right child. For a terminal node, C[i] is the number of training samples of the *i-th* class, which are classified to this node. The label of a terminal node is determined by majority voting. That is, if $C[k] = \max_{\forall i} C[i]$, then, $label = k$.
size	is the size of the node when it is considered as a *sub-tree*. This parameter is useful for finding the fitness of a tree. The size of the root is the size of the whole tree, and the size of a terminal node is 1.

Many results have been obtained for construction of binary decision trees [4]. To construct a binary decision trees, it is assumed that a training set consisting of feature vectors and their corresponding class labels are available. The binary decision tree is then constructed by recursively partitioning the feature space in such a way as to recursively generate the tree. This procedure involves three steps: splitting nodes, determining which nodes are terminal nodes, and assigning class labels to terminal nodes. Among them, the most important and most time consuming step is splitting the nodes.

3 Neural Network Tree

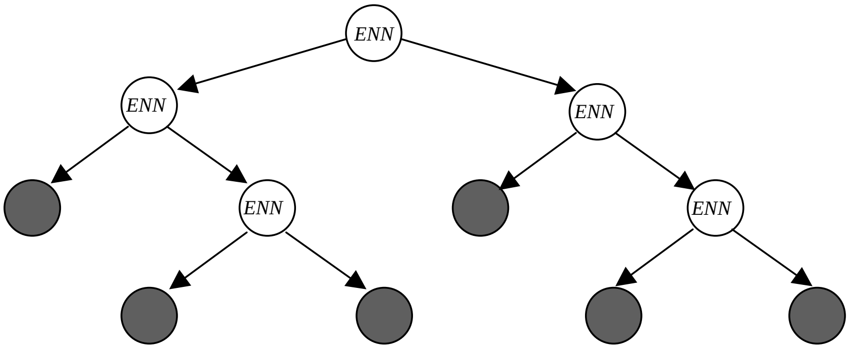


Fig. 1. A neural network tree.

Figure 1 shows an example of neural network trees. A neural network tree is a decision tree, which each non-terminal (internal) node being an expert neural network. We can consider a neural network tree as a modular neural network [5]. To design a neural network tree, we can use the same recursive process as that is used in conventional algorithms [6]. The only thing to do is to embed some algorithm in this process to design each expert neural network, e.g. a simple genetic algorithm into C4.5 [6] algorithm. To simplify the problem, we have two assumptions: 1) The architecture of all expert neural networks are the same (a multilayer perceptron with the same number of layers and the same number of neurons in each layer) that are pre-specified by the user. 2) Each expert neural network has n branches, with $n \geq 2$. *First*, by fixing the architecture of all expert neural networks, we can greatly restrict the problem space for finding an expert neural network for each node. *Second*, we allow multiple branches because each expert neural network is not only a feature extractor, but also a local decision maker. An expert neural network can extract complex features from the given input vector, and then assign the example to one of the *n* groups. To design the expert neural networks, the efficient way seems to be evolutionary algorithms, because we do not know in advance which example should be assigned to which group. The only thing we can do is to choose one expert neural network, to optimize some criterion. For this purpose, we can use a simple genetic algorithm containing merely basic operations: one-point crossover and bit-by-bit mutation. The genotype of a multilayer perceptron is the concatenation of all weight vectors (including the threshold values) represented by binary numbers. The definition of the fitness is domain dependent. The fitness can be defined as the information gain ratio that is used as the criterion for splitting nodes. The basic idea is to partition the current training set in such a way that the average information required to classify. For detailed discussion, refer to [6].

We used the top-down method [2] to design a neural network tree. In this method, we divide the whole training set into two parts first and classify them into two categories: patterns which belong to the left-nodes, and those which belong to the right-nodes. Figure 2 illustrates this idea. In this figure, a circle is a node, a triangle is a sub-tree. The sub-trees are designed separately using genetic algorithm for classifying some of the patterns in the training set. Suppose that there are *n* patterns in the *i-th* class, n_1 patterns are classified to the left-nodes, and $n_2 = n - n_1$ patterns are classified to the right-nodes. To evaluate a sub-tree, we provide all training patterns (in the current training set) to the tree, assign each pattern to a proper category (left or right), and then count the number of correct classifications. Then, the fitness can be defined by

$$fitness = 1 - \frac{number\_of\_misclassififation}{size\_of\_training\_set}. \tag{1}$$

This process is repeated until the current training set contains only patterns of the same class. Size of the whole decision tree is roughly the sum of size of all sub-trees obtained in the recursive procedure.

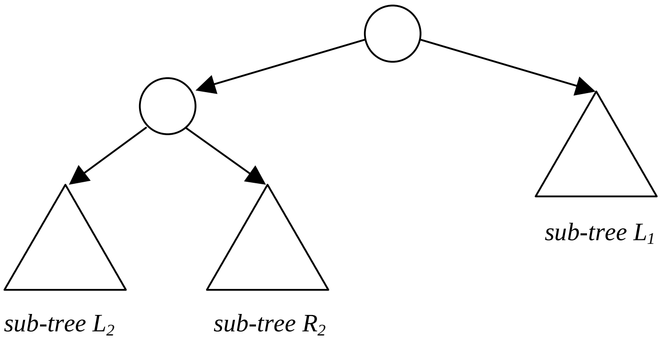


Fig. 2. Divide a tree into many sub-trees.

The neural network tree works as follows. The input is given to the root node first. It is then assigned to the *i-th* child if the *i-th* output of the module is the maximum. If the child is a leaf, the final result is produced locally; otherwise, repeat the same produce recursively. The most notable feature of a neural network tree is that it consists of homogenous neural networks that can be realized using exactly the same functional components (with different parameters), and the whole system can be constructed hierarchically. From such neural network tree, we can easily get an autonomous modular neural network [5]. For example, if the root is a multilayer perceptron with *n* outputs, it can be split into *n* subnets. These subnets can be used to work together with the children. The autonomous modular neural network works like this: for a given input task, each module tries to give an output y along with a number c. If c_i is the maximum, the output of the *i-th* module will be used as the final result. Each module can be an autonomous modular neural network again, which is obtained from the neural network tree by using the same procedure recursively. The basic idea is to design small expert neural networks to extract certain features (and make local decision based on the features) first, and the overall decision can be made by the whole decision tree.

4 Modular Neural Network

Several characteristics of modular architectures suggest that they should learn faster than networks with complete sets of connections between adjacent layers. One such characteristic is that modular architectures can take advantage of function decomposition. If there is a natural way to decompose a complex function into a set of simpler functions, then a modular architecture should be able to learn the set of simpler functions faster than a monolithic multilayer network. In addition to their ability to take advantage of function decomposition, modular architectures can be designed to reduce the presence of conflicting training information that tends to retard learning. We refer to conflicts in training information as crosstalk and distinguish between spatial and temporal crosstalk. Spatial crosstalk occurs when the output units of network provide conflicting error information to a hidden unit. This occurs when the back-propagation algorithm is applied to a monolithic network containing a hidden unit that projects to two or more output units.

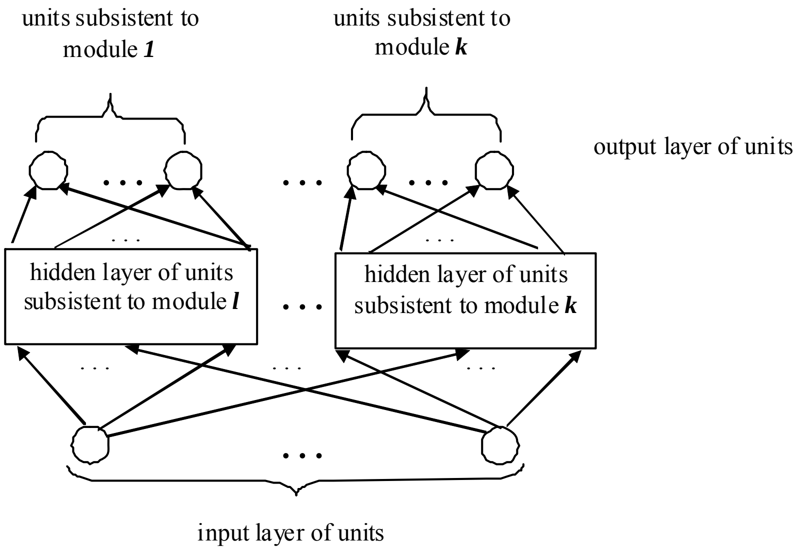

units subsistent to module *1*
units subsistent to module *k*
output layer of units
hidden layer of units subsistent to module *l*
hidden layer of units subsistent to module *k*
input layer of units

Fig. 3. The modular neural network architecture.

A modular architecture, which should generalize better than a monolithic network, involves the difference between local and global generalization. Modular architectures perform local generalization in the sense of the architecture that only learns patterns from a limited region of the input space. Therefore training a modular architecture on a training pattern from one of these regions should not ideally affect the architecture's performance on pattern from the other regions. Modular architectures tend to develop representations that are more easily interpreted than the representations developed by single networks. As a result of learning, the hidden units of the system used in separate networks for the tasks contribute to the solutions of these tasks in more understandable ways that the hidden units of the single network applied to both tasks. In modular networks, a different set of hidden units is used to represent information about the different tasks, see figure 3.

5 Experiments

During our experimental work, we made a very easy comparative adaptation study. In order to test the efficiency of described algorithms, we applied them to *the Fisher's Iris data set* [7] that is the bench test database from the area of machine learning. The Fisher's Iris data set is a multivariate data set introduced by Sir Ronald Aylmer Fisher. The dataset consists of 150 samples from each of three species of Iris flowers (*Iris setosa*, *Iris virginica*, and *Iris versicolor*). Four features were measured from each sample; they are the length and the width of sepal and petal. Based on the combination of the four features, Fisher developed a linear discriminated model to determine which species they are. We used 100 examples for training and the remainder for testing.

Neural Network Tree. Expert neural networks are multilayer perceptrons of the same size 4 - 2 - 2, where four neurons are in the input layer, two neurons are in the hidden layer, and two neurons are in the output layer. For any given examples it is assigned to the *i-th* subset, if the *i-th* output neuron has the largest value (when this example is used as input). All nets are fully connected. To find such expert neural network for each node, we adopt the genetic algorithm, which has the following parameters: number of generation is 1000, the population size is 30, selection rate is 0.9 (e.g. 90% of individuals with low fitness values are exchanged in each generation), crossover rate is 0.7, and mutation rate is 0.01. The number of bits per weight is 16. The fitness is defined directly as the gain ratio. The desired fitness is 0.9. The maximum fitness is 1.0 from its definition. All individuals are sorted according to their priority ranks, and the worst $p \times N$ individuals are simply deleted from the population, where p is the selection rate, and N is the population size. In each experiment, we first extract a subnet from the whole training set, and use it for designing (training) a neural network tree. Of course, we count the number of neurons contained in the whole tree.

Modular Neural Network. We used a three-layer feedforward neural network with architecture is 4 - 6 - 3 (e.g. four neurons in the input layer, six neurons in the hidden layer, and three neurons in the output layer) in our experimental work. Each module has two neurons in a hidden layer and one output neuron. The input layer and the hidden layer are fully interconnected. The input values from the training set were transformed into interval <0; 1> to be use backpropagation algorithms for adaptation. The backpropagation adaptation deals with the following parameters: learning rate is 0.3, and the moment parameter was not used.

Multilayer Neural Network. We used a fully connected three-layer feedforward neural network with architecture is 4 - 4 - 3 (e.g. four units in the input layer, four units in the hidden layer, and three units in the output layer) in our experimental work, because the Fisher's Iris data set [7] is not linearly separable and therefore we cannot use neural network without hidden units. The backpropagation adaptation deals with the same parameters like in the previous model.

6 Conclusions

In this paper, we have studied adaptation process of neural network trees, modular neural network and conventional neural network. Experimental results with a recognition problem show that neural network tree whose nodes are expert neural networks is a neural network model with a comparable quality like modular neural network. Both models have better adaptation results than conventional multilayer neural network architecture but the time complexity for trained neural network trees increases exponentially with the number of inputs, rather the size (i.e. the number of hidden neurons) of each network. Thus it is necessary to reduce the number of inputs [3]. This is impossible for conventional neural networks because the number of inputs is usually fixed when the problem is given.

Table 1. Table of results.

	Neural network tree	**Modular neural network**	**Multilayer neural network**
average error value	*1.5%*	*1.2%*	*3.7%*

All models solve the pattern recognition task from the Fisher's Iris data set [7] in our experiments. The dataset consists of 150 samples from each of three species of Iris flowers (*Iris setosa, Iris virginica and Iris versicolor*). Four features were measured from each sample, they are the length and the width of sepal and petal. Based on the combination of the four features, Fisher developed a linear discriminated model to determine which species they are. The data set was divided into two sets: the training set contained 100 patterns and the test set 50 patterns. The error values for the training set were always near zero because perfect training was performed. Table 1 shows a table of results. There are shown average error values for the test set over 10 runs. Other numerical simulations give very similar results. In the future, we will study properties of neural network trees in detail, and try to propose better evolutionary algorithms for their designing.

References

1. Takeda, T., Zhao Q. F., and Liu, Y.: A study on on-line learning of NNTrees. In: Proc. International Joint Conference on Neural Networks, pp. 145-152, (2003).
2. Zhao, Q. F.: Evolutionary design of neural network tree -integration of decision tree, neural network and GA. In: Proc. IEEE Congress on Evolutionary Computation, pp. 240-244, Seoul, (2001).
3. Mizuno, S., and Zhao, Q. F.: Neural network trees with nodes of limited inputs are good for learning and understanding. In: Proc. 4th Asia-Pacific Conference on Simulated Evolution And Learning, pp. 573-576, Singapore, (2002).
4. Endou, T. and Zhao, Q.F.: Generation of comprehensible decision trees through evolution of training data. In Proc. IEEE Congress on Evolutionary Computation (CEC'2002) Hawai, 2002.
5. Zhao, Q. F.: Modelling and evolutionary learning of modular neural networks. In: Proc. The 6-th International symposiums on artificial life and robotics, pp.508-511, Tokyo (2001).
6. Quinlan, J. R.: C4.5: *Programs for machine learning*. Morgan Kaufmann Publishers, (1993).
7. http://en.wikipedia.org/wiki/Iris_flower_data_set (from 16/1/2009).
8. Di Fernando, A., Calebretta, R., and Parisi, D.: Evolving modular architectures for neural networks. In French R., and Sougne, J. (eds.).Proceedings of the Sixth Neural Computation and Psychology Workshop: Evolution, Learning and Development. Springer Verlag, London, 2001.
9. Fausett, V. L.: Fundamental of Neural Networks, Architecture, Algorithms and Applications, Prentice Hall; US Ed edition, 1994.

INTELLIGENT RECOGNITION, AUTHENTICATION AND CLASSIFICATION

Recognition of Hand Gestures Tracked by a Dataglove: Exploiting Hidden Markov Models Discriminative Training and Environment Description to Improve Recognition Performance

Vittorio Lippi, Emanuele Ruffaldi, Carlo Alberto Avizzano and Massimo Bergamasco

PERCRO laboratory, Scuola Superiore Sant'anna, Pisa, Italy
{v.lippi,e.ruffaldi,carlo,bergamasco}@sssup.it

Abstract. Sequence classification based on Hidden Markov Models (HMMs) is widely employed in gesture recognition. Usually, HMMs are trained to recognize sequences by adapting their parameters through the Baum-Welch algorithm, based on Maximum Likelihood (ML). Several articles have pointed out that ML can lead to poor discriminative performances among gestures. This happens because ML is not optimal for this purpose until the modellized process is actually an HMM. In this paper we present a gesture recognition system featuring a discriminative training algorithm based on Maximal Mutual Information (MMI) and the integration of environment information. The environment is described through a set of fuzzy clauses, on the basis of which a priori probabilities are computed. Adaptive systems such as unsupervised neural networks are used to build a codebook of symbols representing the hand's states. An experiment on a set of meaningful gestures performed during the interaction with a virtual environment is then used to evaluate the performance of this solution.

1 Introduction

1.1 Aim of the Work

Gesture recognition is becoming an important task in technology and science. At first it can be considered as an advanced way to interface the user to an interactive system: exploiting the significance of natural human actions as commands would increase the efficiency, both in using and in learning how to use the system[1].

While performing an operation in a virtual environment a user can test the effectiveness of his practical choices and train himself to work in various situation. Meanwhile the gesture recognition system could trace the steps performed: a model of the behavior of a skilled performer could be produced, and a generic user can be aided by the system to reproduce it. This can be useful in context where staff training, and producing documentation are crucial economical issues in the working process[2].

The gesture recognition system described in this work is designed to provide a tool to track tasks of interest during the assembly of mechanical parts. This system has been developed together with a virtual environment where the task is set. The system is oriented to scalability, in order to be suitable to work on task of realistic complexity. To achieve this the system takes advantage of discriminative training to adapt the hidden Markov models (HMM) used to recognize gestures and a system that produces prior probabilities for gestures. Prior probabilities are computed on the basis of the situation in which the gestures are performed. This makes possible to compare, at every recognition performed, just a small subset of plausible gestures, making the system scalable to an higher number of gestures. If an information about prior probabilities is known during the classifier training, the algorithm used incorporate it. The result is a classifier trained focusing on the differences between gestures more likely to be performed in the same context (and hence confused). Notice that when performing a meaningful task every object has really few ways to be used.

1.2 Structure of the Paper

Sections 2, 3 and 4 describe the solution proposed. In particular section 2 describes how a codebook to translate raw data into symbols is built; section 3 describes how HMMs are trained to classify sequences of symbols produces by user gestures and section 4 describes how the environment description is implemented and integrated with the recognition process. In section 5, an application of the system is presented together with the recognition accuracy achieved. Conclusions and planned improvements are in the section 6.

2 Codebook Definition and Use

2.1 Components and Parameters

Three steps are performed in order to build the codebook:

– complexity reduction by principal component analysis (PCA);
– frequency analysis by Fourier transform;
– quantization by neural networks;

The encoding process is summarized in figure 1, where a series of 11 signals is reduced to 3 by PCA. Note that once the PCA is applied the other two operations are applied independently on the series. This elaboration requires several parameters to be specified:

– the number of series kept after PCA. This parameter determines the total variance preserved;

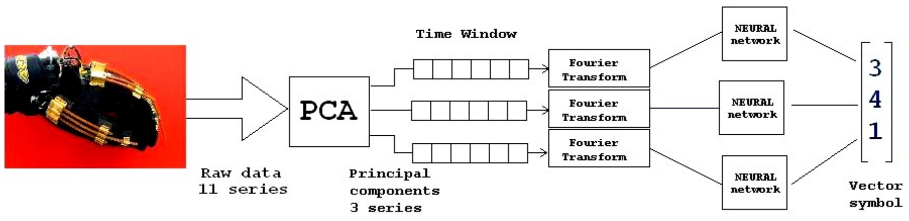


Fig. 1. Encoding input data.

- the time window on which the Fourier transform is applied. This is a trade off between the precision and the responsiveness of the system;
- the number of overlapping samples between time sequences;
- the number of symbols that the sequence classifier should recognize: If the symbols are too many the parameters of the sequence classifier can grow excessively in number and overfitting to the training set can occur.

2.2 Principal Component Analysis

Principal component analysis[3] is a procedure to obtain a linear transformation that, when applied to data series results in independent series. The series produced are ordered on the basis of the variance they hold: almost the whole variance of the system is kept using just some of them.

Usually PCA is applied to data offline, once a series is fully available. In this work an online encoding is needed. The codebook should be defined in a unique way, so the transformation can not be computed online just on a given time window. The training set is hence used to define the transformation offline and to estimate how many components should be hold to achieve a decent precision. The transformation is then applied online to incoming data.

2.3 Frequency Analysis

A discrete Fourier transform is applied on the series before using the neural networks. This allows the system to focus on the dynamics more than on the position. To make the results independent from the position, the first component of the Fourier transform, which represents the mean, is dropped from the obtained vector. The Fourier transform results in a complex vector, and in this form it is used as input for the neural network.

The implemented neural networks can work with complex values as well as real ones. When just the modulus of the Fourier transform is used as input for the neural networks the performance can be strongly degraded. This happens because, often, the phase is an important element in defining the nature of a gesture.

2.4 Neural Quantization

The neural networks are used to finally produce the encoding symbol. The competitive neural network exploited performs a quantization splitting the input space into several clusters[4].

The neural network output is an integer representing the cluster in which the input vector ha been classified. These numbers have no particular meaning, and no relation one with the other.

During the training phase the same training set is used both for the neural networks and the HMMs. The neural networks are trained before the HMMs because they are needed to produce the symbols. The networks are trained by iteratively minimizing the quadratic quantization error on the training set.

3 Sequence Recognition

3.1 Classification Principles

In recognition applications HMM are trained through examples to modelize the process to recognize,typically with an HMM for every process[5] [6]. The classification is usually performed by the assignment of a given input sequence to the class associated to the HMM with the highest probability to emit it, as described in the formula:

$$C = maxarg_i \left(P(S|M_i)\right) \tag{1}$$

Where C is the class given as output for the input sequence S, and M is the performed gesture. In this application, for reasons that will be explained in the next section, the classification is performed by a different formula:

$$C = maxarg_i \left(P(M_i|S)\right) \tag{2}$$

By this formalism, which represents the emission probability as a conditioned probability, the sequence is considered as the evidence produced by the gesture performed. Every gesture M_i is associated to its own probability

$$P(M_i) \tag{3}$$

That is supposed to be given *a priori* and not to be dependent on parameters. This makes sense considering P(M_i) to be a higher level concept than P($S|M_i$). For example, when recognizing gestures, the probability of a sequence to be observed during a gesture is related to movement coordination (low level), while the probability of a gesture to be performed is connected to the task performed (high level). The probability of a sequence S to be observed (namely, produced by the models featured by the classifier) is

$$P_\theta(S) = \sum_i P_\theta(S|M_i)P(M_i) \tag{4}$$

where the subscript θ marks the probabilities dependent on the parameters, represented by the vector θ

3.2 Parameter Estimation

Training the parameters of an HMM to maximize the emission probability of a given sequence is a well known problem for which several algorithms have been proposed in literature. Maximization of likelihood is the basis of the most of them. But representing a well-known and (relatively) easy principle to match the model to the distribution of examples is not optimal for classification problems in the general case[7]

Hence the principle of Maximum Mutual Information (MMI) is followed. The Mutual Information between a sequence S and the model M_i associated to the class i is:

$$I(M_i;S) = \log \frac{P_\theta(S,M_i)}{P_\theta(S)P(M_i)} = \log P_i(S|M_i) - \log(P(S)) \tag{5}$$

The aim of the training is to maximizing the Mutual Information between the given sequence and the model representing the class which the sequence is supposed to be classified in. Another interpretation of MMI[8] is useful to show how the term $P(M_i)$ is not dependent on θ:

$$I(M_i; S) = \log P_\theta(M_i|S) - \log P(M_i) \tag{6}$$

Finding the MMI over θ is equivalent to maximize $P_\theta(M_i|S)$, or adapt parameters to *enhance the probability that, **if** the sequence S is observed, it has been produced by the process M_i*. Notice again that $P_\theta(M_i|S)$ is a property of the whole classifier rather than the single HMM. Unlike ML, MMI approach could not be implemented by an expectation-maximization algorithm[9]. A gradient algorithm is therefore exploited.

The sets of sequences and models are considered in the training by maximizing the function:

$$F_\theta = \sum_i \sum_k I_\theta(M_i; S_k) \tag{7}$$

So the gradient is the sum of the gradients of $I_\theta(M_i; S_k)$ for every HMM M_i and every sequence S_k:

$$\nabla_\theta F_\theta = \sum_i \nabla_\theta \sum_k I_\theta(M_i; S_k) \tag{8}$$

Function 7 resembles the mutual information defined between two probability distributions of sequences and gestures that caused it

$$\sum_i \sum_k P(M_i, S_k) I_\theta(M_i; S_k) \tag{9}$$

but the term $P(M_i, S_k)$ is missing because the probability distribution $P(M, S)$ is not directly available, and computing it from the emission probabilities would cause the gradient to lose the property stated in 8. Anyway the training set itself works as a representation of sequences distribution, in the sense that a sequence S_k associated to an high actual $P(M_i, S_k)$ is represented by a large number of sequences similar to it in the training set, so the formula 7 is an approximation of formula 9. The derivative of formula 7 respect to a parameter θ_i could be written as

$$\frac{\partial I(M; A)}{\partial \theta_i} = \frac{\frac{\partial P_\theta(S|M)}{\partial \theta_i}}{P_\theta(S|M)} - \frac{\sum_{\hat{M}} \frac{\partial P_\theta(S|\hat{M})}{\partial \theta_i} P(\hat{M})}{P_\theta(S)} \tag{10}$$

where $\hat{M}$ are the generic models and M is the model related to the class wanted to recognize the sequence S. In order to compute

$$\frac{\partial P_\theta(S|M)}{\partial \theta_i} \tag{11}$$

the parameters could be evidenced in $P_\theta(S|M)$, for a and b

$$P_\theta(S|M) = \sum_{t=1}^{T} \sum_i \alpha(t-1) a_{ij} b_j(O_t) \beta(t) \tag{12}$$

and for π:

$$P_\theta(S|M) = \sum_i \pi_i a_{ij} b_j(O_0)\beta(0) \tag{13}$$

Applying the product rule to compute, for example, the derivative for a:

$$\begin{aligned} &\tfrac{\partial P_\theta(S|M)}{\partial a_{ij}} = \textstyle\sum_{t=1}^{T} \sum_i \alpha(t-1) \tfrac{\partial a_{ij}}{\partial a_{ij}} b_j(O_t)\beta(t) = \\ &\textstyle\sum_{t=1}^{T} \sum_i \alpha(t-1) b_j(O_t)\beta(t) \end{aligned} \tag{14}$$

Only the sequences considered likely to be confused are chosen as negative example. A *confusion margin* is quantified by the formula:

$$P(M_k|S_j) > \alpha P(M_i|S_j) \tag{15}$$

Where S_j is a sample for the class i, and k is the class which takes S_j as negative example when the condition is verified and α is a parameter to be specified: for $\alpha = 0$ all the sequences are taken in account, for $\alpha = 1$ only the ones misclassified at a certain step of the training algorithm become negative examples.

As previously explained $P(M|S)$ is connected with prior probabilities. This allows to take advantage from high level information in the training process. Batches of samples could be presented with different prior probabilities in order to decide which classes should be affected by negative training.

For example, if a sample S_j for the class i is presented during the training together with a null prior probability $P(M_k)$,S_j will not be a negative example for the class k, being $P(M_k|S_j) = 0$ and so, never greater than $\alpha P(M_i|S_j)$ that is never negative. This is useful because there are some gestures that could occur together in a certain environment and in a certain situation and hence *a priori* more likely to be confused.

4 Environment Description

4.1 Representation through Fuzzy Logic

An *ad hoc* defined language [10] is used to describe a logical model of the environment: objects, classes of objects and proposition about both objects and classes can be defined in a fashion that represents a simplification of a complete syntax for predicative logic. Fuzzy logic is particularly suitable to represent physical concepts like proximity or alignment[11].
In details the environment description consists of:

– a set of statements with their truth values, needed to describe environment state i.e., ”the hand is near the screw, with truth value 0.5”;
– a set of rules to compute the truth values of statements from other statements truth values, used to compute the state of the system and to update the description coherently i.e., ”if a screw is screwed on a bolt then that screw is blocked”;
– a set of rules to compute the probability of a certain gesture to be performed by the user, needed to use the environment description in the recognition process, a brief explanation of the method used is in the next paragraph;

– a set of rules to define the consequences of an action performed by the user, needed to update the scheme in realtime i.e. ”if the user performs the action *pick* and the hand is near an object that object is then in-hand with truth value 1”;

Describing the language in details is beyond the intention of this article, the syntax actually used in the application is hence not presented in this context where it could appear confusing. The examples in the previous list, given in plain text, are intended to show quickly the kind of statements that the described system can represent.

4.2 Inference Engine and Prior Probabilities

The *inference engine* is the set of functions that computes truth values of arbitrary statements from the logical description. In this section the algorithms exploited are described. When the truth value of a statement is required, the following action are performed by the engine:

1. The statement is searched in the description, if it is present its truth value is returned and the research is ended;
2. A rule having the state as *effect* is searched. If found the procedure is repeated for all the terms in the *cause*. The value of truth is than computed and returned.
3. If a value is not found in previous steps, 0 is returned.

Searching the truth value of a statement is like building an *inference tree* where the nodes are rules and leaves are known statements. To avoid that an infinite loop arises sub-threes in the *inference tree* cannot include the rule that represent their root. There is another feature that makes this system different from an usual theorem demonstrator[11]: it relies on rules written by the user to compute the truth values of statements from other statements truth values the latter exploits fixed inferential rules to find true sentences from true sentence.

A vector of gestures prior probabilities is specified for several logical condition. The logical conditions are fuzzy statements. When prior probabilities are required by the application a weighted sum of all the prior probability vectors is given. The truth value of the condition is used as weight for the related vector.

5 Performance

5.1 An Application

This application consists in assembling a small computer case in the virtual environment. The user interacts with the components through an avatar of his/her hand. The

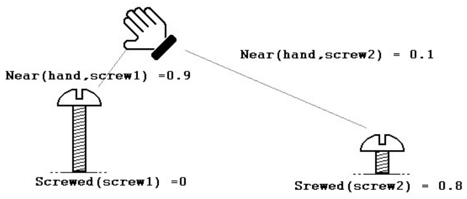


Fig. 2. A situation in a possible environment and fuzzy values for assumption on it.

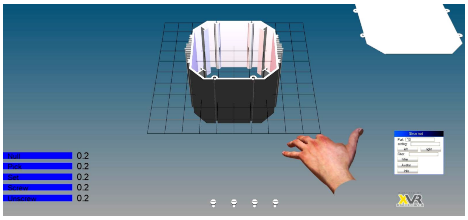


Fig. 3. The virtual environment at the beginning of the task.

recognized gestures trigger effects in the virtual environment. The finger movements performed are registered by sensor equipped glove [12]. The three-dimensional representation of the environment is updated coherently with the logical description. In order to build the case components should be put in place, set with the right orientation and than blocked. To complete the task the operations should be performed in the right order. The case is made with a box, a base and four screws to fix them together. The recognized gestures are: *null*, *pick*, *set*, *screw* and *unscrew*. The gesture *null* does not produce any action, it is recognized when the hand is almost still. It is needed to avoid recognition of random gestures when the use is not doing something. The gesture *pick* is used to take and release objects. *Set* is used to put an object in the right position for assembling. This overcomes the fact that in this application there is not a force feedback and the right position is hence difficult to find. Notice that ”being set” is a state of an object represented in the logical environment and it is anyway needed to mount an piece. The gestures *screw* and *unscrew* are used to fix screws on the box and to remove them. There are 5 degrees of screwing represented by a natural number, when the gesture *screw* is recognized the number is increased, on the opposite unscrewing decreases the number.

5.2 Accuracy

The system has been tested using examples of the 5 gestures produced by 5 users. Every user has been asked to perform every gesture for a minute. To build the codebook the 11 angular values describing the finger positions, sampled at 1 MHz are reduced to 3 series of data through a PCA, then the Fourier transform is applied on overlapping time windows of 50 samples. The resulting sample series have been divided in sequences of 8 symbols. The system has been cross-validated splitting the set of examples into 5 subset, used 4 by 4 as training set.

The main peculiarities of the proposed approach are the use of MMI instead of ML and the use of environment informations. Hence the test consists in comparing confusion matrices in case of HMM trained through ML and through MMI without and with environment information.

The MMI training outperforms ML training. This is evident especially for the gestures *screw* and *unscrew* that are particularly difficult to recognize, being very similar.

The last two tables show the improvement produced by the use of context information. As told in the tables captions, the information added consist just in the hand position and the screw state. Further performance gains would be possible adding information about the task performed by the user, for example taking in account that the

Table 1. Confusion matrix for MMI training.

	null	pick	set	screw	unscrew
null	**0.946**	0	0.054	0	0
pick	0	**0.927**	0	0.037	0.037
set	0.060	0.016	**0.890**	0.020	0.012
screw	0.008	0.042	0	**0.832**	0.117
unscrew	0	0.008	0	0.096	**0.896**

Table 2. Confusion matrix for ML training.

	null	pick	set	screw	unscrew
null	**0.915**	0.015	0.010	0.030	0.010
3	0.070	**0.668**	0.061	0.131	0.050
set	0.060	0.10	**0.702**	0.048	0.090
screw	0.070	0.80	0.030	**0.660**	0.150
unscrew	0.080	0.70	0.080	0.140	**0.680**

Table 3. Confusion matrix for MMI training, the recognition exploits context information: the hand is near a screw positioned to be screwed, the precision rises to 96.83%.

	null	pick	screw
null	**1**	0	0
pick	0	**0.934**	0.065
screw	0.008	0.021	**0.970**

user started to screw after positioning a screw would make more likely his/her intention to screw.

