NEURAL NETWORKS DESIGN USING GA WITH PLEIOTROPY EFFECT

Halina Kwaśnicka, Sebastian Rynkiewicz
Department of Computer Science, Wrocław University of Technology
Wyb. Wyspiańskiego 27, 50-370 Wrocław kwasnicka@ci.pwr.wroc.pl

Abstract
The paper presents results of designing neural networks using evolutionary algorithm with new representation schema. The pleiotropy and polygenic effect is implemented in the scheme of individuals coding. The influence of this effect on evolutionary algorithm efficiency is the main subject of the simulation study. The results obtained using proposed algorithm as a tool of neural network design are compared with outcome of neural network design using classic genetic algorithms.

1. Introduction

Neural networks (**NNs**) are strongly developed fields, started from the earlier forties when McCulloch and Pitts proposed their architecture. The similarities of artificial and biological neural networks lie on possibilities of learning on the base of training set, and generalizing. In the last decades new methods of training have been developed, as well as their mathematical basis. **NNs** are useful in variety of practical applications (Freeman, Skapura 1992).

Genetic algorithms (**GAs**) are also a technique from the widely understood area of artificial intelligence, based on the natural process – biological evolution (Holland 1975, Kwaśnicka 1999). The main differences between conventional optimization methods and genetic algorithm are:

- **GAs** work with coded parameters in the form of chromosomes. Usually chromosomes are bits strings. One chromosome (individual) codes single point on the solutions space.
- **GAs** search the solution working simultaneously with a population of individuals.
- **GAs** do not use derivatives or any other information about optimized function.
- **GAs** use probabilistic rules during search process, exploiting areas with high fitness.

Recently, the *hybrid systems* becomes strongly developed and used to solve many practical tasks (Medsker 1995). Combination of different techniques can give better solution or the solution can be obtained in shorter time. In this paper we focus on the combination of **GAs** and **NNs**. However, various approaches to combination of **GAs** and **NNs** are observed, we focus on **GAs** used to **NNs** design. The main difference between proposed approach and others (Bornholds et al. 1992, Schafer et al. 1992, Kucsu, Tronton 1994) lies in the specific operators of used **GA**, namely – pleiotropy and polygenic effect (so called **GAPP**).

2. Designing neural networks

An artificial neural network is the collection of connected units, called neurons. The connections are weighted and weights are usually real values. **NN** consists of inputs neurons

(they accept input data), output neurons (they produce output data) and hidden neurons – they can be organized in the layers, or connected between themselves in any way.

The process of **NN** design for given task consists of four steps (Miller et al. 1989, Schafer et al. 1992): 1) task definition, 2) selection of **NN** architecture, 3) training, 4) evaluation of trained network. The second step of this process is still rather a matter of art, not the routinized task. A designer of **NNs** must rely on his experience and on the informal heuristics arising from the works of others researchers. Therefore, only a few standard architecture types are developed and applied, e.g., layered feedforward or simple recurrent schemes.

Architecture selection is a key issue in the process of **NNs** design. For layered feedforward **NNs**, literature shows two possible approaches to architecture design: by *network reduction* and by *enlarging* (Ossowski 1994). In the *reduction method* we start from the big (redundant) network, for which we suppose that it can learn the task. Next, the two following steps are iterated: 1) training the neural network satisfactory (up to assumed error), 2) simplifying the network by: (a) removing the connections with relatively small values of weight, (b) removing the 'blind' neurons (e.g., neurons without, or with very small weighted outputs). Process is stopped if farther reduction is impossible or the reduced **NN** cannot satisfactory learn its task. *By enlarging* is an opposite method: we should start from the small **NN**, and develop network that consists of optimal number of neurons. There are numerous procedures enabling such extension (Ossowski 1994). Still we have not commonly accepted good method of **NNs** design, elaboration of direct analytic methods is almost impossible (Harp et al. 1989).

Designing a **NN** is the problem of searching for an optimal architecture. This process can be treated as a searching the surface defined by valuation **NN** performance above the space of all possible **NNs** architectures. The surface is infinitely large, structurally similar **NNs** can have different capabilities, and vice versa: different **NNs** can have comparable capabilities. **GAs** are suitable for searching large, complex, deceptive problem space (Holland 1975). The human brain structure is the result of long natural evolution. Using for designing the structure of **NNs** the method based on analogy to natural evolution seems to be the right direction.

3. Designing NNs by the GAPP – GA with pleiotropy and polygenic effect

Usually, in **GAs** a coding schema between chromosome (consisting of genes) and phenotype is very simple: one gene ↔ one phene (Holland 1975). Such algorithm we will call classic genetic algorithm (**CGA**). The pleiotropy effect means that, in assumed coding schema, single gene codes more than one phene. The polygenic effect occurs if single phene depends on a number of genes (Kwaśnicka 1999). One of the first attempts to use the pleiotropy and polygenic effect is made by L. Altenberg (Altenberg 1994). In this approach, the final fitness function is formed as a sum of a number of components. Sequentially, a new gene is included into a chromosome and components on which it influences are randomly selected. The problem with using this approach to the **NN** topology design is to separate independent phenotypic features of **NN** as components.