6 Conclusions and Future Work

The objective of the presented work was to improve HMM-based gesture recognition performance optimizing parameters adaptation and including high level information in the recognition process, as usually done in speech recognition applications where language models are exploited.In the proposed system this is represented by environment information. A test shown how adapting parameters with MMI produces a better classification performance compared with the usual ML-based training presented in literature, expecially in the classification of similar gestures. The algorithm high computational requirements have been compensated selecting a subset of the training samples at every step, without affecting the recognition performance of the algorithm. The introduction of environment information produced a further general performance improvement. The proposed fuzzy logic simplified inference system assures an efficient realtime update of environment description according to user actions.

In the presented version the action performed by the user can be saved as a list. It should be combined with the environment informations and, produce automatically a manual for the task performed. Another step forward will be the integration of this sys-

Table 4. Confusion matrix for MMI training, the recognition exploits context information: the hand is near an halfway screwed screw, the precision rises to 92.21%.

	null	screw	unscrew
null	**1**	0	0
screw	0	**0.935**	0.065
unscrew	0.084	0.021	**0.970**

tem into an application representing a complex industrial task, including the possibility for the user to use tools.

References

1. Melinda M. Cerney, J.M.V.: Gesture recognition in virtual environments: A review and framework for future development. Technical report, Iowa State University (2005)
2. Brough, J. E., Schwartz, M., Gupta, S.K., Anand, D.K., Kavetsky, R., Pettersen, R.: Towards the development of a virtual environment-based training system for mechanical assembly operations. Virtual Real. 11 (2007) 189–206
3. Moeslund, T.B.: Principal component analysis - an introduction. Technical Report Technical Report CVMT 01-02, ISSN 0906-6233, Aalborg University (2001)
4. Mathworks: Matlab manual. Mathworks. (2004)
5. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
6. L.R. Rabiner, B.J.: An introduction to hidden markov models. IEEE ASSP Magazine 3 (1986) 4– 16
7. Chow, Y.L.: Maximum mutual information estimation of hmm parameters for continuos speech recognition using the n-best algorithm. IEEE 2 (1990) 701–704
8. Bridle, J.S.: Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. Advances in neural information processing systems 2 (1990) 211–217
9. B. H. Juang, L.R.R.: Hidden markov models for speech recognition. Technometrics 33 (1991) 251–272
10. Lippi, V.: Design and development of a human gesture recognition system in three-dimensional interactive virtual environment. Master's thesis, University of Pisa, College of engineering. (2008) Consultation Allowed.
11. Stuart Russel, P.N.: Artificial Intelligence: A Modern Approach. Pearson (2003)
12. Rodriguez, O.P., Avizzano, C., Sotgiu, E., Pabon, S., Frisoli, A., Ortiz, J., Bergamasco, M.: A wireless bluetooth dataglove based on a novel goniometric sensors. Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on 1 (2007) 1185– 1190

Image Enhancement Technique using Adaptive Multiscale Retinex for Face Recognition Systems

Khairul Anuar Ishak[1], Salina Abdul Samad[1]
M. A. Hannan[1] and Maizura Mohd Sani[2]

[1] Dept. of Electrical, Electronics and Systems Engineering
Faculty of Engineering and Built Environment, University Kebangsaan Malaysia
43600, UKM Bangi, Selangor, Malaysia
{nuarscc,salina,eehannan}@vlsi.eng.ukm.my
[2] Institute of Microengineering and Nanoelectronics, University Kebangsaan Malaysia
43600, UKM Bangi, Selangor, Malaysia
maizura@vlsi.eng.ukm.my

Abstract. Various illumination effects in an image are one of the states of difficulty that should be solved in order to get a satisfactory result in face recognition task. The inhomogeneous intensities of the image has led to many plans and algorithms to devastate the cause and next to eliminate the illumination. The focus of this paper is to enhance the image by reducing illumination effects; employing a preprocessing step i.e. adaptive multiscale retinex as the illumination correction method before accomplishing the recognition task. The performance of this method is evaluated using the Yale database and has lower equal error rate compared with single scale retinex and conventional multiscale retinex.

1 Introduction

In face recognition, usually there are some inconsistencies between the real scenes and the training set images. One of them is illumination variations such as shadow, blur, dark and noise occurring in the images. Sometimes this can cause degradation in the algorithm to recognize the face image. In this paper, we want to reduce the unwanted effects in face images by applying adaptive multiscale retinex as a preprocessing step. Multiscale retinex was initially used to provide stability in color images; however it is also competent to be used in gray scale images.

Lightness and color uniformity refer to wide range of intensity and spectral illumination variations [1]. Multiscale retinex is formed from the retinex theory by Edwin Land [2]. Land proposed the idea of retinex as a model of lightness to measure the lightness response in an image.

However Land did not apply the model to image enhancement algorithm, but this is done by Jobson where they define the properties of the surround/center retinex function [3]. The characteristic they describe is single scale retinex when they performed logarithmic after the surround function. They also apply ‘canonical’ gain offset to the retinex output to clip certain parts of the highest and lowest signal excursion. However, single scale retinex can either provide dynamic range compression on small scale, or tonal rendition for large scale image. This limitation expands single scale retinex to a more balanced method that is multiscale retinex.

After that, another characteristic of multiscale retinex were found where other than dynamic range compression, multiscale retinex purposes are to replicate tone in an image in order to reduce its dependencies in lighting conditions and improved spatial resolution[4]. So we use this characteristic in this paper to lessen illumination effects in order to obtain controlled lighting condition in a face image. We modify the present multiscale retinex by including histogram shifting and adaptive histogram equalization to the original algorithm as to have a more uniform face image contrast than the original method. The detail of this proposed method will be discussed in section 3. Before that, section 2 will cover the original theory of multiscale retinex. Section 4 will describe the experimental results and lastly is the conclusion in Section 5.

2 The Original Multiscale Retinex

The original multiscale retinex essentially measure the intensity of an image and estimate the illumination from the proportion of the local image mean intensity value. By applying Gaussian filter, the image is smoothed at different weight and size in order to find the mean of the image. To obtain the retinex output, the filtered image is divided using the illuminated image (input). Then, logarithmic function is done to compress dynamic range of images with large variations in pixel value [5] before the image is reconstructed again using additive function.

The original multiscale retinex algorithm is obtained from single scale retinex [4] as in (1):

$$R(x,y)=\log I(x,y)-\log[F(x,y)*I(x,y)] \tag{1}$$

where I(x,y) is input image, R(x,y) is retinex output, F(x,y) is the Gaussian surround function. Symbol * denotes convolution. Gaussian surround function is given by:

$$F(x,y)=K.e^{-(x^2+y^2)/c^2} \tag{2}$$

where c is Gaussian shaped surrounding space constant. The value of c is related to visual angle in the direct observation which is determined through experiment. K is selected such that:

$$\iint F(x,y)dxdy=1 \tag{3}$$

Until this stage the single scale retinex would only provide tone reproduction and dynamic range compression at certain scale in an image. The image would have only one of the important characteristics. Thus, to overcome this limitation, superposition of different scale at certain weight would solve this problem as shown in (4), where

N is number of scale, where R_{ni} is different scale of single scale retinex. ω_n is the weight of each single scale retinex with equal value.

$$R_{MSRi}=\sum_{n=1}^{N}\omega_n R_{ni} \tag{4}$$

3 The Adaptive Multiscale Retinex

After applying the original multiscale retinex, we found that the image was too dark. This meant that the image brightness and contrast needed to be altered. Thus we modified the algorithm by applying a recombination with the original image. According to [6], a method need to be applied to restore the information in different regions to smoothen the global contras in the image according to which region is darker or brighter. The information here is, different intensity in different regions in the original picture. For this reason, recombination is needed to restore the information as in (5):

$$R_{MSRi}=\sum_{n=1}^{N}\omega_n R_{ni}+\omega_{original}\cdot\log(\text{original})\,. \tag{5}$$

After recombination with weighted original picture, adjustment is made on the histogram by performing a constant shift which helps improve the entire global brightness of the image. To shift the histogram is a simple task, where in the range of 0-255 the image pixels should be. In order to allocate the pixels in the range, we set the initial maximum pixel (MinVal) as 0, and the minimum pixel (MaxVal) as 255. Then we evaluate the entire image pixels one by one and update the new value (NewVal) using (6). Prior to that, every pixel value (PixVal) has to be tested whether it is higher than the MaxVal or lower than the MinVal. If the value is higher, then the MaxVal will be the PixVal value and the similarly, MinVal if the value of PixVal is lower than MinVal. These values are needed to find the new value which is compute from

$$\text{NewVal}=\left[\frac{\text{Pixval-MinVal}}{\text{MaxVal-MinVal}}\right]*255\,. \tag{6}$$

Next, we execute a local image enhancement technique that divides the image into rectangular blocks. Usually how many blocks should be used is determine through experiments. First, obtain the cumulative density function of the small region histogram. Then the centre pixel of the region is equalized (histogram equalization) and moved to the adjacent pixel in the rectangular region. This process is called adaptive histogram equalization (AHE) [7].

Overall, the adaptive multiscale retinex algorithm is shown as in figure 1.

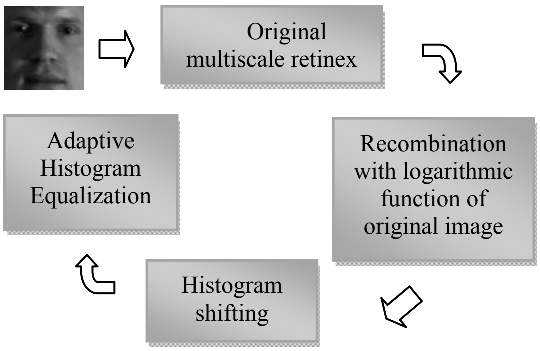

Original multiscale retinex
Recombination with logarithmic function of original image
Histogram shifting
Adaptive Histogram Equalization

Fig. 1. The adaptive multiscale retinex flowchart.

4 Experimental Results

We evaluate the performance of the preprocessing methods using Eigenface [8] as the feature extractor while Euclidean distance is used for the matching purpose. We also implement a fusion of Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) [9] for data reduction. The performance is evaluated using well known benchmarking measures for biometrics system i.e. Equal Error Rate (EER) [10]. To compute EER, two components must be determined. The first is false acceptance rate (FAR), when the impostor is falsely regarded as the client. Another one is false rejection rate (FRR), when the client is falsely regarded as the imposter. Here the client is the authorize person in the face recognition system. The EER is the cross-over value where FAR and FRR coincide.

The dataset we use to evaluate the algorithms is Yale [11], which contains 165 grayscale images of 15 individuals. There are 11 images per subject, one per different facial expression or configuration. The 11 images show various extreme illuminations and pose criteria. We randomly selected 5 images from each subject to be the training sample and 5 images of each subject as the testing sample. Our experimental face condition is cropped face images. The size of all images is standardized to 50x60.

The experiments are done using grayscale images as the inputs. Three methods are compared, i.e., single scale retinex, multiscale scale retinex and the adaptive multiscale retinex. Table 1 shows the EER for these methods where we can see that the EER for the adaptive multiscale retinex is the lowest compared to the original multiscale retinex and single scale retinex.

Table 1. EER comparison of three methods.

Method	EER (%)
Single Scale Retinex	11.87
Multiscale Retinex	10.33
Adaptive Multiscale Retinex	10.27

The EER results correlate with the output images of the three methods. To illustrate, we choose 4 faces from the database which have been illuminated with different lighting conditions: cast shadow, attached shadow, specular reflection and diffuse reflection [12]. The lighting conditions are shown in figure 2. Figure 3 illustrates the outputs for all three methods. For face number 1, only the adaptive multiscale retinex is able to eliminate the diffuse reflection. For face number 2, there are specular reflection and strong cast and attached shadow, where the adaptive algorithm is capable in removing the specular reflection. The condition in face number 3 is the same as number 1. Face number 4 contains cast shadow, specular and diffuse reflection. The adaptive multiscale retinex is able to remove all the lighting distraction on the face image, except the cast shadow.

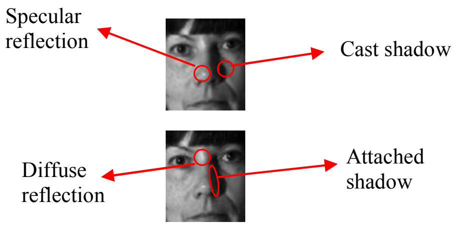


Fig. 2. Different lighting conditions of a face.

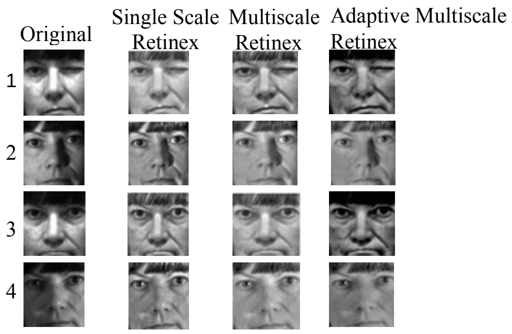


Fig. 3. The image outputs using three methods for person 1, 2, 3, and 4.

Figure 4(a), (b), (c) and (d) show the output histograms for face images 1, 2, 3, and 4 before and after applying the adaptive multiscale retinex. For all the histograms, the upper figure indicates the histogram before and the lower figure is the one after. All the histograms show more balanced tone representation in gray scale values. The histograms centred at the middle show ordinary conditions with peaks and gradually tapering off on the left and right sides of the histogram. This proves that the method is able to reduce broad tonal range from the original face image.

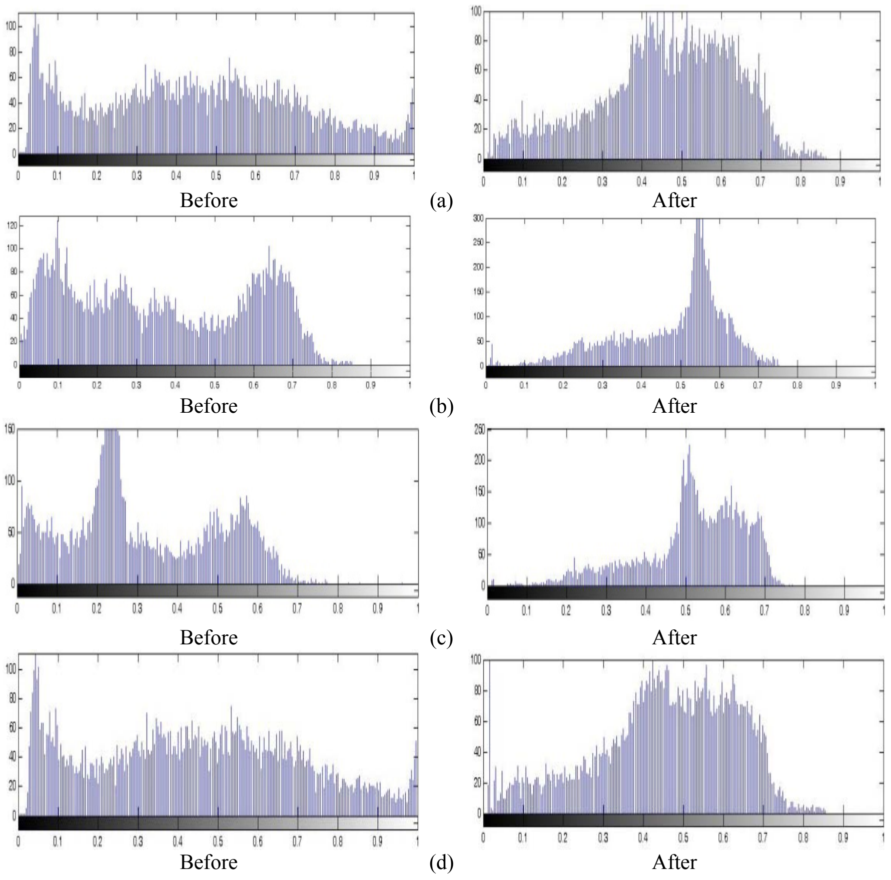


Fig. 4. (a), (b), (c), (d). The histogram before pre-processes (the original image) and after pre-processed with adaptive multiscale retinex for person 1, 2, 3 and 4.

5 Conclusions

In this paper, an adaptive multiscale retinex algorithm is presented. The purpose is to remove illumination appearances. This is achieved by modifying the multiscale retinex algorithm with adaptive histogram equalization and histogram shifting. The performance of this method is tested using the Yale dataset and shown in terms of EER rate and output comparisons with single scale retinex and conventional multiscale retinex.

References

1. Rahman, Z.: Properties of a Center/Surround Retinex Part One: Signal Processing Design. NASA Contractor Report#198194 (1995).

2. Land, E.: An alternative technique for the computation of the designator in the retinex theory of color vision. Proceedings of the national Academy of Sciences of the USA, Vol.83.3078-3080 (1986).
3. J.Jobson,D., Rahman,Z., A.Woodell, G.: A Multiscale Retinex for Bridging the Gap between Color Images and the Human Observation of Scenes. IEEE Transaction on Image Processing, Vol. 6(3).965-976 (1997).
4. J.Jobson,D., Rahman,Z., A.Woodell, G.: Properties and Performance of a Center/Surround Retinex. IEEE Transaction on Image Processing, Vol.6(7).451-462 (1997).
5. Gonzalez,R.C., Woods,R.E.: Digital Image Processing. Addison-Wesley (1992).
6. Herscovitz,M., Yadid-Pecht,O.: A Modified Multi scale Retinex with an improved global impression of brightness for wide dynamic range pictures. Journal of Machine Vision and Application,Vol.15.220-228 (2004).
7. Zhiming,W.,Jianhua,T.: A Fast Implementation of Adaptive Histogram Equalization. Proceedings of The 8th International Conference on Signal Processing (2006).
8. Turk, M. and A.P.Pentland.: Eigenfaces for recognition. J.Cognitive Neurosci, Vol.13.71-86 (1991).
9. Ishak,K.A., Abdul Samad,S.A., and Hussain,A.: A Face Detection and Recognition for Intelligent Vehicles. Information Technology Journal,Vol. 5(3).507-515 (2006).
10. Wayman, J.L.: Error rate equations for the general biometric system. IEEE Robotic & Automation, Vol.6 (3). 35-48 (1999).
11. Georghiades, A.S., Belhumeur, P.N. and Kriegman, D.J.: From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose IEEE Transaction Pattern Analysis Machine Intelligence,Vol.23.643-660 (2001).
12. Nishiyama,M., Kozakaya,T., and Yamaguchi,O.:Illumination Normalization using Quotient Image-based Techniques. Recent Advances in Face Recognition. Austria: I-Tech Education and Publishing (2008).

A Blind Source Separation Algorithm for the Processing and Classification of Electro-oculogram Data

Fernando Rojas[1], Rodolfo V. García[2], Olga Valenzuela[3]
Luís Velázquez[4] and Belén San Román[5]

[1] Department of Computer Architecture and Technology, University of Granada, Spain
frojas@atc.ugr.es
[2] Network Department, University of Holguín, Cuba
rgb1707rgb@yahoo.es
[3] Department of Applied Mathematics, University of Granada, Spain
[4] Centre for the Research and Rehabilitation of Hereditary Ataxias, Holguín, Cuba
[5] University of Granada, Spain

Abstract. Abnormalities in the oculomotor system are well known clinical symptoms in patients of several neurodegenerative diseases, including modifications in latency, peak velocity, and deviation in saccadic movements, causing changes in the waveform of the patient response. The changes in the morphology waveform suggest a higher degree of statistic independence in sick patients when compared to healthy individuals regarding the patient response to the visual saccadic stimulus modeled by means of digital generated saccade waveforms. The electro-oculogram records of six patients diagnosed with ataxia SCA2 (a neurodegenerative hereditary disease) and six healthy subjects used as control were processed to extract saccades. We propose the application of a blind source separation algorithm (or independent component analysis algorithm) in order to find significant differences in the obtained estimations between healthy and sick subjects. These results point out the validity of independent component analysis based techniques as an adequate tool in order to evaluate saccadic waveform changes in patients of ataxia SCA-2.

1 Introduction

The ocular movement records have been widely used in processing and classification of biological signals and pathological conditions: clinical sleep scoring [10, 11], cerebellar dysfunctions [12-14], diagnosis of the visual system [15, 16], amongst others, also in human computer interface and visual guided devices [17-19]. The Spino Cerebellar Ataxia type 2 (SCA-2) is an autosomal dominant cerebellar hereditary ataxia with the highest prevalence in Cuba, reporting up to 43 cases per 100,000 inhabitants in the province of Holguin. In most families there is clinical and neuropathological evidence of additional involvement of brainstem, basal ganglia, spinal cord, and the peripheral nervous system [1]. This form of ataxia occurs commonly in persons of Spanish ancestry in north-eastern Cuba, a figure much higher than that found in western Cuba or in other parts of the world. The high prevalence is probably the result of a founder effect, but might be due to an interaction between a mutant gene and an unidentified environmental neurotoxin [2, 4].

Several studies have reported oculomotor abnormalities in SCA2 [1, 4-8]. Specifically, slowness of saccades has been suggested as a relatively characteristic finding in SCA2[4, 8]. This fact determines significant differences in saccade morphology between healthy individuals and patients with SCA-2, mainly for 60º of stimulus amplitude. The electro-oculographical records are quite different in healthy individuals and patients with a severe ataxia as it is shown in Figure 1 for a smooth pursuit experiment.

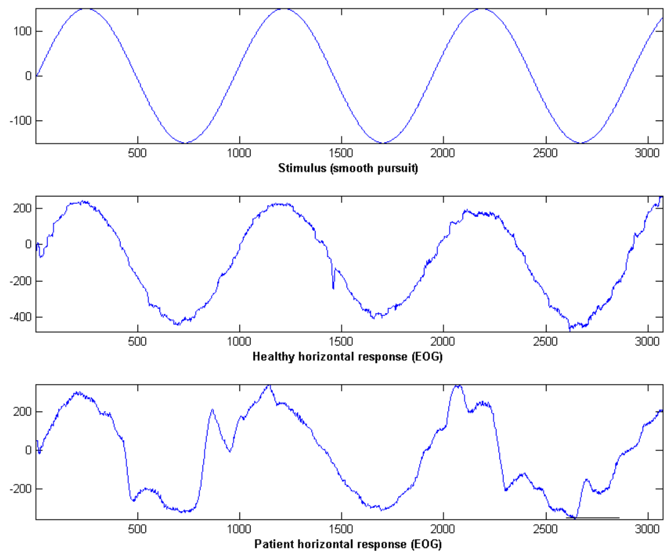


Fig. 1. Electro-oculographic response to a smooth pursuit stimulus (top) obtained for a healthy subject (center) and a patient of SCA-2 ataxia (bottom).

2 Using Blind Source Separation for Ataxia SCA2 Diagnosis

2.1 Hypothesis for the Proposed Method

Independent component analysis is aimed to find a linear transformation given by a matrix **W**, so that the random variables $\mathbf{y}_i$, (i=1,…,n) of $\mathbf{y}=[\mathbf{y}_1,\ldots,\mathbf{y}_n]$ are as independent as possible in:

$$\mathbf{y}(t) = \mathbf{W} \cdot \mathbf{x}(t) \tag{1}$$

This linear blind source separation approach is suitable for the signals obtained by the EOG, as well as in other medical analysis such as electroencephalography (EEG), electrocardiography (ECG), magneto-encephalography (MEG), and functional magnetic resonance imaging (fMRI) [20-26].

As it was shown in Section 1, in the analysis of EOG oriented to the detection of SCA2 experts anticipate two possible behaviors of the individuals: sick and healthy conduct. During an experiment over a healthy subject, the horizontal movement of the eye is expected to follow the stimulus signal. Therefore, the horizontal eye movement and the stimulus will hold a direct dependence between them, i.e. the signals are not independent. In contrast, a sick individual may present a more chaotic response, depending on the severity of the disease. Consequently, the subject response will not depend in such a high degree on the stimulus signal, and the signals are independent (or at least, "not so dependent").

Therefore, the proposed methodology uses independent component analysis as a classification algorithm criterion: if the independence measure (normally mutual information) reveals independence between the individual response and the stimulus signal, then it is rather possible that the individual presents some degree of ataxia or related disease.

2.2 Description of the Blind Source Separation Algorithm

The proposed algorithm for ataxia SCA-2 diagnosis will go along the following steps:

1. Set both horizontal response and stimulus signal in the same phase, i.e. correct the delay between the stimulus change and the saccade.
2. Normalize signals (x).
3. Apply ICA algorithm. Any well known ICA algorithm may be applied at this point (FastICA [27], Jade [28], GaBSS [29-30], etc.).
4. Normalize estimations (y)
5. Calculate error measure between estimations (y) and mixtures (x) according to the root mean square error expression:

$$RMSE(\mathbf{x}_i, \mathbf{y}_i) = \sqrt{\frac{\sum_{t=0}^{N} \left[x(t) - y(t)\right]^2}{N}} \tag{2}$$

6. Depending on the obtained error measure, a simple categorization algorithm (such as C-means) may be applied in order to classify individuals. Otherwise, a human expert may help in subject categorization based on the ICA results.

3 Results

The electro-oculogram recordings of six patients with severe ataxia and six healthy subjects diagnosed and classified in the "Centre for the Research and Rehabilitation of Hereditary Ataxias (CIRAH)" were used in order to perform the analysis of repeated ocular saccadic movement tests for 10^o, 20^o, 30^o and 60^o divergence stimuli.

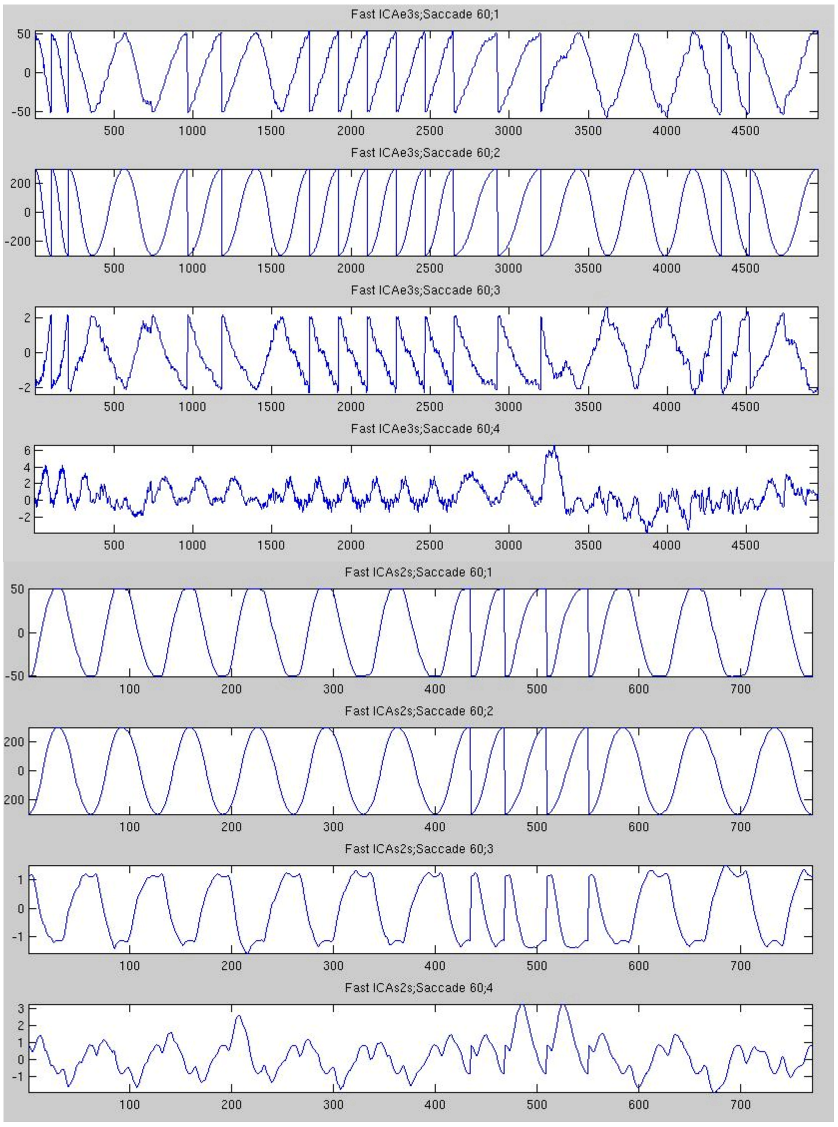


Fig. 2. Stimulus (1), response (2) and ICA components (3 and 4) obtained at 60º of stimulation for patients (top) and control subjects (bottom).

All the records were carried out by the medical staff of CIRAH. Each individual was placed in a chair, with a head fixation device to avoid head movements, the variables were collected by a two channel electronystagmograph (Otoscreen, Jaeger-Toennies). Recording conditions were set as follows: electrodes of silver chloride placed in the external borders of right eye (active electrode) and left eye (reference electrode), high pass filtering 0.002 Hz, low pass filtering 20 Hz, sensitivity 200 µV/division, and sampling frequency 200 Hz. For stimulus generation a black screen CRT display showing a white circular target with an angular size of 0.7º was used.

The stimulus and patient response data are automatically stored in ASCII files by Otoscreen electronystagmograph.

The patient response was filtered using a median filter, to obtain a clean waveform of the patient response, afterwards it was phased with the stimulus. Finally FastICA was applied to get the independent components (See Figure 2).

As Figure 3 depicts, results show that the error measure obtained for SCA-2 patients is clearly differentiable for the same measure obtained for control subjects. That is due to the fact mentioned in the hypothesis (Section 2.1) that if the independence measure reveals independence between the individual response and the stimulus signal, then it is possible that the individual presents some degree of ataxia. When the original signals (stimulus and response) were independent, the estimations are close to those sources and, therefore, the RMS error decreases.

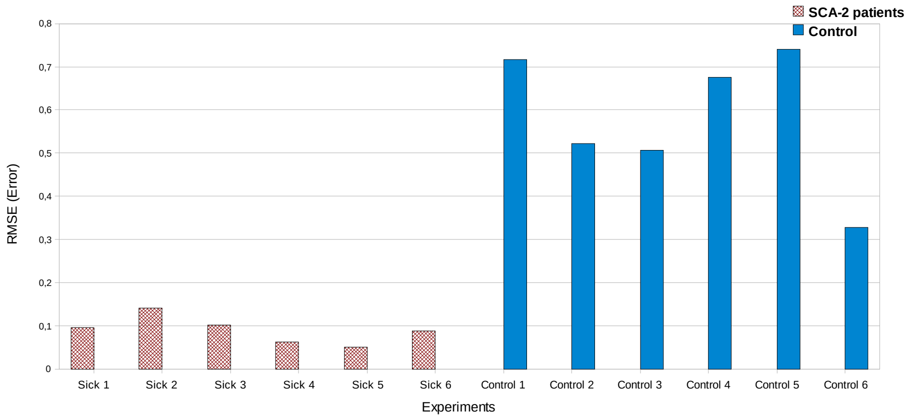


Fig. 3. Root mean squared error between the estimations and the sources after the application of the algorithm to EOG data corresponding to SCA-2 patients (left) and control subjects (right).

4 Discussion

The results were obtained from six control subjects and six patients. Confirming our hypothesis, starting from electro-oculography experiments, patients showed a different behavior in terms of the visual response to a fixed stimulus (see Figure 2 and Figure 3). Therefore, after applying our proposed approach to the raw EOG data, classification and diagnosis can be made easily by simple human inspection of the results. Nevertheless, further research in this line may help in the categorization of the several stages of severity of SCA-2.

The proposed method starts from the assumption that the response to a visual stimulus is different in a healthy individual when compared to the response of an individual afflicted by SCA-2. In the later situation, the response from the individual is not dependent on the visual stimulus, so that the ICA algorithm estimations will be similar to the obtained observations. This criterion has shown to be suitable in order to distinguish between sick (patients) and healthy (control) individuals.

References

1. Bürk, K., Fetter, M., Abele, M., Laccone, F., Brice, A., Dichgans, J., Klockgether, T.: Autosomal dominant cerebellar ataxia type I: oculomotor abnormalities in families with SCA1, SCA2, and SCA3. Journal of Neurology 246 (1999) 789-797.
2. Orozco, G., Estrada, R., Perry, T.L., Araña, J., Fernandez, R., Gonzalez-Quevedo, A., Galarraga, J., Hansen, S.: Dominantly inherited olivopontocerebellar atrophy from eastern Cuba: Clinical, neuropathological, and biochemical findings. Journal of the Neurological Sciences 93 (1989) 37-50.
3. Velázquez Pérez, L., Sánchez Cruz, G., Canales Ochoa, N., Rodríguez Labrada, R., Rodríguez Díaz, J., Almaguer Mederos, L., Laffita Mesa, J.: Electrophysiological features in patients and presymptomatic relatives with spinocerebellar ataxia type 2. Journal of the Neurological Sciences 263 (2007) 158-164.
4. Velázquez, L.: Ataxia espino cerebelosa tipo 2. Principales aspectos neurofisiológicos en el diagnóstico, pronóstico y evaluación de la enfermedad. Ediciones Holguín, Holguín (2006).
5. Rivaud-Pechoux, S., Dürr, A., Gaymard, B., Cancel, G., Ploner, C.J., Agid, Y., Brice, A., Pierrot-Deseilligny, C.: Eye movement abnormalities correlate with genotype in autosomal dominant cerebellar ataxia type I. Annals of Neurology 43 (1998) 297-302.
6. Hutton, J.T., Albrecht, J.W., Kuskowski, M., Schut, L.J.: Abnormal ocular motor function predicts clinical diagnosis of familial ataxia. Neurology 37 (1987) 698-701.
7. Yamamoto, H., Sobue, I.: Analysis of the abnormal ocular movements in spinocerebellar degenerations and the influence of the thyrotropin releasing hormone. Clinical Neurology 20 (1980) 637-645.
8. Klostermann, W., Zühlke, C., Heide, W., Kömpf, D., Wessel, K.: Slow saccades and other eye movement disorders in spinocerebellar atrophy type 1. Journal of Neurology 244 (1997) 105-111.
9. Velázquez-Pérez, L., Almaguer-Mederos, L., Santos-Falcón, N., Hechavarría-Pupo, R., Sánchez-Cruz, G., Paneque-Herrera, M.: Spinocerebellar ataxia type 2 in Cuba. A study of the electrophysiological phenotype and its correlation with clinical and molecular variables. Revista de Neurologia 33 (2001) 1129-1136.
10. Virkkala, J., Hasan, J., Värri, A., Himanen, S.-L., Müller, K.: Automatic sleep stage classification using two-channel electro-oculography. Journal of Neuroscience Methods 166 (2007) 109-115.
11. Estrada, E., Nazeran, H., Barragan, J., Burk, J.R., Lucas, E.A., Behbehani, K.: EOG and EMG: Two Important Switches in Automatic Sleep Stage Classification. Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE (2006) 2458-2461.
12. Spicker, S., Schulz, J.r.B., Petersen, D., Fetter, M., Klockgether, T., Dichgans, J.: Fixation instability and oculomotor abnormalities in Friedreich's ataxia. Journal of Neurology 242 (1995) 517-521.
13. Yokota, T., Hayashi, H., Hirose, K., Tanabe, H.: Unusual blink reflex with four components in a patient with periodic ataxia. Journal of Neurology 237 (1990) 313-315.
14. Hubner, J., Sprenger, A., Klein, C., Hagenah, J., Rambold, H., Zuhlke, C., Kompf, D., Rolfs, A., Kimmig, H., Helmchen, C.: Eye movement abnormalities in spinocerebellar ataxia type 17 (SCA17). Neurology 69 (2007) 1160-1168.
15. Güven, A.l., Kara, S.: Classification of electro-oculogram signals using artificial neural network. Expert Systems with Applications 31 (2006) 199-205.
16. Irving, E.L., Steinbach, M.J., Lillakas, L., Babu, R.J., Hutchings, N.: Horizontal Saccade Dynamics across the Human Life Span. Invest. Ophthalmol. Vis. Sci. 47 (2006) 2478-2484.
17. Kumar, D., Poole, E.: Classification of EOG for human computer interface. Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings, Vol. 1 (2002) 64-67.

18. Brunner, S., Hanke, S., Wassertheuer, S., Hochgatterer, A.: EOG Pattern Recognition Trial for a Human Computer Interface. Universal Access in Human-Computer Interaction. Ambient Interaction (2007) 769-776.
19. Komogortsev, O.V., Khan, J.I.: Eye movement prediction by Kalman filter with integrated linear horizontal oculomotor plant mechanical model. Proceedings of the 2008 symposium on Eye tracking research & applications. ACM (2008) 229-236.
20. Vaya, C., Rieta, J.J., Sanchez, C., Moratal, D.: Convolutive Blind Source Separation Algorithms Applied to the Electrocardiogram of Atrial Fibrillation: Study of Performance. Biomedical Engineering, IEEE Transactions on 54 (2007) 1530-1533.
21. Anemüller, J.r., Sejnowski, T.J., Makeig, S.: Complex independent component analysis of frequency-domain electroencephalographic data. Neural Networks 16 (2003) 1311-1323.
22. Funase, A., Yagi, T., Barros, A.K., Cichocki, A., Takumi, I.: Single trial analysis on saccade-related EEG signal. Neural Engineering, 2007. CNE '07. 3rd International IEEE/EMBS Conference on (2007) 371-374.
23. Hironaga, N., Haruhana, K., Liu, L.C., Fenwick, P.B.C., Ioannides, A.A.: Monitoring of eye movement and its use for artifact elimination. International Congress Series 1270 (2004) 134-137.
24. Milanesi, M., Martini, N., Vanello, N., Positano, V., Santarelli, M., Landini, L.: Independent component analysis applied to the removal of motion artifacts from electrocardiographic signals. Medical and Biological Engineering and Computing 46 (2008) 251-261.
25. Guilhon, D., Medeiros, E., Barros, A.K.: ECG Data Compression by Independent Component Analysis. Machine Learning for Signal Processing, 2005 IEEE Workshop on (2005) 189-193.
26. Kocyigit, Y., Alkan, A., Erol, H.: Classification of EEG Recordings by Using Fast Independent Component Analysis and Artificial Neural Network. Journal of Medical Systems 32 (2008) 17-20.
27. Hyvarinen, A., Oja, E.: A Fast Fixed-Point Algorithm for Independent Component Analysis. Neural Computation 9 (1997) 1483-1492.
28. Cardoso, J.-F.: Source separation using higher order moments. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, Vol. 4 (1989) 2109-2112.
29. Rojas, F., Alvarez, M.R., Salmerón, M., Puntonet, C.G., Martin-Clemente, R.: Adaptive and Heuristic Approaches for Nonlinear Source Separation. Proceedings of the International Joint Conference on Neural Networks, Vol. 1 (2003) 720-725.
30. Martin-Clemente, R., Puntonet, C.G., Rojas, F.: Post-nonlinear blind source separation using methaheuristics. Electronics Letters, Vol. 39, Issue: 24. (2003) 1765- 1766.

ANN and Intelligent Systems Implementation

Implementation of a Modular Neural Network in a Multiple Processor System on Chip to Classify Electric Disturbance

Danniel Cavalcante Lopes, Rafael Marrocos Magalhães
Jorge Dantas de Melo and Adrião Duarte Dória Neto

Federal University of Rio Grande do Norte – UFRN, Natal, RN, Brasil
danniellopes@gmail.com,{rafael,jdmelo,adriao}@dca.ufrn.br

Abstract. This paper shows the effectiveness of a modular neural network composed of multilayers experts trained with a hybrid algorithm implemented in a multiprocessor system on chip. The network is applied on the classification of electric disturbances. The objective is to show that, even a FPGA with hardware restrictions, it could be used to implement a complex problem, when parallel processing is used. To improve the system performance was used four soft processors with a shared memory.

1 Introduction

The artificial neural network has been utilized to solve larger number of engineering problems, including the functions approximation [1], control as well as patterns classification [2]. The type of network, its architecture and its training algorithm are chosen and evaluated according to dimension and complexity of the problems. Scientists have been researching many learning machines methods, including committee machines, to solve complex problems [2, 3, 4].

The quality of energy provided by an electric system is one of the greatest point of interest for concessionaire and electric energy consumers. The literature presents distinct approaches in the acquisition, characterization and classification of disturbs present in power grids. Among these contributions, could be included the application of Souza et al. [5] utilizing multilayer perceptrons with resilient propagation (RPROP) training algorithm in disturbance classification, the discrete wavelet transform in characterization of voltage or current signals made by Machado et al. [6] and the detailed analysis of the electric signal pre-processing influence in neural network classifier [7].

At the same time the growth of the Field Programmable Gate Arrays (FPGA) capabilities, make them viable for the implementation of complete System-on-Chip (SoC) solution on the resolution of some complex problems [8]. Even with this growth, these systems have much less power of processing than a modern general processor. So with this restriction, maybe a more complex problem could not be implement at real time in a small FPGA. The goal of this work is take advantage of ideas from parallel processing to increase the performance of the system.

Following this development perspective, this work reports a learning algorithm for extend modular neural network and the results obtained and also shows an embedded architecture where the algorithm was executed and its performance.

2 Modular Neural Network

Committee Machines are neural network structures that use a concept commonly used: divided and conquer. This concept aims to divide a large and complex task in a set of sub-tasks that are easier to be solved and then regrouped again. From that, the committee machine could be defined, in summary, as a set of learning machine, also called experts, whose decisions are combined to achieve a better answer than the answers achieved individually, in other words, a machine with better performance.

In the last years one of the mains areas of learning machine is the characterization of methods capable to build these committee machines. Them could be divided into static and dynamic structures; the modular network, as seen in Fig. 1, is a dynamic type of committee. It is means that the input signal is used by the gating network to build the global response.

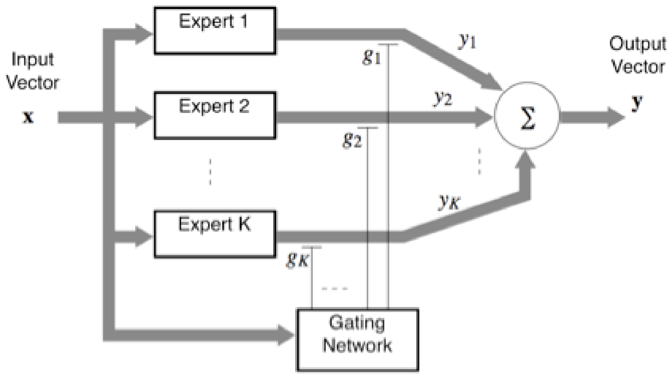


Fig. 1. Modular Neural Network Diagram.

An advantage of modular networks when compared with other neural networks is the learning speed. The learning processing is accelerated in problems where exist a natural decomposition of the data at simple functions. To develop the committee machine architecture and to implement the experts, was selected the multi layer perceptron (MLP).

During the analysis and testing of the modular network presented by Jacobs and Jordan [9], it was observed the structure was efficient for some simple problems. The algorithm, when used with complex problems, is unable to found a good solution. So we decided to use a larger architecture, for this, was opted to add hidden layers, as well as MLP networks, and neurons with nonlinear activation function at both structures: experts and gating network. Also in each layer a bias was added. From the parallelism standpoint, the adopted strategy was to involve an expert for each task, so each MLP network is being treated by a unique processor. The committee machine training, was conducted similarly to the MLP using backpropagation.

To train the modified modular network, was necessary to adjust the modular training algorithm. The way in which it was implemented enables to set, not only, different architectures to each expert, but also specific values of the learning algorithm, such as learning rates, momentum rate and delta-bar-delta parameters. Its possible, if desired, assign independent training sets to each expert network. A brief description from this algorithm is shown below

2.1 Hybrid Algorithm

To train the modified modular network was developed an algorithm adapted according to Jacobs and Jordan [8] algorithms, for the model of Gaussian mixing associative and also the error back propagation algorithm, by including the calculation of the descend gradient. This algorithm is briefly described along this section, more details and the complete algorithm is described at Magalhães et al. [10].

The modular network used is consisted by *K* MLP experts, with L_{esp} layers with q neurons in each layer. And also a gating network of the type MLP with L_{pas} layers with *q* neurons in each layer. The neurons activation functions in all networks can be linear or non-linear. Was chosen the same architecture to all experts, to simplify the implementation in hardware.

2.1.1 First Step

The first step of the algorithm is the calculation of *a priori* probability associated to the *i-th* layer neuron output of the gating network, when from *n-th* application example of training, obtained from (1)

$$g_i(n) = \frac{\exp\left(u_i^{L_{PAS}}(n)\right)}{\sum_{j=1}^{k} \exp\left(u_j^{L_{PAS}}(n)\right)} \qquad (1)$$

where $u_i^{(l)}(n)$ is the *i-th* output neuron of the *l-th* layer from the gating network.

2.1.2 Second Step

The second step of the algorithm is to obtain the values of *a posteriori* probabilities $h_i(n)$ associated to the output neuron *i* from the output of the gating network, when from the *n-th* application example of training, obtained from (2)

$$h_i(n) = \frac{g_i(n)\exp\left(-\frac{1}{2}\| d(n) - y_{e_i^k}(n)\|^2\right)}{\sum_{j=1}^{k} g_j(n)\exp\left(-\frac{1}{2}\| d(n) - y_{e_i^k}(n)\|^2\right)} \qquad (2)$$

where *d(n)* is the expected answer and $ye_i^{(k)}(n)$ is the answer provided by the neuron *i* from the layer *l* of the *k-th* expert for the example *n*.

2.1.3 Third Step

The third step is where are made increment in synaptic weights of the modular network with multiple layers. The synaptic weights from the networks experts are up-to-dated according to the equation (3)

$$w_{e_i^k j}^{(l)}(n+1) = w_{e_i^k j}^{(l)}(n) + \eta \delta_{e_i^k}^{(l)}(n) y_j^{(l-1)}(n) \tag{3}$$

where η is the learning rate, and $\delta_{ei}^{(l)}(k)$ the gradient for the output layer neurons and and y_j is the output of the *j* neuron of the *l-1* layer.

The synaptic weight increment from the gating network is done through (4)

$$a_{p_i j}^{l}(n+1) = a_{p_i j}^{l}(n) + \eta \delta_{p_i}^{l}(n) y_j^{l-1}(n) \tag{4}$$

To validate the algorithm we choose the problem of quality of energy, which is a interest point for energy concessionaire. The application implemented on the system architecture was a pattern classification for power lines disturbances. The methodology of the application and data files and information were the same present in Medeiros et al. [7] study. This application evaluated the performance of an intelligent system classifier, in this case a modular network, in electric disturbances classification.

The approach is done into four mains steps: getting the signal, pre-processing, definition and classification of descriptors. The first step, which comes to obtaining the electrical signals, has been carried out through the oscillograph network of São Francisco hydro Electric Company (CHESF) and also from the simulation via Transient Alternative Program (ATP). The network consists of 370 oscillographs operation with a sampling rate ranging between 20 and 256 samples/cycle. The signals used in this study were collected in voltage lines levels of 69, 230 and 500kV, with a rate of 128 samples/cycle during 14 cycles [13]. These steps are described with more details by Medeiros et al. in [7].

The pre-processing stage is to suggest descriptors that characterize the signs variations when diverted from a certain standard. The third step, which deals with the descriptors definition, is performed from the decomposition of signals from the previous step. Following obtaining the descriptors, four disturbances classes are defined: Voltage Sag, Voltage Swell, Harmonics and Transitories. The Final Step, the classification, is performed by the application of classifiers based on artificial neural networks. Several architectures were tested, as shown at Table 1.

To the classification step were used two sets of data, the first for the training at computer, and the second for validation at FPGA. The training set consisted by 800 patterns formed by the four disturbances classes.

To validate the modular network are used 344 input patterns with their respective expected responses, consisting only of data obtained from the oscillographs.

Table 1. Modular Neural Network Architecture.

Net	MOD-0	MOD-1	MOD-2	MOD-3
Number of Experts	3	3	3	3
Expert Architecture	10:3:4	10:5:4	10:10:4	10:15:4
Gating Architecture	10:5:4	10:5:4	10:10:4	10:15:4
Classification	*98,46%*	*99,48%*	*100%*	*100%*

From Table 1 it can be said that the modular neural network with the proposed algorithm reaches a high amount of accuracy, approximately 100%.

More details of the algorithm implementation, information about performance and a comparative with others neural net architectures are founded at Magalhães [10].

3 Multiple Processors Systems

The idea of parallel processing is not new. A parallel system is made by processing elements (PE) that work in cooperation to solve a problem [11].

Parallel systems can be classified, for example, by the data or instruction flow, using these criteria they could be divided by Flynn's taxonomy into Single Instruction Single Data (SISD), Multiple Instruction Single Data (MISD), Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD). Systems with multiple processors are members of the last class, which has *n* PE working in parallel, processing asynchronous tasks concurrently in order to, in a given time, complete the task.

The MIMD class can be subdivided into two subclasses, according to memory access, a system could use or not a shared memory. The main difference between them is that when using the shared memory, all PE have access to the same memory, while in the other each PE has its own memory. So we can see two paradigms for performing communication between the processors: first, the use of a shared memory; second, a message passing facility.

At the first paradigm, whereby more than one PE could access the same shared memory address to execute the write and read operations, is necessary to protect this block. For example, using a semaphore, this component not allows two processes access the same memory address simultaneously, avoiding a conflict [12].

The second paradigm is the message passing. In this case, a PE uses an interconnection network to send and receive messages, and so establish a communication with other PE. In this configuration each processor has its own memory that it is accessed only by itself. Bus, ring and mesh are examples of network topologies, which exist to build the interconnection of a multiprocessor system. The choice of which network will be used is made according with characteristics as: cost, performance and how many nodes exist.

In our case, we used a shared memory communication; a Nios II processor was chosen as our PE, All components are communicating though the Avalon bus. This is designed to connect on-chip processors and peripherals together in a system on programmable chip (SOPC). Peripherals, that use this bus, could be divided into master and slave the first is able to start data transfers, while the second only transfers data when requested. The Nios II Processor is an example of a peripheral master, while a shared memory is a peripheral slave. When more than one master interacts with a same slave it is necessary an arbiter with an arbitration to determine which master have access to it. The arbitration scheme used by the arbitrator is the round robin.

The Nios II is a soft processor developed by Altera and distributed together with its FPGAs. These are equivalent to a microcontroller, and are used in many different applications. The Nios II has a central processing unit (CPU), memory and peripherals on a single chip. This is a RISC processor for general use. Being a software processor, you can add and configure peripherals to the Nios II, according with the application. The core of the Nios II can be divided into three versions: economic, basic and fast. The developer will choose the most appropriate for their application. The fast core is designed for applications that require high performance. It has cache for data and instructions, which improve, for example, the performance of applications with a large amount of data. The basic version has no cache for data and its performance is about 40% smaller than the fast version, so it should be used in applications where high performance is not a necessary feature. The economic core is half size of the basic version. It has only the necessary functions to be used with Nios II instructions set. This core is used in applications where it is required a simple logic control.

To use the shared memory it is necessary a mutex component, this ensure a mutual exclusion (ME) coordinating the read and write operations. The mutex provide an atomic test-and-set operation that allows a processor to test if the mutex is available and if so, to acquire the mutex lock in a single operation. Without the mutex, a write operation would normally require the processor to execute two separate operations. To do that the mutex has two fields (registers). Each processor has a single identifier (ID). Each mutex has a VALUE field and OWNER field. The VALUE field is always accessible for a processor to read it. A read value of 0x0000 represents mutex availability. If the mutex is available the processor writes its ID in OWNER and a different value of 0x0000 in VALUE. Upon acquiring the mutex the processor performs the operation (write or read) and then finalizes releasing the mutex.

4 System Architecture and Results

Table 2. Configurations Tested.

Number of Processors	Space in the FPGA (LE)	Time to Generate (hh:mm:ss)
4	12170 (61%)	00:42:17
6	17845 (89%)	03:54:26
8	21146 (101%)	error

To design the architecture and obtaining the results was used a Nios II development kit with a FPGA Altera Cyclone EP1C20F400C7 within 20600 logic elements (LE). To generate the parallel architecture, was used an IBM-PC Pentium 4 3.0GHz with 1Gb of RAM, several configurations with different numbers of processors were generated, but the size of the FPGA was the limiting factor in defining their number in the system. Table 2 shows some of these configurations.

At Fig. 1, is possible to see, that the modular network, could be easily divide at small tasks. For example, each expert could be a task, and so implemented at a different processor, as well as the gating network and the sum function.

We made three kind of tests; first, the modular neural network implement one processor, in a serial version, second a parallel version using two processors, and the last one is a parallel version using four processors. We made tests with the message passing facility paradigm and the shared memory. We choose to use shared memory, because it has shown a better performance in this type of problem. At Fig. 2, we could see a diagram of the Modular Neural Network Parallel Algorithm divided in four small tasks and implemented at four processors. Each expert was implemented in a different processor, as well as the gating network with the sum function.

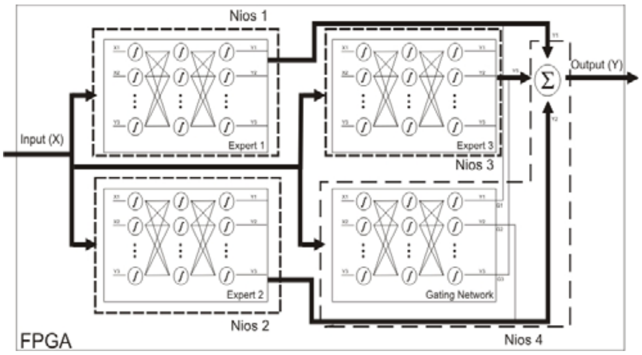


Fig. 2. Modular Neural Network Parallel Diagram.

We choose to implement the gating network and the sum function at the same processor, because at this way we need one less communication. So the parallel algorithm has two communications: first, one-to-all, where the master processor sends the inputs to all slaves (experts); second, all-to-one, that each slave sends its output to the master processor. In our case the master processor is where the gating network and the sum function were implemented. At the next tables, it is possible to see tree algorithms: Table 3 the serial algorithm, Table 4, at first column, the parallel algorithm implement at two processors, and at Table 4, at second column, the parallel algorithm implement at four processors.

Table 3. Serial Algorithm.

```
Begin Serial Algorithm P1
    1. Read inputs and write at the memory
    2. Calculate the Experts outputs
    3. Calculate the Gating Network output
    4. Calculate the General output
End Serial Algorithm P1
```

All the processors have the same program; the first instruction's program is to identify what processor is, to execute its part of the code. Each processor has its own memory and could access a shared memory. The modular neural network executed at the FPGA was the MOD-0 shown at Table 1, this network achieved 98,46% of classification accuracy. MOD-0 is a network with fewer variables so it could be executed even in a small FPGA, as the Cyclone.

Table 4. Parallel Algorithm.

Begin Parallel Algorithm P1	***Begin Parallel Algorithm P1***
1. Read inputs and write at the shared memory	1. Read inputs and write at the shared memory
2. Calculate the Gating Network output	2. Calculates the Gating Network output
3. Wait the Experts outputs	3. Wait the Experts outputs
4. Calculate the General output	4. Calculate the General output
End Parallel Algorithm P1	***End Parallel Algorithm P1***
	Begin Parallel Algorithm P2
Begin Parallel Algorithm P2	1. Wait P1 write the inputs at the shared memory
1. Wait P1 write the inputs at the shared memory	2. Calculate the Expert 1 output
2. Calculate the Experts Output	3. Write Expert 1 Results
3. Write Experts Results	***End Parallel Algorithm P2***
End Parallel Algorithm P2	***Begin Parallel Algorithm P3***
	1. Wait P1 write the inputs at the shared memory
	2. Calculate the Expert 2 output
	3. Write Expert 2 Results
	End Parallel Algorithm P3
	Begin Parallel Algorithm P4
	1. Wait P1 write the inputs at the shared memory
	2. Calculate the Expert 3 output
	3. Write Expert 3 Results
	End Parallel Algorithm P4

To measure the performance of the three algorithms we use a performance counter, which is a component that counts how many clocks a program, and a section of it, need to execute. The serial algorithm has three main functions, as seen in Table 3; a) calculate the experts outputs, b) calculate the gating network output, and c) calculate the global output. The time, which each one needs, is shown in percent, at Fig. 3. The time necessary to read an input is insignificant when compared with the others, so it was not considerate.

The function that needs more time is: experts output, so we decide to make parallel this function at the parallel algorithm. In this it is necessary a flag to syncronize the processors, because the global output only could be calculated after the others. Another function was created, the output experts wait. The algorithm was divided as follow: the master processor calculate the gating network and wait the results from slaves processors, so even the master processor finish with the gating network it needs to wait to calculate de global output. All slaves calculate the experts outputs and send the answers to master. As the algorithm only terminates after the calculation of the global output, the total time of this system is measured on the master.

Results with 2 processors are shown at Fig. 4, at this configuration one processor is the master, and the second processor is the slave, that calculate all the experts outputs. As the tasks of the slave needs more time than the master, this stay 26.41%

of time waiting the results from it. At the second configuration, in which has 4 processors, each slave calculate the output of one expert, results are shown at Fig. 5.

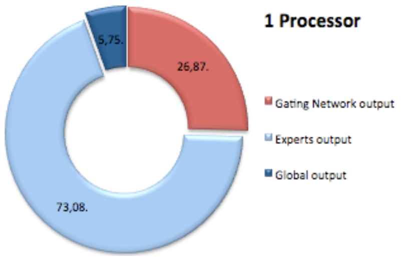


Fig. 3. Time of each function at the serial program.

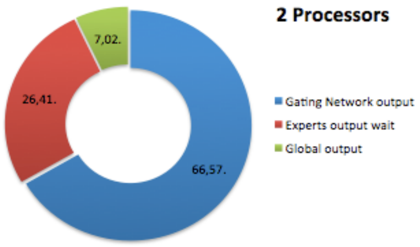


Fig. 4. Time of each function at the master processor at the parallel algorithm with 2 processors.

With 4 processors the master stays only 2,69% of time at idle. The computing time decrised at each improvement, this time, at number of clocks, is shown at Fig. 6.

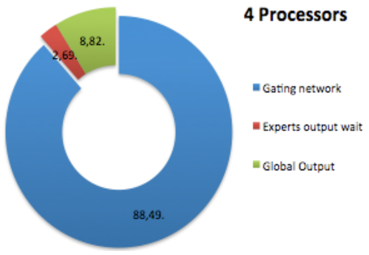


Fig. 5. Time of each function at the master processor at the parallel algorithm with 4 processors.

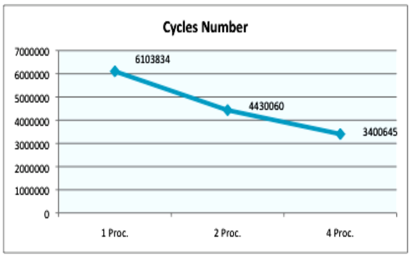


Fig. 6. Total time, meseaured at cycles, of each algorithm.

With the number of clocks of each algorithm, as shown in the figure above, it is possible to calculate the speed-up of the parallel algorithm, this is obtained using the expression (5).

$$G = \frac{T_S}{T_P} \tag{5}$$

where T_S is the serial execution time and T_P is the parallel execution time. The speed-up calculated was 1.47 for 2 processors, and 1.87 for 4 processors.

5 Conclusions

In this paper, the implementation of pattern classification for power lines disturbances using multiple processors on FPGA was made. The classification achieved at MOD-0 was 98.46%.

We use a small Cyclone Altera FPGA, which has performance restrictions, to implement the execution phase of the MOD-0. This operates at a frequency of 50MHz, and has less than half size of new FPGA, even then the best speed-up achieved was 1.87, in other words, 93.5% of improvement at execution time.

This application is portable to different FPGAs, and could have the number of processors easily increased due its scalability. Using a new one a better performance could be achieving increasing the number of NIOS II inside it, and so execute the MOD-1, MOD-2, MOD-3 networks and others applications more complex.

References

1. J. D. Melo, A. D. D. Neto, A. C. M. L. Albuquerque, R. L. S. Alves, “New Parallel Algorithms for Back-propagation Learning”, Proceedings of IJCNN’02, Internacional Joint Conference on Neural Networks Honolulu, USA 2002.
2. M. F. Medeiros Jr. and J. T. Oliveira and E. G. M. Lacerda and J.J.A.L. Leitão, “Disturbances Caracterization in Electric Energy Networks Using Second Generetion Wavelet Transform”, VI Industry Application Conference – VI Induscon, Joinville – Sc, 2004.
3. D. Optiz and R. Maclin, “Popular ensamble methods: an empirical study”, Journal of Artificial Intelligence Research, vol 11, pp. 169-198, 1999.
4. G. Valentini and F. Masulli, “Ensemble of learning machines”, Neural nets WIRN Vietri-02, Series Lectures Notes in Computer Sciences, Springer-Verlag, 2002.
5. B. A. Souza and N. S. D. Brito and W. L. A. Neves and R. B. Lima and S. S. B. Silva, “Comparison between backpropagation and RPROP algorithms applied to fault classification in transmission lines”, Proceedings of The International Joint Conference on Neural Networks & International Conference on Fuzzy Systems, Budapeste, 2004.
6. R. N. Machado and U. H. Bezerra and E. G. Pelaes “Classification of signal with voltage disturbance by means of wavelet transform and intelligent computacional techniques”, WSEAS Transactions on Power Systems, v. 1, pp. 1538-1532, 2006.
7. M. F. Medeiros Jr and C. K. S. Santos and J. T. oliveira and P. S. M. Pires and J. D. Melo and A. D. D. neto and J. J. A. L. Leitão, “Influence of signal processing in the Efficiency of algorithms Based on Neural Networks for Disturbanc Classification”, Computacional Intelligence in Image and Signal Processing, pp. 95 -100, 2007.
8. J. Castillo and P. Huerta and C. Pedraza and J. I. Martínez, “A Self-Reconfigurable Multimedia Player on FPGA.” IEEE International Conference on Reconfigurable Computing and FPGA’s – (ReConFig’ 06) – San Luis Potosí. México Pp. 1 – 6.
9. R. A. Jacobs and M. I. Jordan, “Adaptive mixtures of local expert”, Neural Computation, volume 3, pp 79 – 87, 1991.
10. R. M. Magalhães and C. K. S. Santos and J. D. Melo and M. F. Medeiros, A. D. D. Neto, “Application of a hybrid Algorithm in the Modular Neural Nets Trainning with Multilayers Specialists in Electric Disturbance Classification”, Proceedings of International Conference on Intelligent Engineering Systems, pp. 121-125, 2008.
11. K. Hwang and F. A. Briggs. “Computer Architecture and Parallel Processing.”, Singapore: McGraw-Hill, 846p. McGraw-Hill series in computer organization and architecture, 1985.
12. Hübner, M; Paulsson, K; Becker, J. (2005), ”Parallel and Flexible Multiprocessor System-On-Chip for adaptive Automotive Applications based on Xilinx MicroBlaze Soft-Cores”. 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS’05) – Denver, Colorado, USA. Pp. 149 – 154, 2005.

A Design for Real-time Neural Modeling on the GPU Incorporating Dendritic Computation

Tyler W. Garaas[1], Frank Marino[1], Halil Duzcu[2] and Marc Pomplun[1]

[1] University of Massachusetts Boston
100 Morrissey Boulevard, Boston, MA 02125-3393, U.S.A.
[2] Middle East Technical University, İnönü Bulvarı, 06531, Ankara, Turkey

Abstract. Recent advances in neuroscience have underscored the role of single neurons in information processing. Much of this work has focused on the role of neurons' dendrites to perform complex local computations that form the basis for the global computation of the neuron. Generally, artificial neural networks that are capable of real-time simulation do not take into account the principles underlying single-neuron processing. In this paper we propose a design for a neural model executed on the graphics processing unit (GPU) that is capable of simulating large neural networks that utilize dendritic computation inspired by biological neurons. We subsequently test our design using a neural model of the retinal neurons that contribute to the activation of starburst amacrine cells, which, as in biological retinas, use dendritic computational abilities to produce a neural signal that is directionally selective to stimuli moving centrifugally.

1 Introduction

As with most research topics, neural modeling has broadened into a spectrum of methodologies that sometimes use the terms artificial neural networks, computational neuroscience, and brain models to illustrate methodological differences. The authors of [1] have chosen the terms *realistic brain models* and *simplified brain models* to illustrate two sides of the research spectrum. Realistic brain models refer to models that go to painstaking lengths to model the individual components of neurons and their assemblies. In these models, the goal is often directed toward gaining greater insight into actual brain function [2]. Indeed, this is a very active avenue of research in both the neuroscience and computational neuroscience fields.

Unlike their counterpart, simplified brain models, often implemented as *artificial neural networks* (ANNs), are usually directed toward computing meaningful information. Their simplified nature is both a distinct advantage and disadvantage over realistic brain models, as it is the conceptual and computational intractability that has hindered the use of realistic brain models as functional entities. Despite the large range of successful applications of ANNs, future networks that attempt to solve complex problems such as robust object recognition [3] or modeling complex behaviors [4] will likely require more realistic organization principles.

If neural networks are to be realized in a more biologically realistic manner, the two aforementioned hindrances will need to be overcome. The first, conceptual intractability, is being slowly broken apart by a large number of neuroscientists such as those previously referenced. Their progress has led to many new ideas and models regarding the functioning of individual neurons. A major insight that has emerged from these studies involves the role of the single neuron in the computational abilities of neural networks –both biological and artificial [2, 5]. In particular, the structural organization of the neuron's *dendrite* (the part of the neuron that receives signals from other neurons) has become an important concept in both the theory of biological neuron functioning [2, 5-7] and computational studies [8]. Models that do not take into account the physical structure of the neuron are in effect using *point neurons*. Point neurons are named as such to illustrate the lack of dendrites where afferent neurons instead synapse directly onto the soma (cell body).

One biological phenomenon that has been attributed to interactions between neurons synapsing at proximal locations on a dendrite is the directionally selective activation of starburst amacrine cells [9]. Euler et al. [9] demonstrated that the signal, which responds vigorously for stimuli moving centrifugally –i.e., away from the soma–, is the result of local computations that take place on the dendrites of the cell. Following this result, Tukker et al. [10] created a realistic computational model that showed the directionally selective signal found at the distal tips of the dendrites could be accounted for by an interaction between a temporally delayed global signal and local synaptic input.

The second hindrance, computational intractability, is a result of the large number of calculations needed to model a realistic neuron. One approach that is being used to overcome computational insufficiencies in highly parallel applications, such as neural modeling, is to use the graphics processing unit (GPU) [11], which is the primary computational unit integrated into present-day computer graphics cards.

In this paper, we present a neural network design that has been crafted for execution on the GPU. The design achieves real-time computational abilities while preserving potentially crucial features of realistic brain models such as dendritic computing. To demonstrate the neural network design, we implement a sample network that simulates the subset of the retinal circuitry responsible for generating the directionally selective signal in the starburst amacrine cells.

2 GPU Processing

In recent years, the computational abilities of certain systems have seen enhanced growth due to the expansion of parallel systems such as cluster computing and distributed computing. Another highly parallel paradigm that has recently been exploited by computationally hungry scientists is the GPU, which is currently being used for image processing, computer vision, signal processing, video encoding, and ray tracing, among others [11]; applications such as these have been given the acronym GPGPU for general-purpose computation using graphics hardware.

2.1 Brief Overview of GPU Architecture

The massive computational power underlying the GPU comes from its parallel architecture, which is implemented using a computational unit known as a *stream* *processor*. Essentially, a stream processor is a highly restricted form of a processor core; whereas processor cores are able to perform a wide variety of complex tasks, stream processors use a specialized instruction set to perform the tasks that are repeatedly executed during computer graphics rendering. By performing only a handful of tasks, the GPU can pack hundreds of stream processors into a single GPU, as opposed to the eight processor cores available in modern CPUs at the time of writing.

Given the restricted nature of stream processors, applications that wish to exploit the computational advantages of the GPU must adhere to a narrow flow of execution. This flow is divided into four primary steps: vertex operations, primitive assembly, rasterization, and fragment operations. All programs executed on the GPU must perform all four steps; however, in many GPGPU applications, the first three steps are executed at a bare minimum to support the bulk of the computation, which takes place at the final stage. Those interested in the details of the first three stages are encouraged to visit a community website dedicated to GPGPU programming [12].

The fragment operations that support the bulk of GPGPU computations are performed by a simple program designed to execute on the GPU known as a *fragment shader*. Fragment shaders perform a series of operations that manipulate one pixel of data per execution. However, since there are hundreds of stream processors, many millions of pixels of data can be processed in a very short period of time.

Data used by GPGPU applications must also conform to computer graphics constructs which use images known as *textures* to store data. In traditional computer graphics applications, a texture stores visual attributes not suited for –or too computationally expensive for– representation by geometry, such as the clothes of a character or the asphalt of a highway.

2.2 Neural Networks on the GPU

Many ANNs involve a highly parallel design that is well suited for implementation on the GPU. Consequently, a number of researchers have taken advantage of this to achieve notable gains in execution time [13-15]. For instance, Bernhard and Keriven [13] were able to achieve a 5 to 20 fold increase in performance over a CPU implementation while simulating spiking neural networks for image segmentation. Gobron et al. [14] use the GPU to model the retina using cellular automata, and Woodbeck et al. [15] use the GPU to implement a model of the processing that takes place in the primary visual cortex. However, in each of these instances of neural network processing, simple point-neurons were used to perform the pertinent computations, which will likely be insufficient to for complex tasks such as robust object recognition.

3 Neural Network Design

3.1 Single Neuron Model

As mentioned previously, researchers now believe the physical organization of synapses plays a key role in the processing of information by neurons. In the design presented here, we take into account the organization of afferent synapses to facilitate some of the mechanisms that underlie the computational power of biological neurons. Figure 1 shows two neurons that can be simulated using the present model. The key thing to notice is the labeling of dendrite segments, which permits local computations to take place in individual segments. London and Häusser [5, pp. 509] note, “Because the branch points in the dendritic tree can be seen as summing up the current in individual branches, ... the whole dendrite can implement complex functions.” The addition of this type of organization will allow the use of what London and Häusser refer to as the *dendritic toolkit* [5]. In the sample network we use this dendritic toolkit to compute a direction selective signal in the dendrites of a simulated starburst amacrine cell.

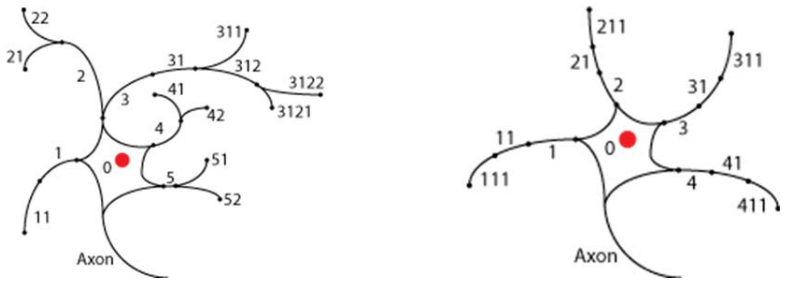


Fig. 1. Illustration of potential neurons in the present model.

3.2 Single Neuron Creation

The single neuron model described in the previous section allows inputs to be grouped together on a single dendrite segment so that local computations can take place independently. However, it may not be entirely clear how the dendrite segments can be generated or how afferent synapses can be connected to each segment. Consequently, we have included the pseudocode for a recursive function that generates the dendrite branching patterns from a *branch code*. For example, the branch code that is used to generate the left neuron in Figure 1 is given on the first line of pseudocode, and the branch code to generate the right neuron is `41110111011101110`.

Essentially, each digit in the branch code, which can be stored in string form, represents the number of new segments that are to be generated from the current location. For example, ‘2’ represents a binary split, ‘1’ represents a single segment, and ‘0’ represents the end of a segment. When a split is encountered, the subsequent digit(s) is used to generate the first segment of the split until that branch is terminated with a ‘0’ at which point the following digit(s) is used to generate the second segment of the most recent split, and so on until all segments have been terminated.

```
branchCode = “52101021010121021010210101 10”
SetupDendriteSegment(int codeIndex)
   if branchCode[codeIndex] equals ‘0’
      delete digit from branchCode at codeIndex and return
   DendriteSegment d = new DendriteSegment
   while SynapsesNeeded() equals true
      Add GetNextSynapse() to d
   for i = Integer of branchCode[codeIndex] to 0
      SetupDendriteSegment(codeIndex+1)
   delete digit from branchCode at codeIndex and return
End SetupDendriteSegment
```

The `SetupDendriteSegment` function can generate a complex branching pattern from a string of digits; however, it still must be decided how afferent inputs will be connected to the individual dendrite segments. Since this is different for each type of neuron, a general procedure is described which invokes two undefined functions, `SynapsesNeeded` and `GetNextSynapse` which are left to the reader to implement as needed by their application.

One final item needs to be handled before the segments are complete, which is the fact that the local membrane potentials of neighboring segments should be part of the local computation that takes place on each individual dendrite segment. This can be solved by simply treating the neighboring segments as synapses and identifying these with a unique synapse type (described in the following section).

3.3 Data Storage

In the present model, functionally identical neurons are arranged into a single layer which forms the functional unit of the network. Each layer of simulated neurons in the network requires three textures to be maintained throughout execution: a *neuron texture*, a *dendrite texture*, and a *synapse texture*. Individual pixels in the neuron texture store the activation (i.e., membrane potential; A^N) for each individual neuron as well as an index to the first dendrite segment of the neuron (I^D) stored in the dendrite texture. Pixels in the dendrite texture store the local membrane potential of each dendrite segment (A^D), an index to the first synapse of the segment (I^S) stored in the synapse texture, and a weight used to moderate how much local potential is transferred from the segment (W^D). Finally, each pixel in the synapse texture stores an index to the afferent membrane potential that is being transferred by the synapse (I^A); for instance, bipolar cells in the sample network receive input from the photoreceptor neurons, which will be indexed by the synapse texture of the bipolar cell's synapses. However, as with biological neurons, synapses in the model can also receive their input from a dendrite segment of another neuron. The synapse texture also contains a weight used to modify the synapse strength (W^S) and the type of the synapse (T) which determines the texture that is indexed by the synapse. Figure 2 illustrates a summary of the data model described above.

3.4 Model Execution

As mentioned in the previous section, functionally identical neurons are grouped into layers. Each layer is executed by two fragment shaders each time a neuron's activations are calculated. The first fragment shader computes the local activation function of individual dendrite segments. This shader executes by beginning at the first indexed synapse in the synapse texture and iterating over subsequent synapses until a blank synapse is encountered –denoted by a -1 in I^A. The membrane potential calculated in this way is then stored in the G component of the dendrite texture. The second shader computes the global activation function of the neurons in the network. This shader works in a manner very similar to the first shader by iterating over the dendrite segments directly connected to the neuron soma, and the activation of the neuron is stored in the G component of the neuron texture.

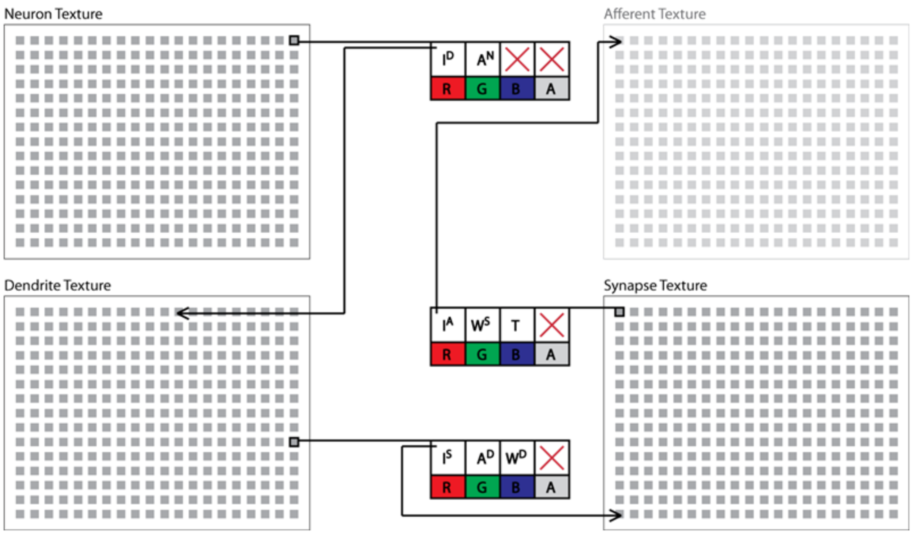


Fig. 2. Data model for the neural network design. Each pixel (grey square) is used to store the various indexes and parameters used to compute the activations of the neurons and dendrites in the network.

4 Sample Network

The sample network presented hereafter is used to demonstrate how the organization principles previously described can be implemented to achieve both function and efficiency. To this end, the following network will be used to model a subset of the retinal circuitry responsible for the directionally selective signal found in the distal dendrite branches of the starburst amacrine cells. In particular, we model the photoreceptors (short-, medium-, & long-wavelength cones; implemented in the model using individual red, green, and blue color channels), horizontal cells (H1 and H2), on-center bipolar cells (short-, medium-, & long-wavelength cells), and starburst amacrine cells. The local membrane potentials of the network demonstrate that the computation of the centrifugal direction selective signal is very similar to that of biological neurons [9, 10].

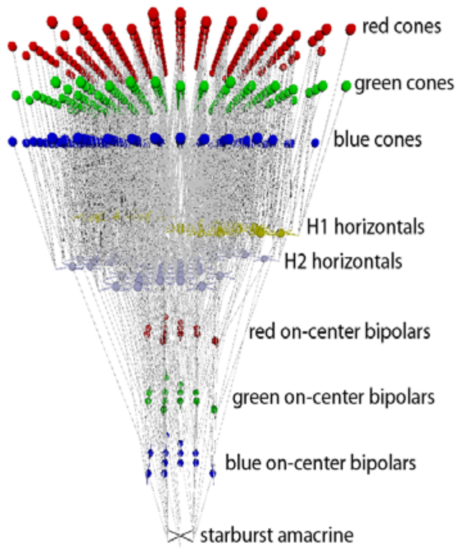
4.1 System Overview

Input to the network is achieved using a standard webcam setup that captures video at 30 frames per second (FPS). After each frame is captured, it is transferred to video memory on the graphics card using the OpenGL API. Fragment shaders, which encompass nearly all of the network processing and implementation, are implemented using OpenGL's high level shading language, GLSL. The neuron, dendrite, and synapse textures are created prior to execution using an extension of the methods detailed in previous sections. All computations were executed on a desktop computer running Windows XP with a 2.6 Ghz AMD 64 bit dual-core processor with 4 GBs or RAM and a Nvidia GTX 280 graphics card.

4.2 Network Organization

The wiring of photoreceptors, horizontal cells, and on-center bipolar cells is modeled directly from the biological wiring described in [9, 10, 16, 17]. Figure 3 (left) illustrates the connections that exist between a single starburst amacrine cell and the cells that contribute to its activation; essentially, the neurons shown for the *classic receptive* field for a single starburst amacrine cell. Dendritic branching of horizontal and bipolar cells incorporates only single dendrite branch segments –e.g., a branch code of `410101010`. Bipolar cells compute a contrast modulated activation through an antagonistic center-surround organization of its afferent synapses from cones (center) and horizontal cells (surround). For the sake of brevity, the equations used to compute neural activations are not presented. However, the neural activations follow very closely those of their biological counterparts as described by Dowling [17], and are similar in effect to those of [18].

Starburst amacrine cell dendrites follow the basic design present in Figure 1. This dendrite branching pattern, though highly simplified, still preserves the necessary geometric relationship that allows the computation of the centrifugal-selective signals as described by Tukker et al. [10]. As dendrite segments of the starburst amacrine cell move progressively farther from the soma, inputs to these segments are selectively received from bipolar cells that are more distant from the starburst cell.



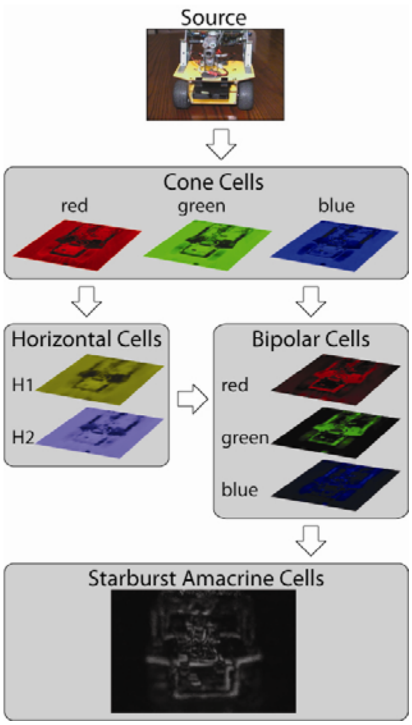


Fig. 3. Connections and data flow in the sample network. (left) A 3-dimensional rendering of the actual connections that contribute to the activation of a single starburst amacrine cell in the sample network. Illustrated neurons represent the classic receptive field of the starburst neuron. (right) Source image and the activations of each layer in the network. Lighter portions represent higher activations whereas the color represents the relative contribution by red, green, and blue components of the source signal.

4.3 Network Results

The sample network consists of nine layers of neurons, more than 225,000 individual neurons, and more than 2 million synapses. Despite its size, the activations of every neuron in the network can be computed in 7 ms or, put another way, the network operates at roughly 142 pulses per second (1 pulse = 1 computation of all neuron activations). In preliminary versions of the network, the GPU version outperformed an analogous, multi-threaded CPU version by a factor of approximately 20 when executed on a 2.4 Ghz quad-core processor.

The activations of the individual layers in the network are illustrated in Figure 3 (right). The activations of all layers, with the exception of the starburst amacrine layer, are illustrated during a single, similar pulse step. The activations of the starburst amacrine layer, however, are from a pulse that is many steps into the future relative to the other layers, which demonstrates the motion of the robot pictured. Although the starburst amacrine cells are directionally selective, their highly overlapped nature instead results in activations similar to image subtraction from traditional image processing techniques. This result in itself could be achieved with a simpler network [c.f., 18]; however, starburst amacrine cells form a crucial input for the computationally more complex directionally selective ganglion cells [9]. As such, this network represents a significant first step in the creation of a larger, more complex network dedicated to modeling the complex visual processes that contribute to our own remarkable visual abilities.

5 Discussion

In the present work we have demonstrated a novel neural network design that combines biologically realistic –and potentially computationally crucial– mechanisms that can be executed in real-time by taking advantage of the highly parallel organization of modern GPUs. This design was then demonstrated by modeling the subset of the retinal neural circuitry that is responsible for computing a directionally-selective signal in the starburst amacrine cell. As in its biological form, the signal computed in the cell is the result of the physical organization of the afferent input on a starburst cell's dendrites.

The sample network, which consists of more than 225k neurons, can be computed at a rate that is nearly 5x the frame rate of the incoming signal. As such, this sample network could already be extended to include much more sophisticate motion processing by including layers subsequent to starburst cells such as directionally selective ganglion cells and neurons in extra-striate areas and the middle temporal area, which are known to be very important to motion processing in primate vision [19]. Additional cortical mechanisms important for primate vision have been proposed to directly rely on computations that take place in the dendrites [e.g., 20, 21]. The single neuron model that is introduced here could be used directly to implement such proposed complex neural mechanisms in a computationally efficient manner. Future work will be directed toward building more complex visual abilities of the network, and, when necessary, utilizing the dendritic toolkit that is supported by the current neural network design to implement such abilities.

References

1. Sejnowski, T. J., Koch, C. & Churchland, P. S. (1988). Computational neuroscience. *Science*, 241, 1299-1306.
2. Koch, C. & Segev, I. (2000). The role of single neurons in information processing. *Nature Neuroscience*, 3, 1171-1177.
3. Gray, C. M., König, P., Engel, A. K. & Singer, W. (1989). Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. *Nature*, 338, 334-347.
4. Vaadia, E., Haalman, I., Abeles, M., Bergman, H., Prut, Y., Slovin, H. & Aertsen, A. (1995). Dynamics of neuronal interactions in monkey cortex in relation to behavioural events. *Nature*, 373, 515-518.
5. London, M. & Häusser M. (2005). Dendritic computation. *Annual Review of Neuroscience*, 28, 503-532.
6. Agmon-Snir, H., Carr, C. E. & Rinzel, J. (1998). The role of dendrites in auditory coincidence detection. *Nature*, 393, 268-272.
7. Mainen, Z. F. & Sejnowski, T. J. (1996). Influence of dendritic structure on firing pattern in model neocortical neurons. *Nature*, 382, 363-366.
8. Segev, I & Rall, W. (1988). Computational study of an excitable dendritic spine. *Journal of Neurophysiology*, 60, 499-523.
9. Euler, T., Detwiler, P. B. & Denk, W. (2002). Directionally selective calcium signals in dendrites of starburst amacrine cells. *Nature*, 418, 845-852.
10. Tukker, J. J., Taylor, W. R. & Smith, R. G. (2004). Direction selectivity in a model of the starburst amacrine cell. *Visual Neuroscience*, 21, 611-625.
11. Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E. & Purcell, T. J. (2007). A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26, 80-113.
12. *GPGPU*. Retrieved April 01, 2009 from http://www.gpgpu.org
13. Bernhard, F. & Keriven, R. (2006). Spiking Neurons on GPUs. *International Conference on Computation Science. Workshop general purpose computation on graphics hardware (GPGPU): Methods algorithms and applications*, Readings, U.K.
14. Gobron, S., Devillard, F. & Heit, B. (2007). Retina simulation using cellular automata and GPU programming. *Machine Vision and Applications*, 18, 331-342.
15. Woodbeck, K., Roth, G. & Chen, H. (2008). Visual cortex on the GPU: Biologically inspired classifier and feature descriptor for rapid recognition. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, AK, U.S.A.
16. Dacey, D. M. (2000). Parallel pathways for spectral coding in primate retina. *Annual Review of Neuroscience*, 23, 743-775.
17. Dowling, J. E. (1987). *The Retina: An Approachable Part of the Brain*. Cambridge, MA, USA. Belknap Press.
18. Garaas, T. W. & Pomplun, M. (2007). Retina-inspired visual processing. *Proceedings of BIONETICS, Workshop on Computing and Communications from Biological Systems: Theory and Applications (CCBS)*. Budapest, Hungary.
19. Blake, R., Sekuler, R., & Grossman, E. (2003). Motion processing in human visual cortex. In J H Kaas and C E Collins (Eds.), *The Primate Visual System.* Boca Raton: CRC Press.
20. Mel, B. W., Ruderman, D. L., & Archie, K. A. (1998). Translation-invariant orientation tuning in visual "complex" cells could derive from intradendritic computations. *The Journal of Neuroscience*, 18, 4325-4334.
21. Barlow, H. (1996). Intraneuronal information processing, directional selectivity and memory for spatio-temporal sequences. *Network*, 7, 251-259.

Minimal Architecture and Training Parameters of Multilayer Perceptron for its Efficient Parallelization

Volodymyr Turchenko and Lucio Grandinetti

Department of Electronics, Informatics and Systems, University of Calabria
via P. Bucci 22B, 87036, Rende (CS), Italy
vtu@si.deis.unical.it,lugran@unical.it

Abstract. The development of a parallel algorithm for batch pattern training of a multilayer perceptron with the back propagation algorithm and the research of its efficiency on a general-purpose parallel computer are presented in this paper. The multilayer perceptron model and the usual sequential batch pattern training algorithm are theoretically described. An algorithmic description of the parallel version of the batch pattern training method is introduced. The efficiency of the developed parallel algorithm is investigated by progressively increasing the dimension of the parallelized problem on a general-purpose parallel computer NEC TX-7. A minimal architecture for the multilayer perceptron and its training parameters for an efficient parallelization are given.

1 Introduction

Artificial neural networks (NNs) have excellent abilities to model difficult nonlinear systems. They represent a very good alternative to traditional methods for solving complex problems in many fields, including image processing, predictions, pattern recognition, robotics, optimization, etc [1]. However, most NN models require high computational load, especially in the training phase (up to days and weeks). This is, indeed, the main obstacle to face for an efficient use of NNs in real-world applications. Taking into account the parallel nature of NNs, many researchers have already focused their attention on their parallelization [2-4]. Most of the existing parallelization approaches are based on specialized computing hardware and transputers, which are capable to fulfill the specific neural operations more quickly than general-purpose parallel and high performance computers. However computational clusters and Grids have gained tremendous popularity in computation science during last decade [5]. Computational Grids are considered as heterogeneous systems, which may include high performance computers with parallel architecture and computational clusters based on standard PCs. Therefore, existing solutions for NNs parallelization on transputer architectures should be re-designed. Parallelization efficiency should be explored on general-purpose parallel and high performance computers in order to provide an efficient usage within computational Grid systems.

Many researchers have already developed parallel algorithms for NNs training on weights (connections), neuron (node), training set (pattern) and modular levels [6-10]. The first two levels are a fine-grain parallelism and the second two levels are a coarse-grain parallelism. Connection parallelism (parallel execution on sets of weights) and node parallelism (parallel execution of operations on sets of neurons) schemes are not efficient while executing on a general-purpose high performance computer due to high synchronization and communication overhead among parallel processors [10]. Therefore coarse-grain approaches of pattern and modular parallelism should be used to parallelize NNs training on general-purpose parallel computers and computational Grids [9]. For example, one of the existing implementation of the batch pattern back propagation (BP) training algorithm [6] has efficiency of 80% while executing on a 10 processors of transputer TMB08. However, the efficiency of this algorithm on general-purpose high-performance computers is not researched yet.

The goal of this paper is to research the parallelization efficiency of parallel batch pattern BP training algorithm on a general-purpose parallel computer in order to form the recommendations for further usage of this algorithm on heterogeneous Grid system.

2 Architecture of Multilayer Perceptron and Batch Pattern Training Algorithm

It is expedient to research parallelization of multi-layer perceptron (MLP) because this kind of NN has the advantage of being simple and provides good generalizing properties. Therefore it is often used for many practical tasks including prediction, recognition, optimization and control [1]. However, parallelization of an MLP with the standard sequential BP training algorithm does not provide efficient parallelization due to high synchronization and communication overhead among parallel processors [10]. Therefore it is expedient to use the batch pattern training algorithm, which updates neurons' weights and thresholds at the end of each training epoch, i.e. after the presentation of all the input and output training patterns, instead of updating weights and thresholds after the presentation of each pattern in the usual sequential training mode.

The output value of a three-layer perceptron (Fig. 1) can be formulated as:

$$y = F_3\left(\sum_{j=1}^{N} w_{j3}\left(F_2\left(\sum_{i=1}^{M} w_{ij}x_i - T_j\right)\right) - T\right) \tag{1}$$

where N is the number of neurons in the hidden layer, w_{j3} is the weight of the synapse from neuron j of the hidden layer to the output neuron, w_{ij} are the weights from the input neurons to neuron j in the hidden layer, x_i are the input values, T_j are the thresholds of the neurons of the hidden layer and T is the threshold of the output neuron [1, 11]. In this study the logistic activation function $F(x) = 1/(1+e^{-x})$ is used for the neurons of the hidden (F_2) and output layers (F_3), but in general case these activation functions could be different.

The batch pattern BP training algorithm consists of the following steps [11]:

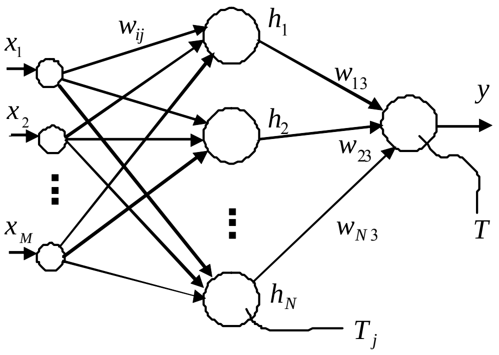


Fig. 1. The structure of a three-layer perception.

1. Set the desired error (Sum Squared Error) SSE=$E_{\min}$ and the number of training iterations t;
2. Initialize the weights and the thresholds of the neurons with values in range (0…0.5) [12];
3. For the training pattern pt:
   - 3.1. Calculate the output value $y^{pt}(t)$ by expression (1);
   - 3.2. Calculate the error of the output neuron $\gamma_3^{pt}(t) = y^{pt}(t) - d^{pt}(t)$, where $y^{pt}(t)$ is the output value of the perceptron and $d^{pt}(t)$ is the target output value;
   - 3.3. Calculate the hidden layer neurons' error $\gamma_j^{pt}(t) = \gamma_3^{pt}(t) \cdot w_{j3}(t) \cdot F_3'(S^{pt}(t))$, where $S^{pt}(t)$ is the weighted sum of the output neuron;
   - 3.4. Calculate the delta weights and delta thresholds of all perceptron's neurons and add the result to the value of the previous pattern $s\Delta w_{j3} = s\Delta w_{j3} + \gamma_3^{pt}(t) \cdot F_3'(S^{pt}(t)) \cdot h_j^{pt}(t)$, $\quad s\Delta T = s\Delta T + \gamma_3^{pt}(t) \cdot F_3'(S^{pt}(t))$, $s\Delta w_{ij} = s\Delta w_{ij} + \gamma_j^{pt}(t) \cdot F_2'(S_j^{pt}(t)) \cdot x_i^{pt}(t)$, $\quad s\Delta T_j = s\Delta T_j + \gamma_j^{pt}(t) \cdot F_2'(S_j^{pt}(t))$, where $S_j^{pt}(t)$ and $h_j^{pt}(t)$ are the weighted sum and the output value of the j hidden neuron respectively;
   - 3.5. Calculate the SSE using $E^{pt}(t) = \frac{1}{2}\left(y^{pt}(t) - d^{pt}(t)\right)^2$;
4. Repeat the step 3 above for each training pattern pt, where $pt \in \{1,...,PT\}$, PT is the size of the training set;
5. Update the weights and thresholds of neurons using $w_{ij}(PT) = w_{ij}(0) - \alpha(t) \cdot s\Delta w_{ij}$, $T_j(PT) = T_j(0) + \alpha(t) \cdot s\Delta T_j$, where $\alpha(t)$ is the learning rate;
6. Calculate the total SSE $E(t)$ on the training iteration t using $E(t) = \sum_{pt=1}^{PT} E^{pt}(t)$;
7. If $E(t)$ is greater than the desired error $E_{\min}$ then increase the number of training iteration to $t+1$ and go to step 3, otherwise stop the training process.

3 Parallel Batch Pattern Back Propagation Training Algorithm

It is obvious from analysis of the batch pattern BP training algorithm in Section 2 above, that the sequential execution of points 3.1-3.5 for all training patterns in the training set could be parallelized, because the sum operations $s\Delta w_{ij}$ and $s\Delta T_j$ are independent of each other. For the development of the parallel algorithm it is necessary to divide all the computational work among the *Master* (executing assigning functions and calculations) and the *Slaves* (executing only calculations) processors.

The algorithms for *Master* and *Slave* processors functioning are depicted in Fig. 2. The *Master* starts with definition (i) the number of patterns *PT* in the training data set and (ii) the number of processors *p* used for the parallel executing of the training algorithm. The *Master* divides all patterns in equal parts corresponding to number of the *Slaves* and assigns one part of patterns to himself. Then the *Master* sends to the *Slaves* the numbers of the appropriate patterns to train.

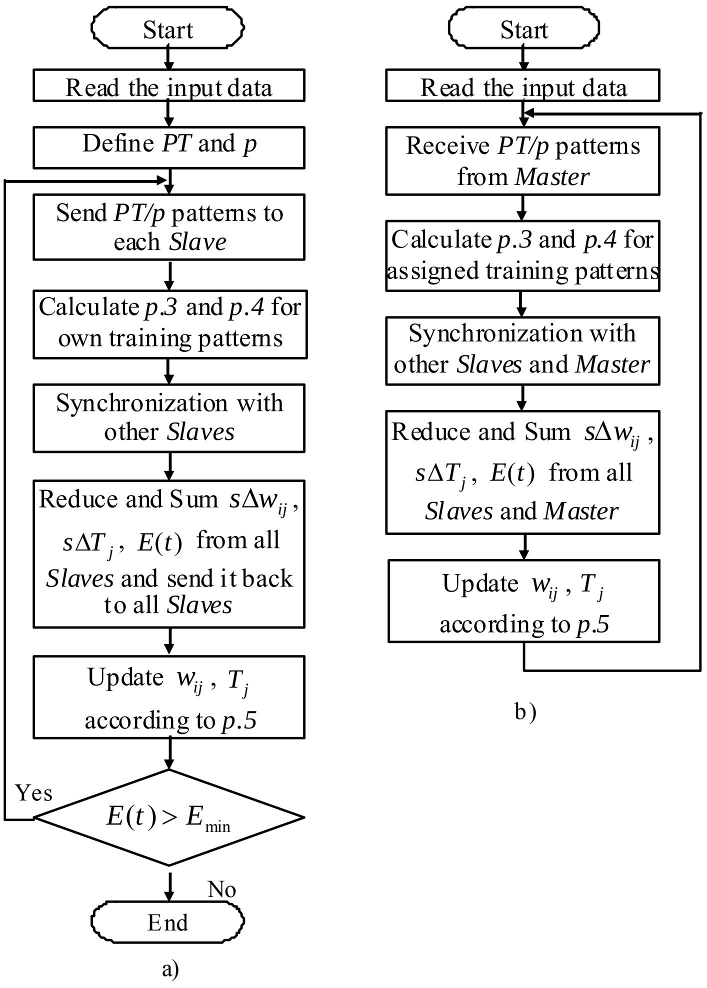
Each *Slave* executes the following operations for each pattern *pt* among the *PT/p* patterns assigned to him:

- calculate the points 3.1-3.5 and 4, only for its assigned number of training patterns. The values of the partial sums of delta weights $s\Delta w_{ij}$ and delta thresholds $s\Delta T_j$ are calculated here;
- calculate the partial SSE for its assigned number of training patterns.

After processing all its assigned patterns, each *Slave* waits for the other *Slaves* and the *Master* at the synchronization point. At the same time the *Master* computes the partial values of $s\Delta w_{ij}$ and $s\Delta T_j$ for its own (assigned to himself) number of training patterns.

The global operations of reduction and summation are executed just after the synchronization point. Then the summarized values of the $s\Delta w_{ij}$ and $s\Delta T_j$ are sent to all the processors working in parallel. Using a global reducing operation and simultaneously returning the reduced values back to the *Slaves* allows a decrease of the time overhead in the synchronization point. Then the summarized values of $s\Delta w_{ij}$ and $s\Delta T_j$ are placed into the local memory of each processor. Each *Slave* and the *Master* use these values $s\Delta w_{ij}$ and $s\Delta T_j$ in order to update the weights and thresholds according to the point 5 of the algorithm. These updated weights and thresholds will be used in the next iteration of the training algorithm. As the summarized value of $E(t)$ is also received as a result of the reducing operation, the *Master* decides whether to continue the training or not.

The software routine is developed using the C programming language with the standard MPI library. The parallel part of the algorithm starts with the call of the *MPI_Init()* function. The parallel processors use the synchronization point *MPI_Barrier()*. The reducing of the deltas of weights $s\Delta w_{ij}$ and thresholds $s\Delta T_j$ is provided by function *MPI_Allreduce()*, which allows to avoid an additional step for sending back the updated weights and thresholds from the *Master* to each *Slave*. Function *MPI_Finalize()* finishes the parallel part of the algorithm.



Start
Read the input data
Define PT and p
Send PT/p patterns to each $Slave$
Calculate $p.3$ and $p.4$ for own training patterns
Synchronization with other $Slaves$
Reduce and Sum $s\Delta w_{ij}$, $s\Delta T_j$, $E(t)$ from all $Slaves$ and send it back to all $Slaves$
Update w_{ij}, T_j according to $p.5$
$E(t) > E_{\min}$
Yes
No
End
a)
Start
Read the input data
Receive PT/p patterns from $Master$
Calculate $p.3$ and $p.4$ for assigned training patterns
Synchronization with other $Slaves$ and $Master$
Reduce and Sum $s\Delta w_{ij}$, $s\Delta T_j$, $E(t)$ from all $Slaves$ and $Master$
Update w_{ij}, T_j according to $p.5$
b)

Fig. 2. The algorithms of the *Master* (a) and the *Slave* (b) processors.

4 Experimental Researches

Our experiments were carried out on a parallel supercomputer NEC TX-7, located in the Center of Excellence of High Performance Computing, University of Calabria, Italy (www.hpcc.unical.it). NEC TX-7 consists in 4 identical units. Each unit has 4 Gb RAM, 4 64-bit processors Intel Itanium2 with a clock rate of 1 GHz. This 16th-processor computer with 64 Gb of total RAM has a performance peak of 64 GFLOPS. The NEC TX-7 is functioning under the Linux operation system.

As shown in [12], the parallelization efficiency of parallel batch pattern BP algorithm for MLP does not depend on the number of training epochs. Parallelization efficiencies of this algorithm are respectively 95%, 84% and 63% on 2, 4 and 8 processors of the general-purpose NEC TX-7 parallel computer for a 5-10-1 MLP with 794 training patterns and an increasing number of training epochs from 10^4 to 10^6.

As shown in [7], parameters such as the number of training patterns and the number of adjustable connections of NN (number of weights and thresholds) define the computational complexity of the training algorithm and, therefore, exert influence on its parallelization efficiency. Therefore, research efficiency scenarios should be based on these parameters. In this case the purpose of our experimental research is to answer the question: what is the minimal/enough number of MLP connections and what is the minimal/enough number of training patterns in the input data set for the parallelization of batch pattern BP training algorithm to be efficient on a general-purpose high performance computer?

The following architectures of MLP are researched in order to provide the analysis of efficiency: 3-3-1 (3 input neurons × 3 hidden neurons = *9* weights between the input and the hidden layer + *3* weights between the hidden and the output layer + *3* thresholds of the hidden neurons and *1* threshold of the output neuron = 16 connections), 5-5-1 (36 connections), 5-10-1 (71 connections), 10-10-1 (121 connections), 10-15-1 (181 connections), 15-15-1 (256 connections), 20-20-1 (441 connections). The number of training patterns is changed as 25, 50, 75, 100, 200, 400, 600 and 800. It is necessary to note that such MLP architectures and number of training patterns are typical for most of neural-computation applications. During the research the neurons of the hidden and output layers have logistic activation functions. The number of training epochs is fixed to 10^5. The learning rate is constant and equal $\alpha(t) = 0.01$.

The parallelization efficiency of the batch pattern BP training algorithm is depicted in Figs. 3-5 on 2, 4 and 8 processors of NEC TX-7 respectively. The expressions $S=Ts/Tp$ and $E=S/p\times 100\%$ are used to calculate a speedup and efficiency of parallelization, where Ts is the time of sequential executing the routine, Tp is the time of parallel executing of the same routine on p processors of parallel computer. It is necessary to use the obtained results as the following: (i) first to choose the number of parallel processors used (Fig. 3 or Fig. 4 or Fig. 5), (ii) then to choose the curve, which characterizes the necessary number of perceptron's connections and (iii) then to get the value of parallelization efficiency from ordinate axes which corresponds to the necessary number of training patterns on abscissa axes. For example, the parallelization efficiency of the MLP 5-5-1 (36 connections) is 65% with 500 training patterns on 4 processors of NEC TX-7 (see Fig. 4). Therefore the presented curves are the approximation characteristics of a parallelization efficiency of the certain MLP architecture on the certain number of processors of a general-purpose parallel computer.

As it is seen from the Figs. 3-5, the parallelization efficiency is increasing when the number of connections and the number of the training patterns is increased. However, the parallelization efficiency is decreasing for the same scenario at increasing the number of parallel processors from 2 to 8. The analysis of the Figs. 3-5 allows defining the minimum number of the training patterns which is necessary to use for efficient parallelization of the batch pattern training algorithm at the certain number of MLP connections (Table 1).

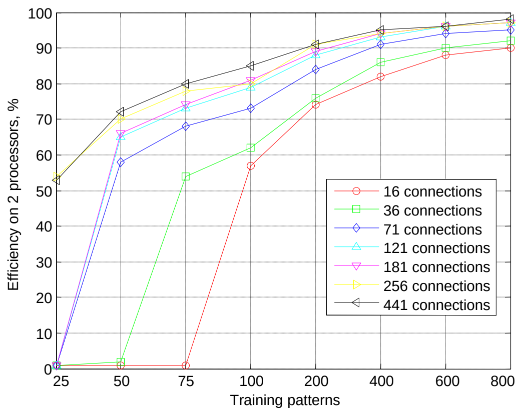


Fig. 3. Parallelization efficiency on 2 processors of NEC TX-7.

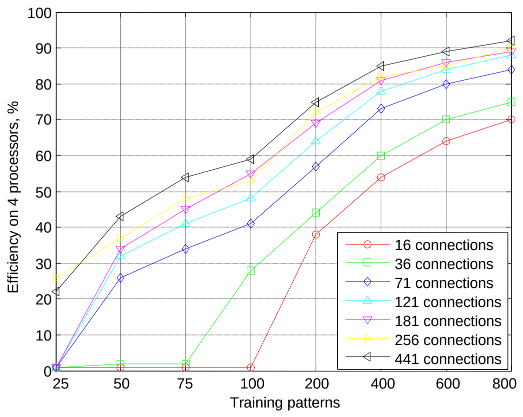


Fig. 4. Parallelization efficiency on 4 processors of NEC TX-7.

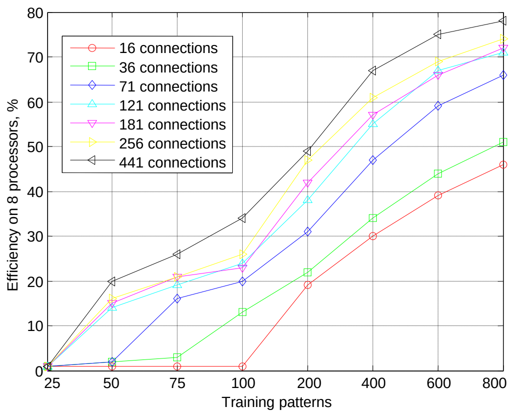


Fig. 5. Parallelization efficiency on 8 processors of NEC TX-7.

For example, the Table 1 shows that the number of training patterns should be 100 and more (100+) for efficient parallelization of MLP with the number of connections more than 16 and less and equal than 36. As it is seen from the Table 1, it is necessary to use more training patterns in a case of small MLP architectures. The minimum number of the training patterns is increasing in a case of parallelization on the bigger number of parallel processors.

Table 1. Minimum number of training patterns for efficient parallelization on NEC TX-7.

2 processors		4 processors		8 processors	
Connections number, C	Training patterns	Connections number, C	Training patterns	Connections number, C	Training patterns
$16 < C \leq 36$	100+	$16 < C \leq 36$	200+	$16 < C \leq 36$	200+
$36 < C \leq 71$	75+	$36 < C \leq 71$	100+	$36 < C \leq 71$	100+
$71 < C \leq 256$	50+	$71 < C \leq 256$	50+	$71 < C \leq 121$	75+
$C > 256$	25+	$C > 256$	25+	$C > 121$	50+

5 Conclusions

The parallel batch pattern back propagation training algorithm of multilayer perceptron is developed in this paper. The analysis of parallelization efficiency is done for 7 scenarios of increasing the perceptron's connections (number of weights and thresholds), in particular 16, 36, 71, 121, 181, 256 and 441 and increasing the number of training patterns, in particular 25, 50, 75, 100, 200, 400, 600, 800. The presented results can be used for estimation a parallelization efficiency of concrete perceptron model with concrete number of training patterns on the certain number of parallel processors of a general-purpose parallel computer. The experimental research proves that the parallelization efficiency of batch pattern back propagation training algorithm is (i) increasing at increasing the number of connections and increasing the number of the training patterns and (ii) decreasing for the same scenario at increasing the number of parallel processors from 2 to 8. The results of analysis of minimum number of training patterns for efficient parallelization of this algorithm show that (i) it is necessary to use more training patterns in case of small architectures of multilayer perceptron and (ii) the minimum number of the training patterns should be increased in a case of parallelization on the bigger number of parallel processors.

The provided level of parallelization efficiency is enough for using this parallel algorithm in Grid environment on the general-purpose parallel and high performance computers. For the future research it is expedient to estimate the factors of decreasing the parallelization efficiency of batch pattern back propagation training algorithm at small number of training patterns and small number of adjustable connections of multilayer perceptron.

Acknowledgements

This research is financially supported by a Marie Curie International Incoming Fellowship grant of the corresponding author Dr. V. Turchenko, Ref. Num. 221524 “PaGaLiNNeT - Parallel Grid-aware Library for Neural Networks Training", within the 7th European Community Framework Programme. This support is gratefully acknowledged.
We wish to thank anonymous referees for thoughtful and helpful comments which improved the readability of the paper.

References

1. Haykin, S.: Neural Networks. Prentice Hall, New Jersey (1999).
2. Mahapatra, S., Mahapatra, R., Chatterji, B.: A Parallel Formulation of BP Learning on Distributed Memory Multiprocessors. Parallel Computing. 22 (12) (1997) 1661–1675.
3. Hanzálek, Z.: A Parallel Algorithm for Gradient Training of Feed-forward Neural Networks. Parallel Computing. 24 (5-6) (1998) 823–839.
4. Murre, J.M.J.: Transputers and Neural Networks: An Analysis of Implementation Constraints and Perform. IEEE Transactions on Neural Networks. 4 (2) (1993) 284–292.
5. Dongarra, J., Shimasaki, M., Tourancheau, B.: Clusters and Computational Grids for Scientific Computing. Parallel Computing. 27 (11) (2001) 1401–1402.
6. Topping, B.H.V., Khan, A.I., Bahreininejad, A.: Parallel Training of Neural Networks for Finite Element Mesh Decomposition. Computers and Structures. 63 (4) (1997) 693–707.
7. Rogers, R.O., Skillicorn, D.B.: Using the BSP Cost Model to Optimise Parallel Neural Network Training. Future Generation Computer Systems. 14 (5) (1998) 409–424.
8. Ribeiro, B., Albrecht, R.F., Dobnikar, A., et al: Parallel Implementations of Feed-forward Neural Network using MPI and C# on .NET Platform. In: Proceedings of the International Conference on Adaptive and Natural Computing Algorithms. Coimbra (2005) 534–537.
9. Turchenko, V.: Computational Grid vs. Parallel Computer for Coarse-Grain Parallelization of Neural Networks Training. In: Meersman, R., Tari, Z., Herrero, P. (eds.): OTM 2005. Lecture Notes in Computing Science, vol. 3762. Springer-Verlag, Berlin Heidelberg New York (2005) 357–366.
10. Turchenko, V.: Fine-Grain Approach to Development of Parallel Training Algorithm of Multi-Layer Perceptron. Artificial Intelligence, the Journal of National Academy of Sciences of Ukraine. 1 (2006) 94–102.
11. Golovko, V., Galushkin, A.: Neural Networks: Training, Models and Applications. Radiotechnika, Moscow (2001) (in Russian).
12. Turchenko, V.: Scalability of Parallel Batch Pattern Neural Network Training Algorithm. Artificial Intelligence, the Journal of National Academy of Sciences of Ukraine. 2 (2009).

ANN Applications in Prediction & Forecasting

Vegetative State: Early Prediction of Clinical Outcome by Artificial Neural Network

L. Pignolo[1], F. Riganello[1], A. Candelieri[2] and V. Lagani[2]

[1] S. Anna Institute, RAN – Research on Advanced Neuro-rehabilitation, Crotone, Italy
{l.pignolo,f.riganello}@istitutosantanna.it
[2] Laboratory for Decision Engineering and Health Care Delivery
Department of Electronic Informatics and Systemistics
University of Calabria, Cosenza, Italy
{candelieriantonio,vlagani}@yahoo.it

Abstract. Residual brain function has been documented in vegetative state patients, yet early prognosis remains difficult. Purpose of this study was to identify by artificial Neural Network procedures the significant neurological signs correlated to, and predictive of outcome. The best networks test set accuracy was 70%, 72% and 70% for the entire patients' group and the posttraumatic and non-posttraumatic subgroups, respectively. The method accuracy does not reflect a perfect classification, but is significantly far from the random or educated guess and is in accordance with the results of previous clinical studies.

1 Introduction

The Vegetative State (VS) is a clinical condition characterized by the absence of awareness (of self and environment), voluntary or purposeful behavioral responses to external stimuli, and communication in the severely brain damaged. Subjects in VS are otherwise awake, often with wakefulness-sleep cycles [1, 2, 3, 4, 5, 6]. Recovery (with varying residual disabilities) occurs only in a portion of patients; resources, staff, logistics and costs requirements for the care of these subjects are imposing irrespective of outcome. Purpose of this study was to identify by artificial intelligence procedures a significant model supporting decision in the early prognosis of VS subjects [7, 8]. It should be noted in this regard that evidence-based neurology indicates significant neurological signs correlated to, and predictive of outcome. Prognosis can be modeled as a regression, classification or survival analysis problem by traditional statistics or machine learning techniques [9]. This study is purposed to demonstrate that reliable classification models predictive of the vegetative state outcome prognosis can be obtained by Artificial Neural Networks (ANN) techniques. Section 2 of this paper outlines dataset and pre- processing; Section 3 describes the experimentation protocol for the training of classification models; Sections 4 and 5 summarize and comment the results.

2 Data Collection and Pre-processing

2.1 Data Collection

Three hundred and thirty three subjects in VS consecutively admitted to the dedicated semi-intensive care unit of the S. Anna – RAN Institute (Crotone, Italy) over a 9-year period (April 1998–March 2006) were considered retrospectively. The VS was clinically defined in all subjects compliant to the criteria suggested by the Multi-Society Task Force and the guidelines of the London Consensus Conference (Multi-Society Task Force, 1994).

For each patient, were entered in the dataset: age, sex, etiology of brain injury (posttraumatic or non-posttraumatic), rating at the Glasgow Coma Scale (GCS) [10] at admission, and twenty-two neurological signs of established relevance in coma and VS [11] (Tables 1) assessed by the attending physician at two-week intervals following procedures and criteria predefined as intrinsic to the UNI ENI ISO 9001:2000 quality standards. Each sign was present or absent (binary attribute). The subjects' condition at discharge was measured by the Glascow Outcome Scale (GOS): GOS_1=death; GOS_2=vegetative state exceeding 1 year in duration; GOS_3=recovery, with severe disabilities; GOS_4=recovery, with mild disabilities; and GOS_5=full recovery or recovery with minimal disabilities not interfering with the everyday life [12]. The GOS is widely used in the evaluation of the VS outcome, but the subject's assignment to any GOS class is subjected to misclassification [13] which could affect the training of classification models. Therefore, the first two classes and the latest two classes of GOS were combined into the GOS_{1-2} and GOS_{4-5} classes respectively, with a resulting sharper separation among classes. The prediction of outcome was estimated at admission and after 50, 100 and 180 days after admission.

Table 1. Clinical signs assessed at two-week intervals and entered into the artificial neural network processing as potential prognostic factors.

Decerebration
Decortication
Conjugated gaze deviation
Skew eye deviation
Blink reflex
Cilio-spinal Reflex
Tactile-oral Reflex
Optic-oral Reflex
Bulldog Reflex
Grasping reflex
Corneal Reflex
Corneal-mandibular reflex
Threat reflex
Myotactic-cervical reflex
Chewing reflex
Sucking Reflex
Oculo-cephalic reflex (with disappearance of the doll's head phenomenon)
Absence of spontaneous motility
Eye tracking
Snout Rabbit sign
Half-moon pucker sign
Klippel sign

2.2 Pre-processing

Etiology of brain injury and the pathophysiology underlying VS are known to influence the outcome. The dataset inclusive of all patients and two data sub-sets of the posttraumatic (n=213) and non-posttraumatic patients were considered. Continuous numerical attributes (such as age and GCS level) were normalized in the interval [0;1] for each dataset; remaining attributes were binary and did not require pre-processing.

3 Experimentation

3.1 Parameter Configuration

The classification models were structured as classical feed-forward ANN , with one or two hidden layers and sigmoid function activation [14, 15]. The number of neurons was varied among 1, 2, 4, 6, 10, 15, 20, 25, 30 and 40 for both the first and second hidden layer. The Stuttgart Neural Network Simulator (SNSS) was used for all the experimentations [16]

The training of the ANN was performed by using the standard Back Propagation algorithm and the "Enhanced Back Propagation" algorithm. The latter introduces the previous arc weight change as a parameter for computing the new arc weight change. SNNS implements both algorithms with the Std_Backpropagation and BackpropMomentum functions. In particular, the Std_Backpropagation function requires the specification of the parameter η (learning rate) and d_{max} (maximal difference between expected and calculated output for each neuron). Besidea η, the BackpropMomentum function needs the momentum μ measuring the influence of the previous arc weight change on the current weight calculation. Table 2 shows the parameters configuration used for the training algorithms.

Table 2. Training algorithms parameters configurations.

Std_Backpropagation		*BackpropMomentum*	
η	d_{max}	η	M
0.1	0.1	0.1	0.2
0.3	0.2	0.3	0.8
0.5		0.5	
0.7		0.7	
0.9		0.9	

3.2 Experimentation Protocol

We used a Training–Validation–Test (TVT) procedure to select the best parameter configuration regulating both the network structure and the training algorithm operation. In particular, for each dataset the following steps were applied:

1. creation of training, validation and test set (see Table 3);
2. for each combination of network and training algorithm parameters:
    a. execution of 200 training cycles;
    b. evaluation of network accuracy on the validation set;
    c. if the total number of training cycles is 20000, then stop; otherwise, return to step a;
3. selection of the network with the best accuracy on the validation set;
4. evaluation of accuracy on the test set.

At the end of the TVT procedure, we obtained three trained ANN (one for each dataset) with their respective accuracy on the test set.

Table 3. Subdivision of instance among training, validation and test sets.

Dataset	*Training*	*Validation*	*Test*
NPT Dataset	80	20	20
PT Dataset	133	30	50
Entire Dataset	200	53	80

4 Results

The best networks test set accuracy was 70%, 72% and 70% for the entire patients' group and the posttraumatic and non-posttraumatic subgroups, respectively. The best parameter configurations are reported in Table 4.

Table 4. Configurations parameters of the best networks. BP: standard back propagation algorithm; EBP: enhanced back propagation; na: not applicable.

Dataset	*Entire dataset*	*PT dataset*	*NPT dataset*
1st hidden layer	30	1	6
2nd hidden layer	30	N.A.	N.A.
Training algorithm	BP	EBP	BP
η	0.7	0.1	0.7
d_{max}	N.A.	N.A.	0.2
μ	0.2	0.8	N.A.

A better understanding of the classificatory performance can be obtained through the analysis of the confusion matrices (see Tables 5 and 6) indicating misclassified elements. We decided to assign instances with unclear evaluation to the "misclassified" class (e.g. the same instance was assigned to two classes at the same time with similar probability).

Table 5. Entire dataset confusion matrix.

	predicted class			
real class	1_2	3	4_5	*Misclassified*
1_2	16	1	3	2
3	5	2	11	0
4_5	1	0	38	0

Table 6. Posttraumatic dataset confusion matrix.

	predicted class			
real class	1_2	3	4_5	*Misclassified*
1_2	6	0	0	3
3	0	0	7	2
4_5	0	0	30	2

Table 7. Non-posttraumatic dataset confusion matrix.

	predicted class			
real class	1_2	3	4_5	*Misclassified*
1_2	8	2	0	0
3	0	5	0	1
4_5	1	1	1	1

5 Comment

The method accuracy does not reflect a perfect classification, but is significantly far from the random or educated guess and is in accordance with the results of previous clinical studies [11]. It should be noted that class GOS_3 has a larger error estimate both in the entire dataset and in the posttraumatic sub-set. The higher misclassification depends on this class taking into account all patients with a severe motor outcome (e.g. paresis of one or more limbs), impaired consciousness (e.g. global amnesia) or both. GOS_3 can therefore be heterogeneous and ANN are unable to identify a major labeling characteristic. Interestingly, test set patients with GOS_3 in the non-posttraumatic dataset are well classified, while GOS_{4-5} subjects of the same dataset are poorly classified. The limited size of the non-posttraumatic sample does not allow further investigation of such phenomenon.

References

1. Jennett B, Plum F. Persistent vegetative state after brain damage: a syndrome in search of a name. Lancet 1972;1:734-6.
2. Dolce G, Sazbon L. The posttraumatic vegetative state. Stuttgart, Thiene, 2002.
3. Laureys S. The neural correlate of (un)awareness: lessons from the vegetative state. Trends Cogn Sci 2005;9:556-9.
4. Jennett B. The vegetative state. Cambridge, UK, University Press, 2002.
5. Multi-Society Task Force on PVS. Statement on medical aspects of the persistent vegetative state. N Eng J Med 1994; 330:. 1499-1508.
6. Zeman A. Consciousness. Brain 2001;124:1263-89 (review).
7. Braakman R, Jennett WB, Minderhoud JM. Prognosis of the post traumatic vegetative state. Acta Neurochirurgica 1988;95): 49-52.
8. Schmutzard E, Kampfl A, Franz G, Pfausler B, Haring HP, Ulmer H, Felber S, Golaszewski S, Aichner F. Prediction of recovery from post traumatic vegetative state with cerebral magnetic-resonance imaging. Lancet 1998;351:1763-67.
9. Rovlias A, Kotsou S.Classification and regression tree for prediction of outcome after severe head injury using simple clinical and laboratory variables. J Neurotrauma. 2004;21:886-93.
10. Teasdale G, Jennet B. Assessment of coma and impaired consciousness: a practical scale. Lancet 1974; 2:81-84.
11. Dolce G., Quintieri M., Serra S., Lagani V., Pignolo L. Clinical signs and early prognosis: a decisional tree, data mining study. Brain Injury. 2008, 22:7, 617 — 623.
12. Jennet B., Bond M. Assessment outcome after severe brain damage: a practical scale. Lancet 1976; 1: 480-484.
13. Pignolo, L., Quintieri M., Sannita, W. G. 'The Glasgow outcome scale in vegetative state: A possible source of bias, Brain Injury, 2009, 23:1,1 — 2.
14. Holte, R.C. (1993). Very simple classification rules perform well on most commonly used datasets. Machine Learning, 11, 63–90.
15. Van Bemmel, J.H.,&Munsen, M.A. (1997). Handbook of medical informatics. Berlin: Springer-Verlag.
16. Zell, A., Mache, N., Hubner, R., Schmalzl, M., Sommer, T., Korb, T. SNNS Stuttgart Neural Network Simulator Users Manual, Version 2.0, report (1992),No. 3/92, IPVR, Universität Stuttgart.

Investigation of the Use of an Artificial Neural Network Method for the Prediction of Crystal Structures of Zeolites from XRD Data

Melkon Tatlier[1] and H. Kerem Cigizoglu[2]

[1] Department of Chemical Engineering, Istanbul Technical University
Maslak, 34469 Istanbul, Turkey
tatlierm@itu.edu.tr
[2] Department of Civil Engineering, Division of Hydraulics, Istanbul Technical University
Maslak, 34469 Istanbul, Turkey
cigiz@itu.edu.tr

Abstract. The possibility of using artificial neural network methods for the prediction of zeolite crystal structures, such as pore size and unit cell dimensions, from X-ray diffraction patterns was investigated. The Generalized Regression Neural Network method and X-ray diffraction data obtained from literature were utilized in these investigations. The predictions made by using this neural network method were, in general, more reliable than those performed by regression. The best predictions were achieved for the estimation of the pore size, while the neural network method improved significantly the very poor results obtained by regression for the unit cell dimensions.

1 Introduction

Zeolites are hydrated microporous crystalline aluminosilicates that may be used in diverse applications related to ion exchange, catalysis, adsorption and separation [1-3]. Zeolites have developed into a large industry due to their unique and versatile properties. They may be utilized in the separation of linear and branched hydrocarbons, for catalytic cracking and hydrocracking or as detergent builders, to name a few, while a significant number of potential applications are waiting to emerge. Zeolites may occur naturally or be synthesized in laboratory conditions. The most significant parameters determining the type of the zeolite formed from a certain initial reaction mixture composition are the synthesis time and temperature as well as the molar ratios of the reactants. Suitable reagents that form a clear solution or a gel mixture should be used to obtain different types of zeolites. After carrying out the synthesis procedure with these reagents, the solid material formed in the solution should be separated by filtration, which is then characterized by using various techniques. X-ray diffraction (XRD) is one of the basic and essential techniques to characterize the solid material thus obtained. XRD may be used to determine the crystallographic structure, grain size and orientation of the crystals. It is commonly utilized to identify unknown substances by comparing diffraction data against a database. The relative abundance of crystalline materials in solid mixtures may also be determined by this technique.

Additionally, when coupled with lattice refinement techniques, it can provide structural information on unknown materials. The achievement of this last deed is not a simple task to perform and requires significant amount of knowledge on the numerous peaks pertaining to the X-ray diffraction patterns of different materials.

Artificial neural networks (ANNs) have the ability to learn from input data and are very useful for the prediction of complex high-dimensional data. ANN methods have a broad range of applications, including research in chemical engineering. Artificial neural networks have been successfully used for dynamic modeling and control of chemical processes and fault diagnosis [4], in the catalytic modeling and design of solid catalysts [5] and for modeling the kinetics of a chemical reaction [6]. The applicability of ANN methods in emulsion liquid membranes [7] and in the prediction/estimation of the vapor-liquid equilibrium data [8] has been investigated. It has also been shown that ANN methods might learn efficiently from available zeolite synthesis data in the literature to predict the complex relationship between the chemical compositions of initial reaction mixtures and the zeolites formed from them [9].

A detailed theoretical investigation of the rather complex and high dimensional relationship between the XRD peaks and the crystallographic properties of various zeolites (as well as other crystalline materials) may be very useful to provide a more common and practical use of the XRD technique in the prediction of the crystal structures of unknown zeolites and other materials. In this study, the Generalized Regression Neural Network (GRNN) method was utilized to perform this investigation. The results obtained were compared to XRD data reported in the literature, as well as to the estimations made by using multilinear regression.

2 Theory

2.1 X-Ray Diffraction Technique

Crystals are regular arrays of atoms, and X-rays can be considered as waves of electromagnetic radiation. Atoms scatter X-ray waves, primarily through their electrons. An X-ray striking an electron produces secondary spherical waves emanating from the electron, which is known as elastic scattering. Although these waves cancel one another out in most directions through destructive interference, they add constructively in a few specific directions, determined by Bragg's law,

$$2d\sin\theta = n\lambda \tag{1}$$

where d is the spacing between diffracting planes in Å, θ is the incident angle in degrees, n is any integer, and λ is the wavelength of the beam in Å. These specific directions appear as spots on the diffraction pattern. It should be mentioned that X-rays have wavelengths on the order of a few angstroms, the same as typical interatomic distances in crystalline solids. This means that X-rays can be diffracted from minerals which, by definition, are crystalline and have regularly repeating atomic structures. In the XRD technique, the X-ray intensity is recorded and reported as a function of the 2θ angle.

2.2 Artificial Neural Networks

Artificial neural networks are black box models that can perform an estimation using limited input and output data patterns. In this study, the Generalized Regression Neural Network (GRNN) method was used to relate the XRD data to the properties of the crystal structures of zeolites.

The basics of the GRNN can be found in the literature [10,11]. The GRNN method does not require an iterative training procedure but instead estimates any arbitrary function between input and output vectors, drawing the function estimate directly from the training data. This method is consistent, that is, as the training set size becomes large, the estimation error approaches zero, with only mild restrictions on the function. The GRNN is used for estimation of continuous variables, as in standard regression techniques. It is based on a standard statistical technique called kernel regression. By definition, the regression of a dependent variable y on an independent x estimates the most probable value for y, given x and a training set. The regression method will produce the estimated value of y, which minimizes the mean-squared error. The GRNN consists of four layers: input layer, pattern layer, summation layer, and output layer. The first layer is fully connected to the second, pattern layer, where each unit represents a training pattern and its output is a measure of the distance of the input from the stored patterns. Each pattern layer unit is connected to the two neurons in the summation layer: S-summation neuron and D-summation neuron. The S-summation neuron computes the sum of the weighted outputs of the pattern layer while the D-summation neuron calculates the unweighted outputs of the pattern neurons. The connection weight between the i^{th} neuron in the pattern layer and the S-summation neuron is y_i, the target output value corresponding to the i^{th} input pattern. For D-summation neuron, the connection weight is unity. The output layer merely divides the output of each S-summation neuron by that of each D-summation neuron. In this method, the spread σ is a smoothing parameter, the optimal value of which is often determined experimentally [12]. When the spread parameter σ is made large, the estimated density is forced to be smooth and in the limit becomes a multivariate Gaussian with covariance $\sigma^2 I$. On the other hand, a smaller value of σ allows the estimated density to assume non-Gaussian shapes, but with the hazard that wild points may have too great an effect on the estimate. In this study, different spreads were tried to find the best one that gave the minimum difference between predicted and experimental values for the utilization of the cross-validation data.

2.3 Method

Zeolites are hydrated microporous crystalline materials. The zeolite framework consists of an assemblage of SiO_4 and AlO_4 tetrahedra, joined together in various regular arrangements through shared oxygen atoms, to form an open crystal lattice. The micropore structure is determined by the crystal lattice, which contains pores of molecular dimensions into which guest molecules can penetrate. The cations (e.g., Na) are placed in special positions near the Al atoms. The pore size varies for different zeolites, depending on the arrangement of the atoms forming the zeolite crystal structure. The crystal structure of a material or the arrangement of atoms in a crystal structure can be described in terms of its unit cell. The unit cell is a tiny box with one or more spatial arrangements of atoms. The unit cells stacked in three-dimensional space describe the bulk arrangement of atoms of the crystal. The crystal structure has a three dimensional shape. The unit cell may be represented by its lattice parameters, including the length of the cell edges and the angles between them.

Data obtained from the literature [13], describing the XRD patterns of different zeolites were used in the estimations carried out by using the ANN method. Preliminary estimations indicated that the GRNN method was more successful in the prediction of the zeolite crystal structure from XRD data when compared to the Radial Basis Function-Based Neural Networks (RBF) and Feed Forward Back Propagation (FFBP) methods, which were also examined. Thus, the GRNN method was used for the estimations performed in detail. The components of the input vector were the 2θ angles of eight XRD peaks with the highest intensity pertaining to different zeolites, while the components of the output vector were the pore sizes (r) and lengths of the unit cell edges (a,b,c) of these zeolites. The pore sizes and unit cell dimensions of zeolites generally vary between a few angstroms and a few nanometers. Since the pores of some zeolites are not uniform and some others may have pore channels of different lengths, the largest dimensions of the pores were taken into consideration in this study.

The application of the ANN to data consisted of two steps. The first step was the training of the neural networks, which comprised the presentation of training data (data set 1) describing the input and output to the network and obtaining the interconnection weights. The components of the input vector were eight different 2θ angles of the XRD peaks, while the components of the output vector were the pore sizes and the three different lengths of the unit cell edges of corresponding zeolites. The input and output data were normalized between 0 and 1 prior to the training. Once the training stage was completed, the ANNs were applied to the cross-validation data (data set 2). Determining an appropriate architecture of a neural network for a particular problem is an important issue, since the network topology directly affects its computational complexity and its generalization capability. The number of hidden layers and the number of nodes in the hidden layers were determined after trying various network structures. The network structure providing the best result was determined according to the success of the predictions performed by using the cross-validation data set. The ANN method was used to predict only one component of the output vector at a time.

The number of data used for training was 55 while that used for cross-validation was 7. The zeolites consisting of silicon, aluminum, oxygen, water and different cations were taken into consideration in the investigations carried out in this study.

The results obtained by using the GRNN method were compared to the actual values [13] as well as to those values estimated by using multilinear regression. In regression, the relationships between the 2θ angles of eight XRD peaks and the pore sizes and lengths of the unit cell edges of zeolites were determined by using data set 1. The information obtained was used in the estimation of the pore sizes and lengths of the unit cell edges of the zeolites investigated in data set 2. Since, to our knowledge, a similar theoretical attempt, for determining such a relationship has not been performed before, the comparison of the results obtained from ANN methods to those determined by multilinear regression may be a reasonable first approach. The regression model tested in this study was of simple linear form, as given below. R represents either the pore size or the lengths of the unit cell edges of zeolites

$$R = a_0 + a_1(2\theta_1) + a_2(2\theta_2) + a_3(2\theta_3) + a_4(2\theta_4) + a_5(2\theta_5) + a_6(2\theta_6) + a_7(2\theta_7) + a_8(2\theta_8)\ a_8(2\theta_8) \quad (2)$$

The relative error (d) was used to monitor the success of the ANN method and regression used in the prediction of zeolite crystal properties from the 2θ angles of the X-ray diffraction patterns. d was determined by taking into consideration the deviation (%) of the pore sizes or lengths of the unit cell edges of zeolites, calculated by using the ANN method (c_{calc}), from the corresponding actual values (c_{act}).

$$d = |c_{act} - c_{calc}| / c_{act} \times 100 \quad (3)$$

d_m was defined as the arithmetic mean of the relative errors obtained for the different data used in prediction.

3 Results and Discussion

3.1 Criteria assuring Best Performance for the GRNN Method

The predictions of the pore sizes and lengths of the unit cell edges of zeolites investigated in data set 2 were performed by using GRNN method mentioned above, and data set 1 for training. As mentioned before, the network structure providing the best result was determined according to the success of the predictions performed by using the cross-validation data. It was also established that the conditions providing the best results in the testing stage could allow the ANN method to exhibit quite high performances in the training stage.

For the GRNN method, spread factors in the range 0.02-0.2 were the conditions determined to give the best results. The spread factor was determined to be equal to 0.15, 0.075, 0.2 and 0.02 for r (pore size of zeolites), a, b and c (lengths of unit cell edges of zeolites), respectively, by using the cross-validation data for the optimization. When the optimization was performed by using training data (data set 1), without taking into consideration data set 2, the d_m values, representing the deviation of the predicted values of data set 1 from experimental values, were less than 10% for all the cases investigated. When the spread factor was optimized by using data set 2, the predictive power of the GRNN method was not reduced significantly. The predictions made by using the GRNN method for the pore size and lengths of the unit cell edges of zeolites are detailed below.

3.2 Evaluation of the Predictions made by the GRNN Method

The results obtained by the GRNN method are depicted in Figures 1-4 for r (pore size), a, b, and c (lengths of the unit cell edges) of zeolites, respectively. The deviations of the results obtained by using the ANN method from the actual values may be observed more clearly from Table 1. The performance exhibited by multilinear regression may also be seen in the table. It may be observed from Figures 1-4 and Table 1 that the GRNN method provided fairly good fits to the actual results for most of the data, though there were some discrepancies. The average deviation from actual results was smaller for the pore size predictions while the largest deviation was observed for length, a, of the unit cell edge.

Table 1. Relative errors obtained for the predictions.

Method	d_m (%)			
	r	a	b	c
GRNN	12.0	30.9	17.5	22.2
regression	14.7	149.4	169.2	160.7

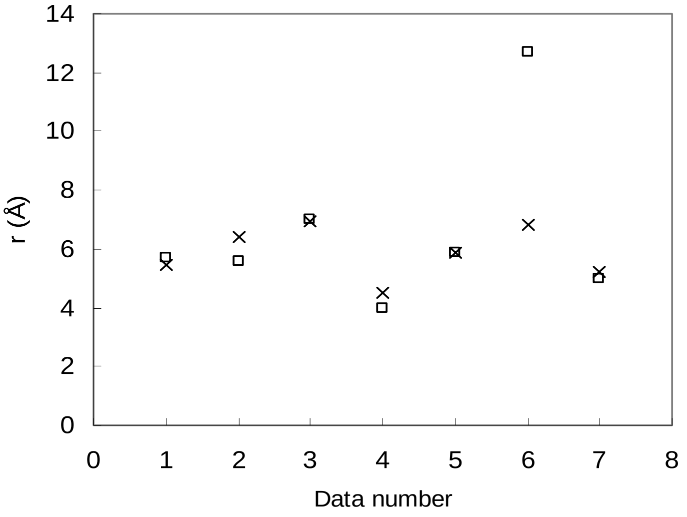


Fig. 1. Zeolite pore size predictions by (x) GRNN in comparison to (_) actual values.

The results obtained for regression given in Table 1 should also be taken into consideration before arriving at a conclusion about the success of the ANN method in the prediction of the zeolite crystal structure properties. When multilinear regression was utilized, the average deviation from the actual values was slightly higher than that of the GRNN method for the prediction of the pore size. However, the predictions of the lengths of the unit cell edges were very poor with multilinear regression. The average deviation was equal to about 150-170% for the estimation of all the three edges of the unit cell. When this information is taken into consideration, the average deviations between 17% and 31% provided by the GRNN method may be regarded to be quite promising. For further improvement, additional XRD data, for example, those pertaining to zeolite-like materials may also be included for the training of the neural networks. Furthermore, the height and width of the XRD peaks may also be taken into consideration as components of the input vector. Different ANN methods may also be tested for possible improvements in the prediction of crystal structures from XRD data.

The superiority of the ANNs over conventional methods for the prediction of complex and high dimensional relationships, such as the one investigated in this study, can be attributed to the capability of the ANNs to capture the nonlinear features and generalize the structure of the whole data set. ANN methods are flexible alternatives and standard ANN software can be used to construct intricate multipurpose nonlinear solutions. The method has no limitations in the form of fixed assumptions or formal constraints. The neural network has a distributed processing structure. Each individual processing unit or the weighted connection between two units is responsible for one small part of the input–output mapping system.

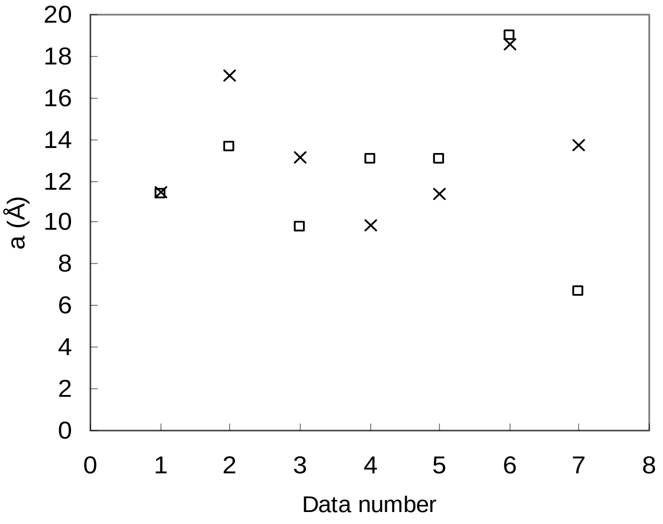


Fig. 2. Unit cell length, a, predictions by (x) GRNN in comparison to (_) actual values.

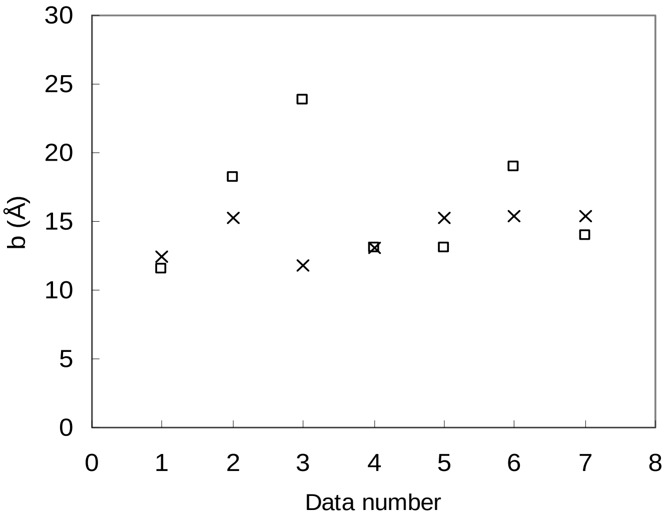


Fig. 3. Unit cell length, b, predictions by (x) GRNN in comparison to (_) actual values.

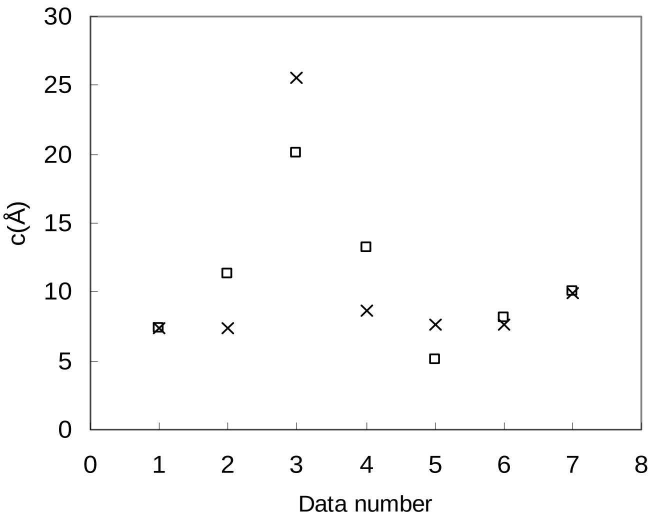


Fig. 4. Unit cell length, c, predictions by (x) GRNN in comparison to (_) actual values.

4 Conclusions

It was determined that neural networks might learn from XRD data to predict some properties of the crystal structures of zeolites. The predictions made were, in general, much more reliable than those performed by the multilinear regression. The best prediction was made for the pore sizes of zeolites, which also represented the case where the difference between the success of the predictions made by regression and neural networks was the smallest. The improvement provided by the use of the GRNN method, when compared to regression, was quite significant for the predictions of the lengths of the unit cells of zeolites.

The use of artificial neural network methods may allow a better understanding of the relationship between the X-ray diffraction patterns and the crystallographic properties of zeolites as well as other materials. This will ease and support the discovery of novel crystal materials since a short and practical characterization by using available XRD data will become possible. It should also be remembered that in case training and cross-validation data other than those adopted in this study, (e.g., those pertaining to different types of materials) could be used with GRNN as well as other ANN models to make the predictions, the relative success of prediction might still improve. Recurrent neural networks and optimization of neural network architecture by using genetic programming are methods that may be tested for providing further developments.

References

1. Weitkamp, J.: Zeolites and Catalysis. Solid State Ionics 131 (2000) 175-188.

2. Caro, J., Noack, M., Kolsch, P., Schafer, R.: Zeolite Membranes-State of Their Development and Perspective. Microporous Mesoporous Mater. 38 (2000) 3-24.
3. Ruthven, D.M.: Zeolites as Selective Adsorbents. Chem. Eng. Progr. 84 (1988) 42-50.
4. Hussain, M.A.: Review of the Applications of Neural Networks in Chemical Process Control- Simulation and Online Implementation. Artif. Int. Eng. 13 (1999) 55-68.
5. Huang, K., Chen, F., Lu, D.: Artificial Neural Network-Aided Design of a Multicomponent Catalyst for Methane Oxidative Coupling. Appl. Catal. A 219 (2001) 61-68.
6. Serra, J.M., Corma, A., Chica, A., Argente, E., Botti, V.: Can Artificial Neural Networks Help the Experimentation in Catalysis? Cat. Today. 81 (2003) 393-403.
7. Chakraborty, M., Bhattacharya, C., Dutta, S.: Studies on the Applicability of Artificial Neural Network (ANN) in Emulsion Liquid Membranes. J. Membrane Sci. 220 (2003) 155-164.
8. Sharma, R., Singhal, D., Ghosh, R., Dwivedi, A.: Potential Applications of Artificial Neural Networks to Thermodynamics: Vapor-Liquid Equilibrium Predictions. Comp. Chem. Eng. 23 (1999) 385-390.
9. Tatlier, M., Cigizoglu, H.K., Erdem-Şenatalar, A.: Artificial Neural Network Methods for the Estimation of Zeolite Molar Compositions That Form from Different Reaction Mixtures. Comp. Chem. Eng. 30 (2005) 137-146.
10. Tatlier, M., Cigizoglu, K.B, Cigizoglu, H.K., Erdem-Şenatalar, A.: Low-Silica Zeolite Coatings Prepared by Using Predictions from an Artificial Neural Network Method. J. Porous Mater. 15 (2008) 389-395.
11. Specht, D.F.: A General Regression Neural Network. IEEE Trans. Neural Networks 2 (1991) 568-576.
12. Tsoukalas, L. H., Uhrig, R. E.: Fuzzy and Neural Approaches in Engineering. Wiley, New York (1997).
13. Kim, B., Kim, S., Kim, K.: Modeling of Plasma Etching Using a Generalized Regression Neural Network. Vacuum 71 (2003) 497-503.
14. Treacy, M.M.J., Higgins, J.B.: Collection of Simulated XRD Powder Patterns for Zeolites. Elsevier, Amsterdam (2001).

Typhoon Damage Forecasting with Self-Organizing Maps, Multiple Regression and Decision Trees

Kazuhiro Kohara and Ryo Hasegawa

Department of Electrical, Electronics and Computer Engineering
Chiba Institute of Technology, 2-17-1, Tsudanuma, Narashino, Chiba, 275-0016, Japan
kohara.kazuhiro@it-chiba.ac.jp

Abstract. Damage caused by typhoons to both people and structures has decreased in Japan due to improvements of countermeasures against natural disasters, however, such damage still occurs. A typhoon warning that represents the risk posed by a typhoon with high accuracy should be issued appropriately. Thus, we propose a new typhoon warning system which forecasts the likely extent of damage associated with a typhoon towards humans and buildings. The relation between typhoon data and damage data is investigated and typhoon damage is forecast using typhoon data. Self-organizing maps (SOM), multiple regression analysis and decision trees were used for typhoon damage forecasting. We consider two types of forecasting: two-class (*yes* or *no*) and three-class (*small*, *medium* or *large* scale) damage forecasting. Experimental results on accuracy of two-class and three-class forecasting with SOM were 93.3% and 96.8%, respectively. The accuracy with SOM was much better than that with multiple regression and decision trees. We recommend a new typhoon damage forecasting method based on these results.

1 Introduction

Intelligent techniques such as back-propagation neural networks (BPNN) [1], self-organizing maps (SOM) [2], decision trees [3] and Bayesian networks [4] have been extensively investigated, and various attempts have been made to apply them to identification, prediction and control [e.g., 1-10]. Harada et al. applied BPNN to forecasting typhoon course [8], Takada et al. applied BPNN to forecasting typhoon damage of electric power systems [9] and Udagawa et al. applied Bayesian networks to rain prediction [10]. This paper applies intelligent techniques to forecasting typhoon damage to human and buildings.

Damage caused by typhoons to both people and structures has decreased in Japan due to improvements of countermeasures against natural disasters, however, such damage still occurs [11, 12]. A typhoon warning that represents typhoon menace with high accuracy should be issued appropriately. A typical typhoon warning currently issued may be “This typhoon is large and very strong”. We propose a new typhoon warning which forecasts the risk of damage scale to both human and buildings. We investigate relation between typhoon data and damage data and forecast typhoon damage using typhoon data. The typhoon data includes the month when the typhoon was born, latitude and longitude where the typhoon was born, lowest atmospheric pressure, maximum wind speed and total precipitation. Damage data includes human damage data such as number of fatalities and injured persons and building damage data such as number of completely destroyed houses and number of houses under water.

We use SOM, multiple regression analysis and decision trees for typhoon damage forecasting. SOM [2] are neural networks which consist of two layers: input layer and map layer. As an interesting feature of SOM, teaching vectors are not required and input vectors are automatically classified in accordance with similarity, updating the weight of the *winning neuron* and the neighbor neurons. After trained by SOM algorithm, the weight vectors of the neurons form the cluster of input vectors. A decision tree [3] is an inductive learning algorithm. In a decision tree algorithm, an explicit decision boundary is extracted from the training data, and an example *E* is classified into class *c* if *E* falls into the decision area corresponding to *c*. *Viscovery SOMine 4.0* was used as SOM software and *See5 release 1.19* was used as decision tree software with default parameter values.

2 Forecasting Damage Data using Typhoon Data

139 data records of typhoon data and damage data from June 1981 to September 1999 were collected from the typhoon database [13, 14]. The types of typhoon and damage data are shown in Table 1. There are nine types of typhoon data and nine types of damage data, divided into three types of human damage and six types of building damage. We used 111 data records (to September 1995) for learning and 28 data records (from July 1996) for testing.

Table 1. Types of typhoon data and damage data used in this study.

Typhoon data	Month when the typhoon was born, Latitude and longitude where the typhoon was born, Lowest atmospheric pressure, Maximum wind speed, Total, one-hour and twenty-four-hour precipitation, Life span
Damage data	Number of fatalities, Number of injured persons, Number of dead and injured persons
	Number of completely destroyed houses, Number of half destroyed houses, Number of partially destroyed houses, Total number of damaged houses, Number of houses under water, Total number of destroyed non-house structures

The average and maximum of every damage type are shown in Table 2. The minimum of every damage type was zero.

Table 2. Average and maximum of every type of damage data.

Data type	Average	Maximum
Number of fatalities	5.5	100
Number of injured persons	39.2	1499
Number of dead and injured persons	44.8	1561
Number of completely destroyed houses	21.9	541
Number of half destroyed houses	1839.8	169877
Number of partially destroyed houses	1051.7	85989
Total number of damaged houses	2913.4	170418
Number of houses under water	7829.6	174124
Total number of destroyed non-house structures	163.6	15840

In this study, we consider two types of typhoon damage forecasting: two-class (*yes* or *no*) and three-class (*small*, *medium* or *large* scale) damage forecasting.

3 Two-class (Yes or No) Damage Forecasting

In two-class damage forecasting, a predictor is trained by two values (0 and 1). In this case, 0 means that the damage is zero (*no*) and 1 means that the damage is not zero (*yes*). Experiments were made with nine types of continuous typhoon data as inputs and one damage data (two values) as an output. Here, we expect that typhoon data such as lowest atmospheric pressure, maximum wind speed and precipitation can be forecast with high accuracy by a weather forecasting system such as Japanese SYNFOS [15] and hence actual typhoon data was used as inputs.

Table 3. Average accuracy of two-class (*yes* or *no*) damage forecasting.

Method	Learning data	Test data
Self-organizing maps (SOM)	100%	93.3%
Multiple regression (MR)	70.9%	70.2%
Decision trees (DT)	77.7%	63.9%

Table 4. Accuracy of two-class (*yes* or *no*) damage forecasting for test data.

Damage type	SOM	MR	DT
No. fatalities	92.9%	57.1%	50.0%
No. injured persons	89.3%	75.0%	75.0%
No. dead and injured persons	96.4%	89.3%	85.7%
No. completely destroyed houses	92.9%	57.1%	57.1%
No. half destroyed houses	89.3%	71.4%	71.4%
No. partially destroyed houses	92.9%	67.9%	60.7%
Total no. of damaged houses	96.4%	75.0%	64.3%
No. houses under water	96.4%	85.7%	78.6%
Total no. destroyed non-house structures	92.9%	53.6%	32.1%
Average	93.3%	70.2%	63.9%

SOM: self-organizing maps, MR: multiple regression, DT: decision trees

The average accuracy of two-class (*yes* or *no*) damage forecasting for the three intelligent methods is shown in Table 3. Here, average accuracy means the average of the accuracy of nine damage data. The average accuracy of the learning and test data using SOM was 100% and 93.3%, respectively. This experiment confirmed that damage data are well related with typhoon data and that SOM learned the nonlinear relation very well. The accuracy for each damage test data is shown in Table 4. Each damage data was forecast very well by SOM. The accuracy with SOM was much better than that with multiple regression and decision trees.

4 Three-class (Small, Medium or Large Scale) Damage Forecasting

In three-class damage forecasting, two experiments were made with nine types of continuous typhoon data as inputs and one damage data as an output. In the first experiment, a predictor is trained by continuous damage data. As this is a regression problem, decision trees were not used. The average of each damage data was calculated as shown in Table 2. *Small scale* corresponds to under half of the average, *medium scale* corresponds to between half of the average and the average, and *large scale* corresponds to over the average, respectively. The prediction was considered accurate when both the predicted value and the actual value correspond to the same size. The average accuracy of three-class damage forecasting when trained by continuous damage data is shown in Table 5. The average accuracy of the learning and test data with SOM was 100% and 78.6%, respectively. The accuracy for each damage type is shown in Table 6. The accuracy with SOM was much better than that with multiple regression, however, each damage data was not always forecast very well by SOM. For example, accuracy for number of fatalities and number of partially destroyed houses was 67.9% and 92.9%, respectively.

Table 5. Average accuracy of three-class (*small*, *medium* or *large* scale) damage forecasting when trained by continuous damage data.

Method	Learning data	Test data
Self-organizing maps (SOM)	100%	78.6%
Multiple regression (MR)	52.8%	43.7%

Table 6. Accuracy of three-class (*small*, *medium* or *large* scale) damage forecasting for test data when trained by continuous damage data.

Damage type	SOM	MR
No. fatalities	67.9%	35.7%
No. injured persons	75.0%	46.4%
No. dead and injured persons	75.0%	42.9%
No. completely destroyed houses	60.7%	46.4%
No. half destroyed houses	89.3%	42.9%
No. partially destroyed houses	92.9%	42.9%
Total no. of damaged houses	89.3%	39.3%
No. houses under water	82.1%	39.3%
Total no. destroyed non-house structures	75.0%	57.1%
Average	78.6%	43.7%

In the second experiment, a predictor is trained by three values (0, 1 and 2). As this is a classification problem, decision trees were used. In the learning data, 0 means that the damage is *small scale*, 1 means the damage is *medium scale* and 2 means the damage is *large scale*. The prediction was considered accurate when the predicted size was equal to the actual size. The average accuracy of three-class damage forecasting when trained by three values is shown in Table 7. The average accuracy of the learning and test data with SOM was 100% and 96.8%, respectively. This also confirmed that damage data are also well related with typhoon data. The accuracy for each damage type is shown in Table 8. Each damage type was also forecast very well by SOM. For example, accuracy for number of fatalities and number of partially destroyed houses was 85.7% and 100%, respectively. The accuracy with SOM was also much better than that with multiple regression and decision trees.

Table 7. Average accuracy of three-class (*small, medium* or *large* scale) damage forecasting when trained by three values.

Method	Learning data	Test data
Self-organizing maps (SOM)	100%	96.8%
Multiple regression (MR)	77.5%	65.1%
Decision trees (DT)	90.1%	78.6%

Table 8. Accuracy of three-class (*small, medium* or *large* scale) damage forecasting for test data when trained by three values.

Damage type	SOM	MR	DT
No. fatalities	85.7%	42.9%	78.6%
No. injured persons	100%	53.6%	71.4%
No. dead and injured persons	100%	53.6%	71.4%
No. completely destroyed houses	92.9%	39.3%	53.6%
No. half destroyed houses	100%	85.7%	96.4%
No. partially destroyed houses	100%	85.7%	85.7%
Total no. of damaged houses	100%	85.7%	92.9%
No. houses under water	92.9%	50.0%	60.7%
Total no. destroyed non-house structures	100%	89.3%	96.4%
Average	96.8%	65.1%	78.6%

5 Conclusions

We investigated typhoon damage forecasting with intelligent techniques. Using nine types of typhoon data as inputs to SOM, experimental results on the average accuracy of two-class (*yes* or *no*) and three-class (*small, medium* or *large* scale) damage forecasting were 93.3% and 96.8%, respectively. The accuracy with SOM was much better than that with multiple regression and decision trees. As a result, a typhoon forecasting method is proposed as follows: 1) Evaluate two-class damage forecasting, 2) When two-class forecasting result is *yes*, evaluate three-class damage forecasting, 3) Issue a typhoon warning based on above three-class damage forecasting. For example, such a warning may be issued, “According to the Japanese typhoon database, we forecast that the coming typhoon has a risk of causing both large scale human and building damage. Please take care.” In further research, we will consider more detailed damage forecasting and use other predictors such as support vector machines.

References

1. Rumelhart, D., Hinton, G., Williams, R.: Learning internal representations by error propagation. In: Rumelhart, D., McClelland, J., the PDP Research Group (eds.): Parallel Distributed Processing, Vol. 1. MIT Press, Cambridge, MA (1986).
2. Kohonen, T.: Self-Organizing Maps. Springer (1995).
3. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993).
4. Jensen, F.: Bayesian Networks and Decision Graphs. Springer (2001).
5. Pham, D., Liu, X.: Neural Networks for Identification, Prediction and Control. Springer (1995).
6. Kohara, K.: Neural networks for economic forecasting problems. In: Leondes, C. T.: Expert Systems - The Technology of Knowledge Management and Decision Making for 21st Century -. Academic Press (2002).
7. Kohara, K.: Combining selective-presentation and selective-learning-rate approaches for neural network forecasting of stock markets, Proceedings of International Workshop on Artificial Neural Networks and Intelligent Information Processing. Madeira (2008) Pp 3-9.
8. Harada, H., Momma, E., Ishii, H., Ono, T.: Forecast of typhoon course using multi-layered neural network (III), Proceedings of National Convention of the Institute of Electrical Engineers of Japan, Toyama (2007) Vol. 3, 111.
9. Takata, H., Kawaji, S., Ha, T.: Study on a Prediction Method of Typhoon Damage of Electric Power Systems in each District on the Main Island in Kagoshima Prefecture, Technical Report 48, Faculty of Engineering, Kagoshima University (2006).
10. Udagawa, S., Nishio, S., Kimura, M.: Rain prediction by the Bayesian network, Proceedings of National Convention of the Information Processing Society of Japan (2005) Vol.3, 237-238.
11. Murayama, K.: Introduction to Typhoon Study. Yama-Kei Publishers (2006).
12. Nyoumura, Y.: Weather Damage Prediction and Countermeasure. Ohmsha (2002).
13. National Research Institute for Earth Science and Disaster Prevention (2008). http://www.bosai.go.jp/index.html
14. National Institute of Informatics (2008). http://agora.ex.nii.ac.jp/digital-typhoon
15. Japan Weather Association (2008). http://www.jwa.or.jp/synfos/

Intelligent Systems and Applications

Particles Gradient: A New Approach to Perform MLP Neural Network Training based on Particle Swarm Optimization

César Daltoé Berci and Celso Pascoli Bottura

Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas
Av. Albert Einstein 400, Campinas, Brazil
cesardaltoe@hotmail.com, cpbottura@fee.unicamp.br

Abstract. The use of heuristic algorithms in neural networks training is not a new subject. Several works have already accomplished good results, however not competitive with procedural methods for problems where the gradient of the error is well defined. The present document proposes an alternative for neural networks training using PSO(Particle Swarm Optimization) to evolve the training process itself and not to evolve directly the network parameters as usually. This way we get quite superior results and obtain a method clearly faster than others known methods for training neural networks using heuristic algorithms.

1 Introduction

ANNs (Artificial Neural Networks) is a computational paradigm inspired in the operation of the biological brain, and seeks to explore certain properties present in the human neural processing, that are very attractive from the computational view point[10]. In this paper will be treated a specific type of ANNs, called MLP, a widely applied ANNs model, for which is found a vast literature.

MLP ANNs training process are usually a non linear optimization process, frequently based on the gradient of the ANNs error surface, which is calculated through the *backpropagation* algorithm [18],[19]. Among the more efficient methods can be mentioned, the quasi-Newton method: *BFGS* [1] and the conjugated directions method: *Scaled Conjugate Gradient* [14],[15], all of them considered order 2 optimization methods.

The present document introduces a new approach for MLP ANNs training using PSO [11],[17],[12],[6] (or another heuristic algorithm like genetic algorithms, however, differing from others proposals founded in the pertinent literature [20],[16] [21],[7],[9]). The proposed method: PG uses a sub-optimum gradient, and unit steps are taken in the direction of this gradient. This proposal makes use of the full exploratory capacity of the PSO, united with the efficiency of gradient descent methods.

2 The Particle Gradient Method

There are some works that try to optimize the ANNs vector of parameters using meta-heuristic algorithms, in most cases genetic algorithm are implemented. This process is in general more onerous from the computational cost[1] viewpoint, and shows poor results (with respect to the process convergence rate) when compared to conventional optimization methods, based on gradient descent. This scenario created the main motivation to this work, to create a meta-heuristic training method, that rivals procedural training methods with respect to convergence rate, here understood as learning efficiency, even when implemented in a digital machine.

The proposed method, represents an alternative solution inspired in the work of Chalmers *The Evolution of Learning: An Experiment in Genetic Connectionism*[8], that applied evolutionary processes to evolve the learning process itself, and not its solution.

In the PG: Particles Gradient method, the PSO algorithm is not used to optimize the vector of parameters itself, but to optimize the learning process by finding a sub-optimum gradient vector at each learning iteration (season).

2.1 Codification

The PSO algorithm applied in the proposed PG algorithm, uses a population of n_p particles, with a real codification described as follows:

- Particle Position: Vector containing real values belonging to the space R^N, represent the error gradient for an ANN.
- Fitness: $e(x, \theta - p)$ where p is a particle position, x is the ANN input and θ is its vector of parameters.
- Speed Modulation: A method to control the particles speed to provide a faster convergence rate [3].

2.2 The PG Algorithm

This algorithm is based on the gradient descent algorithm where steps are taken in the direction of a gradient vector, however, here unit steps are taken in the direction of a sub-optimum gradient found by the PSO algorithm.

To each iteration of the main algorithm, an initial gradient vector is calculated using the *Backpropagation* method, or simply by setting it equals to origin, which is clearly a good estimate, and than inserted in the population[2]. Later a vector representing a sub-optimum gradient is evolved by the PSO algorithm by n_i generations, and then an unit step is taken in this new descent direction.

The use of the initial gradient seeks to accelerate the convergence of the PSO algorithm, giving to him a reference point. However, the execution of the method without the use of an initial gradient also obtains good results, even similar to ones found using an initial gradient, as can be seen in [4]. This procedure is useful when it is not possible to accomplish the retro-propagation phase of the ANN, impeding the gradient vector construction, and consequently the application of faster procedural optimization methods.

[1] Here the computational cost of a procedure is understood as the number of sum and multiplication operations necessary to accomplish this

[2] Inserting a simple point in a particle's population consists of creating a new particle in the point's position

The GP algorithm for MLP ANNs training is described in detail as follow.

Algorithm 1: Particles Gradient.

```
Determine: $n_p$, $n_i$, $n_{max}$;
Initialize: $\theta$;
for i=1 to $n_{max}$ do
    Calculate $g_0$ by the retro-propagation of the error, or set $g_0 = 0$;
    Find the gradient: $g = PSO(n_p, n_i, g_0)$ ;
    Unit step: $\theta_{i+1} = \theta_i - g$
end
```

where PSO represents a Particle Swarm Optimization algorithm.

The algorithm 1, is the result a exhaustive study of the training process, and the functional analysis of the relations between the quantities of interest, taking into account the dimensionality of the involved spaces and the characteristics a priori known about the problem.

These studies converged to a method where the search blind is applied in a space of same dimension of the vector of parameters. However, this choice brings a great advantage with respect to the cost function to be optimized, or the surface where the particles will be moved. This choice provides a condition very particular to the PSO method, which is widely explored in this work.

In the application of a blind search method, as in case, the efficiency of the optimization process may be considerably increased if the method is initialized near the neighborhood of a local optimum point, which represents a good solution to the problem. However, this points are not known a priori, nor their neighborhoods.

The same happens in the ANNs training, the error surface is not complete known, so nothing can be done to increase the training algorithm efficiency. Moreover, it is known that a sufficient small step in a descent direction is ways a minimizing step. Therefore, we conclude that a R^N vector, representing a descent direction, will be certainly found in a neighborhood of the origin.

This information is the main goal of the proposed method, given to blind search method what it needs, a good initial condition. This approach gives the PSO method a considerable efficiency increase, so the algorithm PG, presents a significantly higher convergence rate when compared with others meta-heuristic methods in the same context.

Another important feature of the PSO method, is the intrinsic parallelism of the algorithm. Its implementation in a sequential machine, as a digital computer, will generate a process computationally very onerous, however, in a completely parallel machine, still hypothetical, is obtained a very faster and efficient process.

3 Examples and Comparisons

With the intention of determining the relative efficiency of the proposed method compared with others founded in the literature, some ANN application examples are used, and the efficiency of the learning process is evaluated when the network is trained by different methods.

In this document will be considered, as comparison bases, two quasi-Newton methods, two conjugated directions methods and one simple gradient descent method, all described as follows:

- GRAD: Optimium Gradient [13]: Gradient descent method with super-linear convergence, the fastest among methods with linear convergence rate.
- DFP: Davidson Fletcher Powell [13]: quasi-Newton Method with quadratic convergence.
- BFGS:Broyden Fletcher Goldfarb Shanno [1]: quasi-Newton Method with quadratic convergence, and with smaller sensibility to the bad numerical conditioning than the DFP method.
- FR: Fletcher Reeves [1]: Conjugated directions method, with N-steps quadratic convergence.
- SCG: Scaled Conjugated Gradient[15]: Conjugated directions method that do not use unidimensional searches. It possesses N-steps quadratic convergence, and it is the fastest among these methods from the computational cost viewpoint.

The process of unidimensional search used in the algorithms GRAD, DFP, BFGS and FR is the golden section method, applied by 30 iterations on the initial interval.

3.1 Motor Currents

In theory, the current of a three-phase induction motor can be easily calculated on the basis of motor voltage and power, as shown in equation (1).

$$I = \frac{P}{\sqrt{3}V\eta} \tag{1}$$

where P and V represent the motor power and tension respectively. The variable η takes into account the power factor and efficiency of the motor, that are based on construction factors, the mechanical load and the rotation of the motor. Thus, it is clear difficult to specify the variable η and so, the motor current.

The problem in question uses a neural estimator for the current calculation, based on motor power, voltage and rotation, through a MLP network containing 3 neurons in its sensorial layer, and with 1 neuron in its output layer.

The set used in the training consist on 300 samples obtained from manufacturers catalogs, including motors that meet the following values ranges:

- Power: 0.1 a 330 KW.
- Rotation: 600,900,1200,1800 e 3600 rpm.
- Tension: 220, 380 e 440 V.
- Current: 0.3 a 580 A.

In a first analysis will be used in comparison to the proposed algorithm, a training algorithm that uses the PSO technique directly applied to minimize the error surface with respect to the vector of parameters.

This method uses exactly the same meta-heuristic of the proposed algorithm, and also the same implementation, differing only in the application approach.

The configuration of both methods are described as follows:

Neural Network:

- Topology: 3 neurons in sensorial layer, 9 neurons in hidden layer and 1 neuron in output layer.
- Linear output.
- Hyperbolic tangent activation functions.

Particle Swarm Optimization: PSO

- 136 particles.
- 400 iterations.
- Speed modulation [3],[5].

Particles Gradient: PG

- 136 particles.
- 5 epocs.
- 80 iteration per epoc.
- Speed modulation.

Due the stochastic characteristics of both methods tested, it is necessary a statistical analysis of the results. Has been chosen in this paper the completion of 20 repetitions of the training process and subsequent analysis of average results, which are summarized in figure 1.

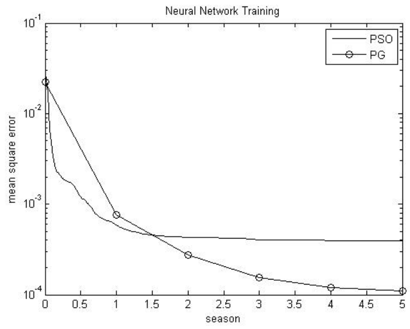


Fig. 1. Average values for mean squared error during the training process.

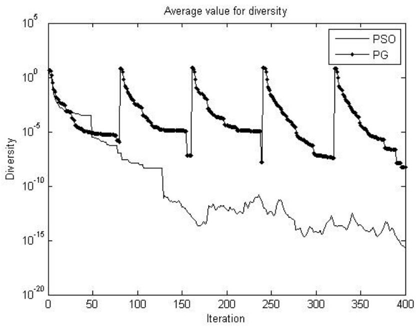


Fig. 2. Average values for diversity during the training process.

The graph shown in Figure 1 represent the average error value obtained by each method in each epoc of training process, whereas for PSO algorithm intermediate values, because it have accomplished 400 iterations, taken to give the same conditions for the two methods.

Is clearly to see that the proposed method is quite superior to the PSO algorithm. Like both use the same heuristic, and more, they have the same implementation, becomes clear the fact that the proposed approach significantly increase the optimization process efficiency, accomplishing the main objective.

An analysis also relevant in this study, is to verify the populations diversity in both methods, what together with the results above, gives an more accurate understanding about the optimization mechanism. The metric here chosen to measure the diversity value, is the variance of the particles fitness. Figure (2) shows the average value of diversity for both methods in each iteration[3] of the training process.

Its is clearly in Figure 2 the great superiority of the proposed method with respect to diversity preservation when compared to the PSO algorithm. This fact is due mainly to restart of the population for each new epoc, which restores its maximum diversity[4].

A joint analysis of both Figures 1 and 2 provides a more complete and mature understanding of the optimization mechanism used by both particles swarms. In the proposed method, the high preservation of diversity, is providing mobility enough to swarm continues finding best solutions, while in the PSO algorithm, the drop in diversity cause a rapid convergence at the beginning of the process, but after some time the population loses its exploratory capacity and becomes incapable of improving its solution.

Thus, is clear that the design here proposed, creates a method with high convergence rate and also high diversity preservation, what are not intrinsic of the PSO algorithm.

Now seeking to compare the efficiency of the PG with the previously mentioned procedural methods, a ANN containing 3 neurons in its hidden layer was used, having the configuration: 3-3-1. The result of the network training can be visualized in the Figure 3 and Figure4.

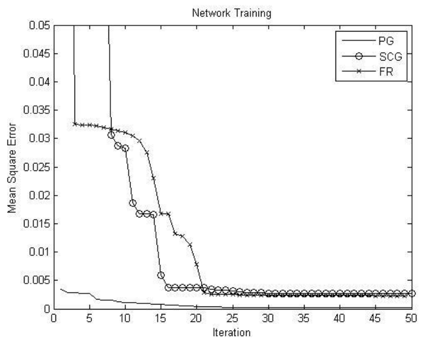


Fig. 3. Average values for mean squared error during the training process.

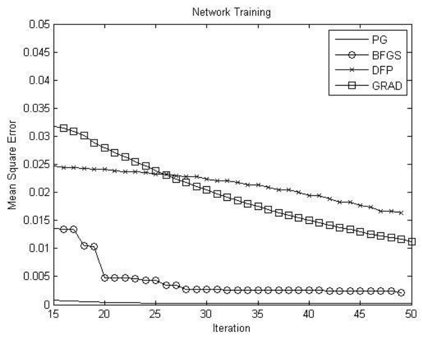


Fig. 4. Average values for diversity during the training process.

[3] Here iteration is used to describe the intermediate steps of training processes, and one iteration of main process, of the proposed method, is referred as epoc

[4] The reader may ask if restarting the population in the method PSO do not lead to the same result, which clearly does not happen for obvious reasons, but these will not be discussed here

Again, due the stochastic characteristic of the PG, these figures show its average behavior for a total of 20 repetitions of the training process. For this example the PG have presented quite superior results when compared with the procedural methods.

In spite of to providing a significant reduction of the network mean square error in each epoc, the proposed algorithm is quite onerous from the computational cost viewpoint, given that a search process should be completed at each iteration. In that way, the execution of the algorithm may become too *slow*, depending of its configuration.

In [15], the author proves the superiority of the method SCG, in respect to computational cost and convergence rate, over the other methods here analyzed. This result is based in the need of unidimensional search required by most methods, that has computational cost $O(N^2)$ per iteration. The SCG method presents a computational cost: $O(2N^2)$ per epoc, that is very inferior to the ones of the methods GRAD, DFP, BFGS and FR that possess computational costs:[5] $O(31N^2)$.

The proposed method presents a total cost $O(n_i n_p N^2)$, that can become quite superior to the ones of the other methods depending on the choice of n_i and n_p. However, the fast convergence of the method compensate, in some cases, this high computational cost.

Now, to build a efficient comparison, let us assume that the function $O(\cdot)$ is linear with respect to the parameters [6] n_p and n_i, we can write:

$$\begin{aligned} C_t^{SCG} &= C_t^{GP} \\ n_e^{SCG} O(N^2) &= n_e^{GP} O(n_i n_p N^2) \\ n_e^{SCG} O(N^2) &= n_e^{GP} n_i n_p O(N^2) \\ n_e^{SCG} &= n_e^{GP} n_i n_p \\ n_e^{GP} &= \frac{n_e^{SCG}}{n_i n_p} \end{aligned} \tag{2}$$

Therefore, if is selected a number of epocs for the algorithm SCG: $n_e^{SCG} n_e =^{GP} n_i n_p$, it is estimated that both algorithms show a very similar computational cost, allowing an analysis of the relative efficiency with respect to convergence rate $\times$ computational cost.

To accomplish this comparison, lets consider the following parameters set:

- Number of epocs: SCG: $n_e^{SCG} = 3200$, GP: $n_e^{GP} = 4$
- Iterations per epoc: $n_i = 60$
- Number of particles: $n_p = 30$
- Number of repetitions: $n_r = 5$

[5] This value is due to the fact that the unidimensional search to makes 30 iterations for each training epoc

[6] This assumption is made to allow a brief analysis of the problem in order to consider the nonlinearity of this function to obtain an accurate result, would not add significant information to present analysis, due the fact that this functions are affected by the methods implementation, and not by its definitions

The Figure 5 shows the ANN mean squared error, after been trained by both methods, in each repetition.

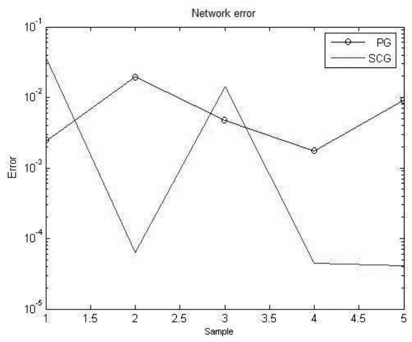


Fig. 5. Comparison between the algorithms: GP and SCG.

In the Figure 5 we can easily see that despite of the proposed method present a higher computational cost, its efficiency rivals the efficiency of SCG method, actually considered one of the most efficient method.

3.2 Curve Fitting

In this example a group of 100 test samples of input-output pairs was used for a quadratic function $y = x^2$, where a white noise of width 10^{-4} was inserted in both signs (input and output). The training was accomplished for various networks configurations, using the proposed algorithm and the algorithms previously mentioned. The results can be observed in Tables 1 and 2.

Table 1. Learning Results: PG SCG FR.

Architecture	PG	SCG	FR
1-3-1	0.00016	0.00066	0.00030
1-6-1	0.00019	0.00069	0.00031
1-9-1	0.00018	0.00078	0.00033
1-18-1	0.00020	0.00082	0.00035
1-6-6-1	0.00024	0.01065	0.01395
1-12-12-1	0.00018	0.01135	0.01735
1-6-12-6-1	0.00017	0.00667	0.05779
1-12-18-12-1	0.00014	0.01764	0.06142

Table 2. Learning Results: BFGS DFP GRAD.

Architecture	BFGS	DPF	GRAD
1-3-1	0.00032	0.00079	0.01012
1-6-1	0.00037	0.00529	0.01197
1-9-1	0.00039	0.00617	0.01207
1-18-1	0.00045	0.00657	0.01287
1-6-6-1	0.01161	0.00959	0.01342
1-12-12-1	0.00943	0.00993	0.01377
1-6-12-6-1	0.00070	0.01060	0.01398
1-12-18-12-1	0.00420	0.01254	0.01485

For this problem it is also possible to notice that the final errors obtained by the proposed method, were also quite inferior to the ones of the others tested algorithms. Another outstanding characteristic observed in the exposed results is the robustness of the PG method with respect to variations in the ANN topology. Due the stochasticity of the learning process, it is possible to infer that the PG method has presented the same final errors results for the several tested configurations.

Several other tests were also accomplished [2], based on different problems with different network topologies, including comparisons with genetic algorithms. In all of this tests was founded similar results to those here presented, confirming all conclusions here taken.

4 Conclusions

The method proposed in this paper represents a new approach for MLP ANNs training using meta-heuristic methods, with different features of others similar methods.

The use of a particle swarm optimization algorithms, as another meta-heuristic methods like genetic algorithms, in ANN training was, until now, not competitive given the inferior performance of these methods when compared to procedural optimization techniques. This new approach, however, is competitive in this scenery, reaching results sometimes comparable with the ones of the *faster* MLP ANN training methods, and still preserving characteristics of the heuristic methods.

One of the main advantages of the PG method, is the possibility to train ANNs with the same efficiency of methods as BFGS and SCG, without knowledge about the error gradient, enlarging its application to several problems, as the one proposed in [4],[2], where the ANN is trained to estimate the state of a dynamic system, and is not possible to calculate the ANN error, and so, the error gradient is unavailable.

Another outstanding characteristic for the proposed method is the preservation of the probability to converge to a global optimum, as in the particle swarm optimization, that is superior from that observed in procedural methods, where it is very probable that they will converge to an local optimum closer to where the method was initiated.

The high computational cost, characteristic of heuristic methods as the genetic algorithms, also is present in the proposed method that is more onerous that the other methods here discussed when implemented in a digital machine. However, it is clear from the shown examples, that this high computational cost is compensated by the accelerated convergence rate. Moreover, the intrinsic parallelism of the proposed method, allows its an implementation in a parallel machine, that can be several orders of magnitude faster than the procedural methods here discussed, inherently sequential.

So we may conclude that the Particles Gradient method here proposed, represents a viable alternative solution for ANNs training, in some situations even being the preferable one, and in a more complex and unusual application, mainly when is difficulty or even impossible the construction of the gradient vector, this method can be one of better choices.

References

1. R. Battiti and F. Masulli. Bfgs optimization for faster and automated supervised learning. *INNC 90 Paris, International Neural Network Conference,*, pages 757–760, 1990.
2. Csar Dalto Berci. *Observadores Inteligentes de Estado: Propostas.* Tese de Mestrado, LCSI/FEEC/UNICAMP, Campinas, Brasil, 2008.
3. Csar Dalto Berci and Celso Pascoli Bottura. Modulao de velocidade em otimizao por enxame de partculas. *Proceedings of 7th Brazilian Conference on Dynamics, Control and Applications. Presidente Prudente, Brasil*, 7:241–248, 2008.

4. Csar Dalto Berci and Celso Pascoli Bottura. Observador inteligente adaptativo neural no baseado em modelo para sistemas no lineares. *Proceedings of 7th Brazilian Conference on Dynamics, Control and Applications. Presidente Prudente, Brasil*, 7:209–215, 2008.
5. Csar Dalto Berci and Celso Pascoli Bottura. Speed modulation: A new approach to improve pso performance. *Accepted Article: IEEE Swarm Intelligence Symposium 2008, Sheraton Westport at St. Louis, Missouri*, 2008.
6. F. den van Bergh. *An Analysis of Particle Swarm Optimization*. PhD thesis, Departament of Computer Science, University of Pretoria, 2002.
7. Jürgen Branke. Evolutionary algorithms for neural network design and training. In *1st Nordic Workshop on Genetic Algorithms and its Applications*, 1995. Vaasa, Finland, January 1995.
8. D.J. Chalmers. The evolution of learning: An experiment in genetic connectionism. *Proceedings of the 1990 Connectionist Summer School*, pages 81–90, 1990.
9. A. Fiszelew, P. Britos, A. Ochoa, H. Merlino, E. Fernndez, and R. Garca-Martnez. Finding optimal neural network architecture using genetic algorithms. *Software & Knowledge Engineering Center. Buenos Aires Institute of Technology. Intelligent Systems Laboratory. School of Engineering. University of Buenos Aires.*, 2004.
10. C. Fyfe. *Artificial Neural Networks*. Department of Computing and Information Systems, The University of Paisley, Edition 1.1, 1996.
11. F. Heppner and U. Grenander. *A Stochastic Nonlinear Model For Coordinate Bird Flocks*. AAAS Publications, Washington, DC, 1990.
12. J. Kennedy and R.C Eberhart. Particle swarm optimization. *Proceedings of IEEE Int. Conf. on Neural Network*, Piscataway, NJ:1942–1948, 1995.
13. D.G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 2nd edition, 1984.
14. M.F. Mller. Learning by conjugate gradients. *The 6th International Meeting of Young Computer Scientists*, 1990.
15. M.F. Mller. A scaled conjugate gradient algorithm for fast supervised learning. *Computer Science Department, University of Aarhus Denmark*, 6:525–533, 1990.
16. D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 762–767, 1989.
17. C.W. Reynolds. Flocks, herds, and schools: A distributed behavior model. *Computer Graphics*, 21(4):25–34, 1987.
18. D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin. *Backpropagation: The basic theory*. Lawrence Erlbaum Associates, Inc., 1995.
19. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation, in: Parallel distributed processing: Exploration in the microstructure of cognition. *Eds. D.E. Rumelhart, J.L. McClelland, MIT Press, Cambridge, MA.*, pages 318–362, 1986.
20. Udo Seiffert. Multiple layer perceptron training using genetic algorithms. *ESANN'2001 proceedings - European Symposium on Artificial Neural Networks*, pages 159–164, 2001.
21. Zhi-Hua Zhou, Jian-Xin Wu, Yuan Jiang, and Shi-Fu Chen. Genetic algorithm based selective neural network ensemble. *Proceedings of the 17th International Joint Conference on Artificial Intelligence.*, 2:797–802, 2001.

Data-mining Approaches for the Study of Emotional Responses in Healthy Controls and Traumatic Brain Injured Patients: Comparative Analysis and Validation

F. Riganello[1], L.Pignolo[1], V. Lagani[2] and A. Candelieri[2]

[1] S. Anna Institute, RAN – Research on Advanced Neuro-rehabilitation, Crotone, Italy
{f.riganello,l.pignolo}@istitutosantanna.it
[2] Laboratory for Decision Engineering and Health Care Delivery
Department of Electronic Informatics and Systemistics
University of Calabria, Cosenza, Italy
{vlagani,candelieriantonio}@yahoo.it

Abstract. Relationship between Heart Rate Variability (HRV) and emotions subjectively reported by 26 healthy subjects during symphonic music listening have been investigated through Data Mining approaches. Most reliable decision models have been successively adopted to forecast an emotional assessment on a group of 16 Traumatic Brain Injured patients during the same type of stimulation, without algorithms retraining. The most performing decisional models have been a Rule Learner (ONE-R) and a Multi Layer Perceptron (MLP) but, comparing them, the first one was the best in terms of reliability both on validation and independent test phases. Furthermore, ONE-R provides a simple "human-understandable" rule useful to evaluate emotional status of a subjects depending only on one HRV parameter: the normalized unit of Low Frequancy BandPower (nu_LF). Specifically, the classification by HRV nu_LF matched that on reported emotions, with 76.0% of correct classification; tenfold cross-validation: 70.2%; leave-one-out validation: 71.1%. On the other hand, MLP approache has provided an accuracy of 82.69% on healthy controls, but it has decreased to 47.11% and 46.15% on 10folds-cross and leave-one-out validation respectively. Finally, the accuracy has resulted in 51.56% when the MLP model has been applied to the posttraumatic subjects, while the ONE-R accuracy has resulted in 70.31%. Data mining proved applicable in psychophysiological human research.

1 Introduction

Data-mining or hybrid techniques are used in medicine to sort significant information out of large databases in mutagenicity studies, predictive toxicology, disease classification, selective integration of multiple biological databases, etc. Applications in neurology have focused on prognostic studies [1-2] or in the classification of emotional responses [3] . The Heart Rate Variability (HRV) is an emerging objective measure of the continuous interplay between the sympathetic and parasympathetic autonomic nervous sub-systems. It is thought to provide information also on complex patterns of brain activation, including emotional responses [4-11]. For instance, HRV abnormalities are reportedly common in psychiatric or brain damaged patients [12-14]; HRV proved a useful predictor of outcome in brain injured patients [15-16]. HRV studies require quantitative approaches and large numbers of parameters are generated when the parametric and non-parameric HRV spectra are computed.

In our study we measured several HRV parameters from 26 healthy subjects during the listening of music samples, suitably selected to induce specific emotional status. At the end of each listening we asked to report felt emotions, without any reference to pre-selected categories and to the feelings that the subjects thought the music was intended to induce. On the basis of these interviews, we identified to macro-categories of emotions (positive and negative emotions) and defined a two-classes classification problem. The same procedure has been adopted for acquiring data from a group of 16 posttraumatic subjects. More details about the two datasets construction are reported in another our study [3]. Since the great amount of variables (35 HRV parameters) and the relatively weak consolidated knowledge about the problem, data mining techniques have represented an effective solution for identifying a relationship between HRV parameters and emotions, without any preliminary assumption about data distribution.

We compared different classification learning strategies through suitable validation techniques, such as 10folds-cross and leave-one-out validation and test on the independent test set.

2 Material and Methods

2.1 Patients and Controls

Two groups of subjects were studied:

1. twenty six healthy volunteers (14 women; mean age: 31.7±7.1 yrs., age range: 21-45 yrs.; high-school to university education);
2. sixteen patients without severe disabilities completing rehabilitation after severe traumatic brain injury (5 women; mean age: 21.6±3.0 yrs., age range: 17 to 33 yrs., grammar to high school education); and

Subjects were informed in full detail about the study purpose and experimental procedures and the ethical principles of the Declaration of Helsinki (1964) by the World Medical Association concerning human experimentation were followed. The procedures for data collection and the experimental setting caused no physical or emotional discomfort and were discontinued whenever the subject felt tired or tense or at the subject's request. Controls and posttraumatic subjects had no musical training (see [3] for methodological detail).

2.2 Stimulus Conditions

Four music samples (Table 1) were selected following characterization by intrinsic structure and expected emotional response as indicated by the available formal complexity and dynamics descriptors. These descriptors reportedly relate music structure to self-assessed emotions and allow to characterize the emotional status along a continuum from euphoria and well-being to melancholy, severe anxiety and perceived aggressive tendencies [17-18].

Table 1. Selected music samples.

1) Luigi Boccherini: Quintet op. 11 n. 5, *Minuetto* (duration: 3' and 50");
2) Piotr Ilitch Tchaïkovski: sixth symphony, op. 74, *first movement* (duration: more than 10");
3) Modest Petrovich Mussorgsky: *St. John's Night on the Bald Mountain* (duration: more than 10");
4) Edvard Grieg: Peer Gynt, op 23, *The morning* (duration: 4' and 20")

Experiments on patients and control took place at the same time of the day in a familiar environment and did not interfere with the posttraumatic subjects' medical/rehabilitative schedules. Subjects were comfortably lying on an armchair, with constant 24^{o} C ambient temperature and in absence of transient noises. They were exposed binaurally (earplugs) to the four selected music samples balanced for loudness and played in random sequence to minimize carry-on effects. There was a 20-min rest between consecutive samples to avoid overstimulation and excessive fatigue.

The heart beat was recorded from the beginning of the music sample and for a total of 300 beats (3',36"±24", with 83.7± 9.5 beats/min and a resulting total recording time between 3',12" and 3':55") by means of the *Virtual Energy Tester* (*Elamaya Instruments*, Milano, Italy). The photopletismographic sensors were positioned on the third phalange of the left hand middle finger in order to minimize the subjects' discomfort consistent with the guidelines of the Task Force of European Society of Cardiology and the North American Society of Pacing and Electrophysiology[19]. The photopletismographic signal was sampled at 100 samples/sec; the series of consecutive intervals between heart beats was analyzed in the time and frequency domains by the *HRV advanced analysis software* developed at the Department of Applied Physics, University of Kuopio, Finland [20]. The non-parametric (Fast Fourier Transform, Welch spectrum) and parametric (autoregressive) spectra were computed (Table 2). The power spectral density from 0.01 Hz to 0.5 Hz was computed with 0.001 Hz resolution and three frequency ranges (very low frequency [VLF]: 0.01-0.04 Hz; low frequency [LF]: 0.04-0.15 Hz; and high frequency [HF]: 0.15 Hz – 0.5 Hz) were considered (Table 2).

At the end of each music sample, controls and posttraumatic subjects were requested to classify their emotions, without reference to any pre-selected categories and irrespective of the emotional feeling they thought the music was intended to induce. The distribution of the emotions expressed for each music sample was determined [17-18].

Table 2. Spectral and Statistical Parameters.

Statistical Parameters	Spectral Parameters
Mean RR interval (Mean RR) and SD (STD RR);	Very Low Frequency (VLF), Low Frequency (LF), High Frequency (HF) and normalized unit (nu) in FFT and autoregressive spectra
Mean Heart Rate (Mean HR) and SD (STD HR);	VLF Peak frequency in FFT and autoregressive spectra
Root mean square of SD (RMSSD);	LF Peak frequency in FFT and autoregressive spectra
number (NN50) and percentage (pNN50) of NN intervals longer than 50 ms	HF Peak frequency in FFT and autoregressive spectra
	Power Spectrum of VLF, LF, HF and Total in FFT and autoregressive spectra
	% of VLF, LF, HF in FFT and autoregressive spectra
	Ratio LF/HF, nu LF, nu HF, nu LF/HF in FFT and autoregressive spectra

2.3 Data Mining Techniques

We adopted several classification approaches for identifying an association between HRV parameters and the self-assessed emotions. To this aim, two different classes or categories were defined for the reported emotional status: *positive* (happiness, joy, serenity, calm,…) and *negative* (fear, anxiety, tension, scare,…). The control group was selected as training set for several *data-mining* classification techniques, such as Decision Trees, Support Vector Machines, Artificial Neural Networks and Rules Learner, provided by the WEKA open source software (Waikato Environment for Knowledge Analisys) [21, 22].

Training set included 104 cases (26 healthy subjects x 4 music samples) and 35 variables (Table 2). In a first step, two internal validation techniques based on training data (namely the 10 folds-cross and leave-one-out validation) were used to evaluate how much extracted models would fit the new data.

Most reliable decision models have been selected and adopted to forecast an "*emotional status assessment*" on the independent test set (16 posttraumatic subjects). The models used to obtain this assessment on the test set has been not "re-learned" from it. The independent test set included 64 cases (16 posttraumatic subjects x 4 music samples) and the aforementioned 35 HRV parameters.

In this way, we have been able to estimate "real" reliabilities for the selected models. Figure 1 shows how we have designed our study.

Since every classification technique adopts a different structure for explaining the relationship between input variables and output (HRV parameters and subjectively reported emotions, respectively in our study) it may be really difficult to select *a priori* a strategy for learning the relationship from the available data. For such reason we decided to adopt several classification learning procedures applicable to our problem, such as Decision Trees, Support Vector Machines, Rule Learners and Artificial Neural Networks, all available in WEKA.

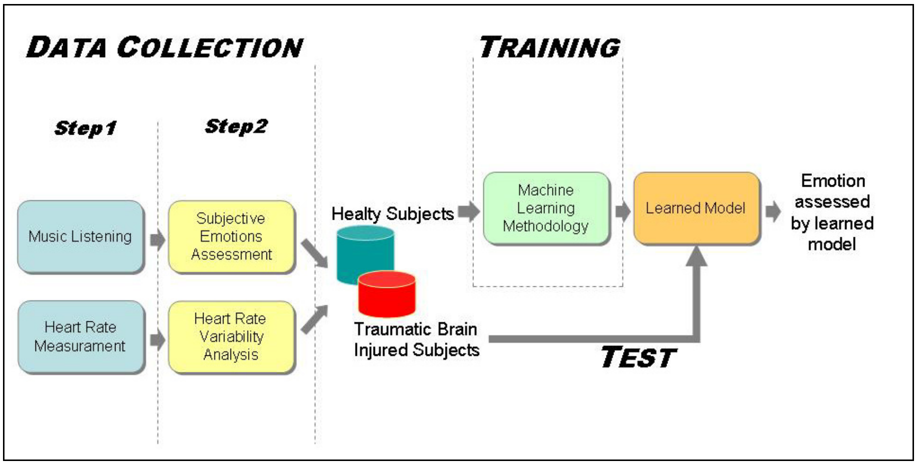

DATA COLLECTION
TRAINING
Step1
Step2
Music Listening
Subjective Emotions Assessment
Heart Rate Measurament
Heart Rate Variability Analysis
Healty Subjects
Traumatic Brain Injured Subjects
Machine Learning Methodology
Learned Model
Emotion assessed by learned model
TEST

Fig. 1. Phases of the Study.

Respect to 10fold-cross and leave-one-out validation, we have obtained that the most reliable classification techniques were ONE-R [23] and Multi Layer Perceptron [24]. The first one is a Classification Rule Learner and works in a very easy way: it accepts a training set as input and searches for a “1-rule” classifying instances on the basis of a single variable (attribute). Initially, the algorithm ranks attributes according to error rate on the training set then, if the attribute is numerical, the algorithm divides the range of possible values into several disjoint intervals. In order to avoid overfitting (that is when each interval contains only a value belonging to one class) ONE-R permits to set a parameter named “bucket-size” that is the minimum number of instances in an interval. We obtained the best performing ONE-R configuration setting 7 as minimum number of instances in one interval. In figure 2 we report the extracted rule (note that *nu_LF* can belong to only one interval, so only one prediction is possible).

If **nu_LF** *is in*	[0; 29.35)	negative emotion predicted
	[29.45; 46.37)	positive emotion predicted
	[46.37; 64.25)	negative emotion predicted
	[64.25; 72.45)	positive emotion predicted
	[72.45; 100]	negative emotion predicted

[. ; .) inferior extreme included – superior extreme excluded

Fig. 2. Decision model for emotional status assessment learned through ONE-R algorithm on healthy controls.

Multi Layer Perceptron (MLP) is a particular Artificial Neural Network (ANN) architecture also known as “feed forward architecture”, and it is composed by, at the least, three different layers: input layer, hidden layer and output layer. Generally, the hidden layer can be more then one and also the number of neurons into hidden layers can also vary, where a neuron is the “simple” processing unit of the net.

As each ANN, MLP is a mathematical/computational model based on biological neuronal networks and is commonly applied to model complex input/output relationships or to identify data patterns of distribution/correlation. Structurally, an ANN is composed by a set of neurons, linked each other by a large number of (usually nonlinear) weighted connections. Each neuron is able to calculate a specific function, given the inputs and the weights on the connections are adjusted in order to minimize some criteria as, usually, the errors number.

Using MLP we have obtained the best results setting up the following net parameters: only one the hidden layer with 5 neurons, Learning Rate=0.2, Momentum=0.1 and 1000 training epochs. Moreover we have previously ranked the variables by preprocessing the dataset by ONE-R WEKA Filter, which evaluates the worth of an attribute by using the ONE-R strategy. Finally, the best results for MLP were obtained selecting the first 8 ranked attributes (nu_LF, powerHF, STDRR, gender, powerVLF, peakVLF, peakLF and peakHF). The MLP architecture is showed in figure 2.

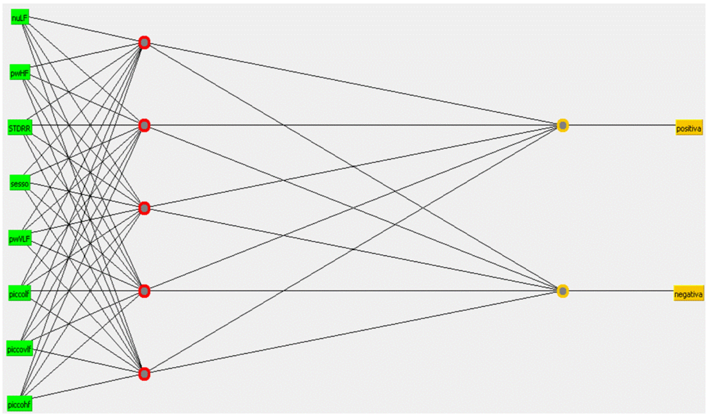


Fig. 3. Multi Layer Perceptron architecture.

3 Results

ONE-R procedure sorted out the nu_LF descriptor as the more significant spectral parameter in the selected experimental conditions and for the study purpose. The ONE-R allowed a good trade-off between the portion of correct sample/label association on the entire training set (recognition) and the portion of correct sample/label association on validation and independent test phases (generalization).

In table 3 we summarize our results both for ONE-R and MLP. For the first one, an overall correct classification on training set (healthy controls) has been obtained on about 76.0% of the instances, 81.0% and 69.0% of negative and positive emotions, respectively; (tenfold cross-validation: 70.2%; leave-one-out validation: 71.1%).

When applied to the independent test set (posttraumatic subjects), the classification accuracy performed by the ONE-R decision model was comparable: 70.3% of correct classification on the entire test set, 65% and 74% of positive and negative emotions, respectively.

Despite the greater accuracy of the MLP on the entire training set (82.7% vs. 76.0%, for MLP and ONE-R respectively), MLP accuracy decreased to 47.11% and 46.11% on 10folds-cross and leave-one-out validation phases respectively. Moreover, also the correct attribution for each class of emotion has decreased: 33.33% and 57.63% for positive and negative emotions, respectively, on 10folds-cross validation, and 26.67% and 61.02% for positive and negative emotions, respectively, on leave-one-out validation.

Furthermore, also on independent test phase MLP has provided lower reliability than ONE-R approaches: 70.31% of accuracy for ONE-R versus 51.56% for MLP (32% and 64.10% for positive and negative emotions respectively). All the results are summarized in Table 3.

Table 3. Results One-R vs MLP.

Analysis of data: ONE-R vs MLP				
	On training data	**10 Folds-cross**	**Leave-one-out**	**Independent Test Set**
OneR	75.96%	70.19%	71.15%	70.31%
MLP	82.69%	47.11%	46.15%	51.56%
Attribution of positive emotions				
	On training data	**10 Folds-cross**	**Leave-one-out**	**Independent Test Set**
OneR	68.89%	57.78%	64.44%	68.00%
MLP	68.89%	33.33%	26.67%	32.00%
Attribution of negative emotions				
	On training data	**10 Folds-cross**	**Leave-one-out**	**Independent Test Set**
OneR	81.36%	79.66%	76.27%	71.79%
MLP	93.22%	57.63%	61.02%	64.10%

4 Comment

Although the HRV is an emerging objective measure also in neurophysiology, there is still a lack of knowledge about a possible and tangible relationship between heart activity and the emotional status assessment. On the other hand, Data Mining techniques may offer practical advantages for analyzing data, also medical [25, 26], and they have proved useful analysis tools in our study, searching for the most reliable and frequent relationships between HRV parameters measured during symphonic music listening and emotions subjectively reported by a group of 26 healthy subjects at the end of each listening. Several data mining methods have been investigated and evaluated through the most suitable validation techniques. Reliability of resulting relationships has been then tested on an independent test group of 16 posttraumatic patients, without algorithm retraining.

ONE-R algorithm (a classification rule learner) has provided the best performances and reliability, identifying a single HRV parameter (notably the *nu_LF* measure) as the most relevant for assessing the emotional status, both for healthy controls and posttraumatic subjects.

In this study ONE-R proved more effective then the best MLP configuration and provided a simple *"if…then"* rule. Furthermore, this rule can be easily applied, in combination with the non-invasive technique for HRV data acquisition (by a photopletismographic sensor), to evaluate the emotional conditions of unconscious subjects (such as subjects in vegetative state) in order to establish, in a more objective way, when is better to continue or interrupt any contact or stimulation.

References

1. Herskovits, H. E., Joan, P. G.: Application of a data-mining method based on Bayesian networks to lesion-deficit analysis. 19(4) (2003) 1664-1673.
2. Chen, S., Haskins, E. W., Ottens, K. A., Hayes, L. R., Denslow, N., Wang., K. W. K.,: Bioinformatics for Traumatic Brain Injury: Proteomic Data Mining.: Data Mining in Biomedicine. 7 (2008) 363-387.
3. Riganello, F., Quintieri, M., Candelieri A., Conforti, D., Dolce G.: Heart Rate Variability: an artificial intelligence study on healthy and traumatic brain injured subjects: Journal of Psychophysiology. 22(3) (2008) 166-174.
4. Lafranchi, P. A., & Somer, V.K.: Arterial baroflex function and cardiovascular variability: Interactions and implications. American Journal of Physiology – Regulatory Integrative and Comparative Physiology. 283 (2003) R815–R826.
5. Malpas, S. C.: Neural influences on cardiovascular variability: Possibilities and pitfalls. American Journal of Physiology – Regulatory Integrative and Comparative Physiology. 282 (2002) H6–H20.
6. Nikki, S. R.: Intense emotional response to music: A test of the physiological arousal hypothesis. Psychology of Music. 32 (2004) 371–388.
7. Urakawa, K., & Yokoyama, K.: Music can enhance exercise- induced sympathetic dominancy assessed by HRV. Tohoku Journal of Experimental Medicine. 205 (2005) 213–218.
8. Mashin, V. A., & Mashina, M. N.: Analysis of the HRV in negative functional states in the course of the psychological relaxation session. Human Physiology. 26 (2000) 420–425.
9. Critchley, H.D., Rotshtein, P., Nagayi, Y., O'Doherty, J., Mathias, C.J., & Dolan, R.: Activity in the human brain predicting heart rate response to emotional facial expression. Neuro- Image. 24 (2005) 751–762.
10. Fraizer, T. W., Strauss, M.E., & Steinhauer, S. Respiratory sinus arrhythmia as an index of emotional response in young adults. Psychophysiology. 41 (2004) 75–83.
11. Appelhans, B. M., & Luecken, L.J.: Heart rate variability as an index of regulated emotional responding. Review of General Psychology. 10 (2006) 229–240.
12. Cohen, H., Benjamin, J., Geva, A.B., Matar, M.A., Kaplan, Z., & Kotler, M.: Autonomic dysregulation in panic disorder and in posttraumatic stress disorder: Application of power

spectrum analysis of heart rate variability at rest and in response to recollection of trauma or panic attacks. Psychiatry Research. 96(1) (2000) 1–13.

13. Draper, K., Ponsford, J., & Schönberger, M.: Psychosocial and emotional outcomes 10 years following traumatic brain injury. Journal of Head Trauma Rehabilitation. 22 (2007) 278–287.
14. Scholten,M. R., van Honk, J., Aleman,A.,&Kahn, R.S.: Behavioral inhibition system (BIS), behavioral activation system (BAS), and schizophrenia: Relationship with psychopathology and physiology. Journal of Psychiatric Research. 40 (2006) 638–645.
15. Briswas, A. K., Scott, W.A., Sommerauer, J.F., & Luckett, P.M.: Heart rate variability after acute traumatic injury in children. Critical Care Medicine. 28 (2000) 3907–3912.
16. Keren, O., Yapatov, S., Radai, M.M., Elad-Yarum, R., Faraggi, D., Abboud, S.: Heart rate variability of patients with traumatic brain injury during postinsult subacute period. Brain Injury. 19 (2005) 605–611.
17. Imberty, M.: Epistemic subject, historical subject, psychological subject: Regarding Lerdhal and Jackendoff 's generative theory of music. In I. Deliege & J.A. Sloboda (Eds.), Perception and cognition of music Hove, UK: Psychology Press. (1997) 429–432.
18. Tarasti, E.: A theory of musical semiotics. Bloomington, IN: Indiana University Press. (1994).
19. Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology of Circulation.: Heart rate variability: Standard of measurement, physiological interpretation, and clinical use. Circulation. 93 (1996) 1043-1065.
20. Niskanen, P.J., Tarvainen , M.P., Ranta-aho, P.O., & Karjalainen P.A.: Software for advanced HRV analysis. University of Kuopio Department of Applied Physics. Computer Methods and Programs Biomedicine. 76(1) (2004) 73-81.
21. Witten, H.W., & Eibe, F.: Data mining – Practical machine learning tools and techniques with Java implementations. San Francisco, CA: Morgan Kaufman. (2005).
22. Eibe, F.: Machine learning with WEKA. Department of Computer Science, University of Waikato, New Zealand. (2004). http://puzzle.dl.sourceforge.net/sourceforge/weka/weka.ppt
23. Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. Machine Learning. 11 (1993) 63–90.
24. van Bemmel, J.H.,&Munsen, M.A.: Handbook of medical informatics. Berlin: Springer-Verlag. (1997).
25. C. Robert, C.D. Arreto, J. Azerad, J.F. Gaudy. *Bibliometric overview of the utilization of artificial neural networks in medicine and biology.* Scientometrics, (2004) Vol. 59, No. 1 (117.130).
26. ChungK Lee, SK Yoo, YoonJ Park, NamHyun Kim, KeeSam Jeong, ByungChae Lee (2005). *Using Neural Network to Recognize Human Emotionsfrom Heart Rate Variability and Skin Resistance,* Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference Shangai, China, September 1-4, 2005.

Image Feature Significance for Self-position Estimation with Variable Processing Time

Kae Doki, Manabu Tanabe, Akihiro Torii and Akiteru Ueda

Dept. of Mechanical Engineering, Aichi Institute of Technology
1247 Yachigusa, Yakusa-cho, Toyota, Aichi, Japan
{doki,torii,aueda}@aitech.ac.jp

Abstract. We have researched about an action planning method of an autonomous mobile robot with a real-time search. In the action planning based on a real-time search, it is necessary to balance the time for sensing and time for action planning in order to use the limited computational resources efficiently. Therefore, we have studied on the sensing method whose processing time is variable and constructed a self-position estimation system with variable processing time as an example of sensing. In this paper, we propose a self-position estimation method of an autonomous mobile robot based on image feature significance. In this method, the processing time for self-position estimation can be varied by changing the number of image features based on its significance. To realize this concept, we conceive the concepts of the significance on image features, and verify three kinds of equations which respectively express the significance of image features.

1 Introduction

An autonomous mobile robot is one of the most interesting targets in the field of the robotics. It is very expected not only in the industrial field but also in the community like an office, a hospital and a house in the future. Among various kinds of problems for an autonomous mobile robot, we have focused on and researched about its action planning methods with the real-time search [1][2]. In the action planning with the real-time search, a robot action is acquired through the recognition of the current situation and the action search on the ground. Therefore, the computational resource for the action planning is limited, and the situation around a robot changes every moment in a dynamic environment where the robot moves. Under these circumstances, we have to consider the balance between the time for recognition and the time for action search according to the situation around the robot in order to utilize the limited computational resource efficiently. This idea is based on Anytime Sensing which has been proposed by S.Zilberstein et.al[3], and one of crucial features in the action planning with the real-time search. As an example of recognition, we deal with a self-position estimation problem for the robot with vision and construct a self-position estimation system with variable processing time based on the above idea.

In this system, it is assumed that the robot estimates the current self-position by matching an acquired input image with stored images which indicate certain positions in the environment. The normalized correlation coefficient [7] is applied as a criterion

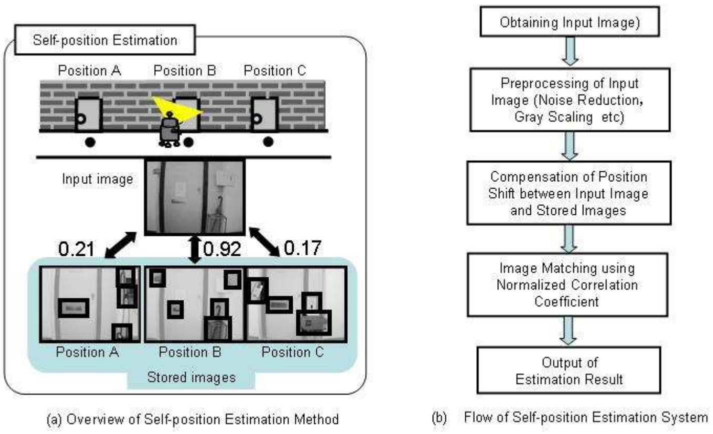

Self-position Estimation
Position A
Position B
Position C
Input image
0.21
0.92
0.17
Position A
Position B
Position C
Stored images
(a) Overview of Self-position Estimation Method
Obtaining Input Image)
Preprocessing of Input Image (Noise Reduction, Gray Scaling etc)
Compensation of Position Shift between Input Image and Stored Images
Image Matching using Normalized Correlation Coefficient
Output of Estimation Result
(b) Flow of Self-position Estimation System

Fig. 1. Self-position Estimation System.

of the template matching. In order to use the limited computational resource efficiently, only image features in a stored image are used in the self-position estimation process. Moreover, the processing time for the self-position estimation is varied by changing the number of the image features utilized for the template matching. In this case, it is important that the number of the image features is varied according to priority in order not to deteriorate the performance of the self-position estimation.[5][6] To realize this idea, it is essential to decide how important each image feature is for the self-position estimation. By changing the number of the image features based on image feature importance, stable self-position estimation can be realized even if the time for the self-position estimation is shortened. According to these reasons, a new self-position estimation method with variable processing time is proposed based on image feature significance in this research. In this paper, the concepts based on image feature importance are conceived, and three types of equations are defined as an indicator of significance on image features. These equations are compared through some experimental results.

2 Self-Position Estimation with Variable Processing Time

2.1 Self-position Estimation System

As a framework of a self-position estimation problem, the robot moves in the indoor environment and estimates which key location it is around in the environment. It obtains images at key locations and image templates are generated with them as a preparation for the self-position estimation.At the self-position estimation, an input image is obtained and it is processed through image processing for noise reduction and gray scaling. Then, an input image matches stored image templates and the estimation result is output. In the case of Fig.1(a), the highest value of the similarity is one with the position B. Therefore, the output is the position B as the estimation result. In this paper, the normalized correlation coefficient(NCC) is applied as a criterion for the similarity.

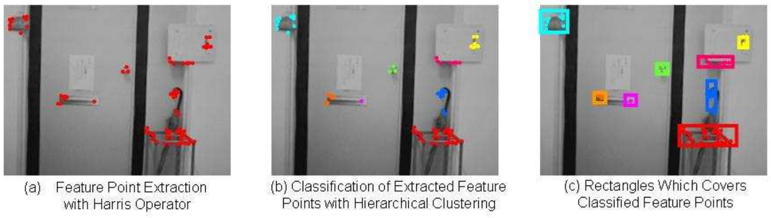

(a) Feature Point Extraction with Harris Operator
(b) Classification of Extracted Feature Points with Hierarchical Clustering
(c) Rectangles Which Covers Classified Feature Points

Fig. 2. Generation Method of Image Template.

$$R = \frac{\sum_x^M \sum_y^N \{f(x,y) - \bar{f}\} \cdot \{g(x,y) - \bar{g}\}}{\sqrt{\sum_x^M \sum_y^N \{f(x,y) - \bar{f}\}^2} \sqrt{\sum_x^M \sum_y^N \{g(x,y) - \bar{g}\}^2}} \qquad (1)$$

Eq.1 indicates the normalized correlation coefficient between two images F and G. They are grey-scale images whose size is $M \times N$. The parameters $f(x,y)$ and $g(x,y)$ are the brightness of the pixel at the coordinate (x,y) in each image F and G. The parameters $\bar{f}$ and $\bar{g}$ are the mean value of $f(x,y)$ and $g(x,y)$ respectively. When $f(x,y)$ is entirely identical to $g(x,y)$, the value of R is 1.0. On the other hand, when the image $f(x,y)$ is completely different from $g(x,y)$, R is -1.0. Here, it is assumed that $f(x,y)$ is an input image obtained at the current position of the robot, and that $g(x,y)$ is an image template stored in the robot. By the image template matching using this equation, it is estimated that the robot is around the position which is indicated by the image template with the highest value of R in the stored image templates.

2.2 Changing Method of Processing Time

The time for the image template matching occupies most of the processing time for the self-position estimation in this system. This is because NCC is the pixel-based calculation which requires computational resources. According to the above equation, the computational amount for calculating R is in proportion to the number of pixels. Therefore, only image features are utilized for the self-position estimation, and the number of the image features is changed in order to change its processing time.

The image features are extracted as some rectangle areas from a stored image through the image processing. shows. A set of extracted rectangles is called as an image template in this paper, and template generation method is described in the next section. The similarity between an input image and an image template is the average of NCC value on all rectangle areas utilized for the self-position estimation.

2.3 Generation Method of Image Templates

Fig.2 shows the generation method of the image template for the self-position estimation. First of all, feature points in each stored image are extracted with Harris operator[8] which is widely utilized for image feature extraction. As Fig.2(a) shows, a mass of feature points is extracted around the distinctive parts in a stored image. Thus, these points are grouped with hierarchical clustering as Fig.2(b). Then, a rectangle is created

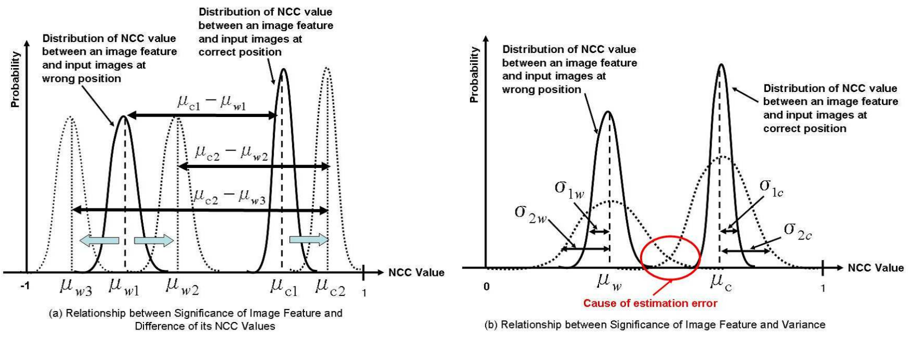


Fig. 3. Image Feature Importance from Viewpoint of Self-poisition Estimation.

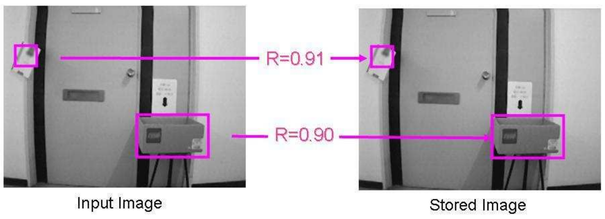


Fig. 4. Image Feature Importance from Computational Resources.

which covers a group of feature points as Fig.2(c) shows. These rectangle areas are utilized for image features for the self-position estimation.

3 Addition of Significance to Image Features

3.1 Concept of Image Feature Significance

The processing time for the self-position estimation is varied by changing the number of the image features utilized for the template matching in this system. In this case, the more effective image feature for the self-position estimation should be given higher preference over the others. For example, when the processing time is short, only crucial image features to the self-position estimation are utilized to guarantee the quality of the self-position estimation. On the other hand, more image features are used for the self-position estimation to realize robust self-position estimation when the processing time is long. In order to realize the above idea, it is essential to decide how important each image feature is for the self-position estimation. Therefore, the following three concepts are conceived to define the importance of each image feature for the self-position estimation.

Fig.3 shows the relationship between the NCC values for one image feature and its significance. The horizontal axis is a NCC value between an image feature and an obtained input image, and the vertical axis is its probability. Input images are influenced by the robot motion, therefore NCC values changes for the same image feature.

In the self-position estimation system, the current position is estimated as one which is indicated by the image template with the highest similarity for an input image. Therefore, the higher NCC values for the images obtained at correct position are better for the self-position estimation on an image feature. Moreover, the higher NCC values for the images obtained at wrong positions is worse for the self-position estimation on the image feature even if the NCC value is pretty high as Fig.3(a) shows. Therefore, the difference between NCC values at correct and wrong positions is important for the self-position estimation in addition to the absolute value of NCC at the correct position. Moreover, when the NCC values widely changes for various input images, the distribution of NCC value at the correct position overlaps the one at the wrong positions as Fig.3(b). This overlap causes the error of the self-position estimation. Therefore, the variance of NCC value should be also considered for stable self-position estimation.

Fig.4 shows the relationship between the size of the image feature and its significance. This is the concept from the viewpoint of the computational resources. As mentioned in Section 1, the computational resources is limited in the action planning with the real-time search, therefore it is important to utilize the computational resources efficiently. Here, let two rectangles drawn in Fig.4 stand for image features for the estimation of a certain position. As mentioned in Section 2.2, the computational amount for the self-position estimation is in proportion to the number of pixels of an image feature. Therefore, if the NCC values of these two image features are very similar as this figure shows, it is apparent that the small image feature is better than the large one. According to this reason, the size of image features should be considered on the image feature significance.

3.2 Equations of Image Feature Significance

Based on the above concepts, the three kinds of equations are formulated as a indicator of significance on image features.

$$E_1 = k_1\mu_c + (\mu_c - \max_j \mu_{w(j)}) \tag{2}$$

$$E_2 = k_1\mu_c + \{(\mu_c - k_2\sigma_c) - \max_j(\mu_{w(j)} + k_2\sigma_{w(j)})\} \tag{3}$$

$$E_3 = E_2 - k_3\frac{S_p}{\sum_{q=1}^{h} S_q} \tag{4}$$

μ_c, μ_w	:	mean value of NCC of correct or wrong images
σ_c, σ_w	:	variance of NCC of correct or wrong images
j	:	number of positions
$k_1 \sim k_3$	:	weight
S_p	:	number of pixels of image features
h	:	number of image features

Eq.(2) indicates the significance on an image feature which considers only the mean value of NCC at the correct position and the difference of NCC values between the correct position and wrong ones.In this equation, the first term is the absolute value of NCC for an input image obtained at the correct position. The second term is the difference of NCC values between the correct and wrong positions. k_1 is a weight in order to balance these two parameters. The larger this value becomes, the more important the image feature becomes for the self-position estimation. Eq.(3) indicates the significance on an image feature which considers the variance of NCC values at the correct and wrong positions.The larger the variance of NCC values becomes, the smaller the second and third terms become. Eq.(4) indicates the image feature significance which considers the computational amount for the self-position estimation in addition to the quality of the self-position estimation. The second term is a percentage of the pixel of an image feature for all image features. Therefore, the second term becomes large when the size of an image feature becomes large.

4 Experimental Results and Verification

4.1 Experimental Setup

In order to examine the proposed method, we performed some experiments. First of all, six image templates were generated which respectively distinguished from the position A to the position F. The value of significance was calculated for each image feature in a image template according to the above equations. The weights in Eq.2, Eq.3 and Eq.4 were 0.14, 1.20, and 1.20 respectively. These parameters were decided experimentally. Then, a real robot performed the self-position estimation with the acquired image templates in a real environment. In this experiment, the number of the image features was changed from one up to five according to the priority given by each equation of the image feature significance. The self-position estimation was performed a hundred times at each key location.

4.2 Generated Image Templates

Fig.5 shows the image templates which were generated with the method in Section 2.3. As this figure shows, rectangles which cover distinctive parts in each stored image were generated as an image template. Fig.6 shows the top five rectangles ranked by each equation expressed in Section 3.2 for the templates of Position E. This is because the rank of the image features in this image template changed characteristically. As this figure shows, unique image features have higher priority in each image template. Moreover, comparing image features ranked by Eq.(2) with ones by Eq.(3), the ranking of some image features changes. This is because image features which are ranked lower in Eq.(3) are much influenced by the robot motion, therefore, the variance of NCC values becomes large. In addition, image features with a larger size are ranked lower by Eq.(4) in comparison with ones ranked by Eq.(4). This is because larger image features are ranked by Eq.(4) from the viewpoint of the computational resources.

4.3 Performance of Self-position Estimation

In order to examine the image feature significance ranked by Eq.(2), Eq(3) and Eq.(4), the experiment of the self-position estimation with a real-robot was performed. In this

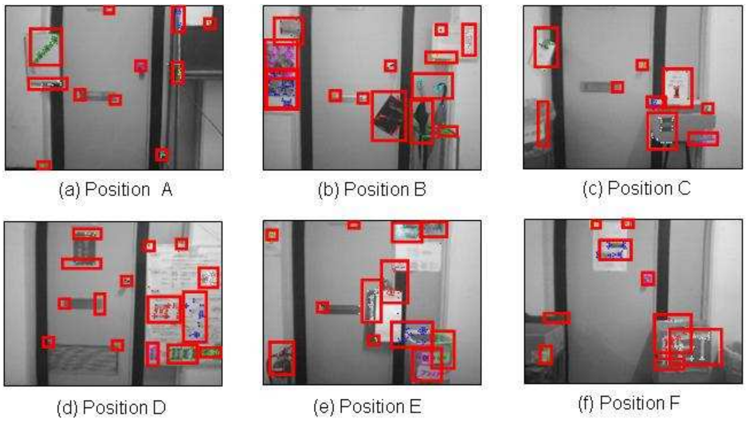


Fig. 5. Generated Templates.

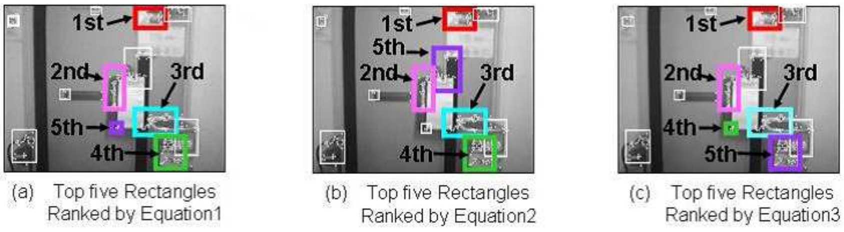

1st
2nd
3rd
4th
5th
(a) Top five Rectangles Ranked by Equation1
1st
5th
2nd
3rd
4th
(b) Top five Rectangles Ranked by Equation2
1st
2nd
3rd
4th
5th
(c) Top five Rectangles Ranked by Equation3

Fig. 6. Comparison of Image Feature Ranking(Position E).

experiment, the number of the image feature changed according to the priority ranked by each equation, and the self-position estimation was performed with only selected image features. The robot succeeded the estimation of its current position at all positions in all trials. In order to compare the quality of each equation, the relationship between the performance of the self-position estimation and each equation on image feature significance was examined. In addition, the relationship between the processing time and each equation on image feature significance was also examined.

Fig.7 shows the difference of NCC values between the correct and wrong positions on the image template of the position E. The larger this value becomes, the more clearly the image template can distinguish the correct position from wrong ones. According to this result, all image templates have almost same quality Fig.8 shows the processing time for the self-position estimation when the number of the image features in an image template changes. As Fig.8 shows, the processing time with the image template ranked by Eq.4 is the shortest in all equations. This is because Eq.4 considers the computational resources for the self-position estimation. According to these results, Eq.4 is the best indicator of image feature significance.

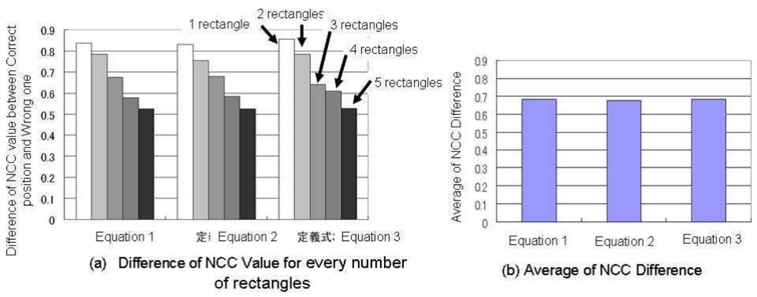


Fig. 7. Difference of NCC values between the correct and wrong positions for the template.

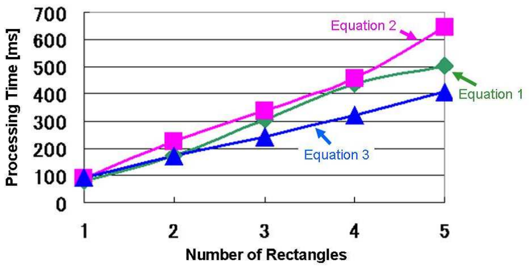


Fig. 8. Processing Time for Self-position Estimation.

5 Conclusions

A new self-position estimation method with variable processing time was proposed based on image feature significance in this research. It was demonstrated that the stable and efficient self-position estimation could be realized with an image template based on image feature significance even if the number of the image feature was changed. For the future work, the self-position estimation with an omni-directional image will be tackled based on the proposed method. Moreover, it will be examined how the environmental change influences the performance of the self-position estimation.

References

1. K.Fujisawa et. al: *A real-time search for autonomous mobile robots,* J.of the Robotics Society of Japan, Vol.17, No.4, pp.503-512,1999
2. K.Doki et.al: *Environment Representation Method Suitable for Action Acquisition of an Autonomous Mobile Robot,* Proc. of International Conference on Computer,Communication and Control Technologies,Vol.5, pp.283-288,2003
3. S.Zilberstein et al.: *Anytime Sensing, Planning and Action: A Practical Model for Robot Control,* the ACM magazine,1996
4. K.Doki et al.: *Self-Position Estimation of Autonomous Mobile Robot Considering Variable Processing Time,* Proc. of The 11th IASTED International Conference on Artificial Intelligence and Soft Computing, 584-053(CD-ROM), 2007

5. T.Taketomi, T.Sato, N.Yokoya: *Fast and robust camera parameter estimation using a feature landmark database with priorities of landmarks*, Technical report of IEICE. PRMU, Vol.107, No.427, pp.281-286, 2008
6. N.Mitsunaga, M.Asada: *Visual Attention Control based on nformation Criterion for a Legged Mobile Robot that Observes the Environment while Walking*, J. of the Robotics Society of Japan, Vol.21, No.7, pp.819-827, 2003
7. Y.Satoh et.al: *Robust Image Registration Using Selective Correlation Coefficient*, Trans. of IEE Japan, Vol. 121-C, No.4, pp.1-8, 2001
8. Nixon, Mark S. , Aquado, Alberto S.: *Feature Extraction and Image Processing(2ND)*, Academic Pr, 2008
9. M.Asada et. al: *Purposive Behavior Acquisition for a Robot by Vision-Based Reinforcement Learning*, Journal of the Robotics Society of Japan, Vol.13, No.1, pp.68-74,1995
10. H.Koyasu et.al: *Recognizing Moving Obstacles for Robot Navigation Using Real-Time Omni-directional Stereo Vision*, J.of Robotics and Mechatronics, Vol.14, No.2, pp.147-156,2002

Author Index

Proceedings of the 5th International Workshop on
Artificial Neural Networks and Intelligent Information
Processing - ANNIIP 2009
ISBN: 978-989-674-002-3
http://www.icinco.org