In our approach, the genotype of **GAPP** is a vector of numbered genes g_i, i=1,..., N. Each gene is a binary coded real value, therefore a chromosome is a bits string. Mutation and crossover are implemented as in classic genetic algorithm. The phenotype is a vector of numbered M phenes: f_1,..., f_M, phenes are real values. The binary pleiotropy matrix (named PP) is used during decoding process, each element of PP is equal to 0 or 1.

$$[Genotype]_{1xN} \times [PP]_{NxM} = [Phenotype]_{1xM} . \quad (1)$$

Binary PP can be easy modified, as mutation.
In our experiments we assume:

- Designed **NN** is full-connected and feedforward,
- **NN** has neuron called bias,
- A number of input neurons is know (**noIn**),
- A number of output neurons is know (**noOu**),
- Activation function in neurons is **sigmoid**,
- Outputs are real values from a set [0,1],
- A number of hidden neurons is assumed (**noHi**).

The chromosome is a list of weights (weights from bias to hidden neurons, from bias to output neurons, from inputs neurons to hidden neurons, from inputs neurons to output neurons, from hidden neurons to output neurons, from hidden neurons to hidden neurons). Assuming a maximal number of hidden neurons (**noHi**) we decide about dimension *M* of *PP*. A number of weights (**NoWeights**) is equal to:

$$NoWeights = noHi + noOu + noIn \cdot noHi + noIn \cdot noOu + noHi \cdot noOu + \frac{noHi \cdot (noHi - 1)}{2}. \quad (2)$$

NoWeights determines dimension *M* of *PP*. On the base of chromosome, the **NN** is built and tested using the training set. For each example *k* the means square error is calculate:

$$\delta_k = \frac{1}{N}\sum_{i=1}^{N}(ou_i - w_i)^2 , \quad (3)$$

where δ_k is en error for *k*th example, ou_i – value of *i*th output neuron for *k*th example, w_i – desired value of *i*th output neuron for *k*th example.

Fitness (**Fit**) of **NN** coded by a chromosome is calculated as a mean square error for whole training set (*L* is a number of examples in training set):

$$Fit = \frac{1}{L}\sum_{k=1}^{L}(1 - \delta_k)^2 . \quad (4)$$

Our algorithm **GAPP** finds a maximum of **Fit**, what means that it minimizes partial errors δ_k. Fitness value **Fit** is a value from the range [0,1]. Taking into account that desired outputs are {0,1}, we can predefine the fitness using following error (φ_k) for *k*th example:

$$\varphi_k = \sum_{i=1}^{N}\xi(ou_i - w_i), \quad (5)$$

where :

$$\xi(x) = \begin{cases} 0, & if\ |x| < 0.5 \\ 1, & if\ |x| \geq 0{,}5 \end{cases} \quad (6)$$

Modified fitness measure Γ is equal to:

$$\Gamma = \sum_{k=1}^{L}\varphi_k , \quad (7)$$

where : φ_k is an error for *k*th example, *L* – a number of examples in a training set.

$\Gamma = 0$ means that the network is trained and gives proper outputs for all examples.

In our algorithm we include modification of *PP* matrix (Figure 1). Modification of single column of *PP* influences on a single weight of **NN**. Changes in column are accepted only if they give better network. Sometimes only changes in a number of columns can improve designed network, therefore additional operations are proposed. They enable adding and deleting neurons. "Improvement operations" for *PP* matrix are following:

- Adding, deleting, modification of connections from inputs layer to given neuron,
- Adding, deleting, modification of connections from given neuron to output layer,
- Adding, deleting, modification of neuron in hidden layer,

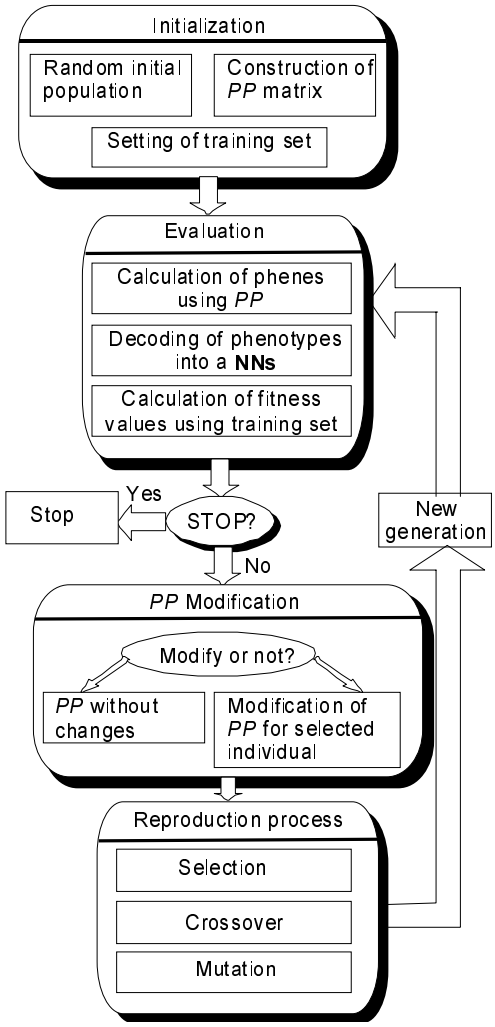

Figure 1. Designing of **NNs** by a **GA** with *PP* matrix

- Adding, deleting, modification of connections between two selected neurons,
- Modification of matrix raw (modification of strength of pleiotropy effect of given gene),
- Modification of randomly selected elements in the *PP* matrix.

Following parameters controls mutation and modification of PP matrix:

- A ratio of population stabilization,
- A ratio of individuals similarity (higher mutations when individuals are similar),
- A ratio of similar results given by all networks from the population.

There are three types of *PP* modifications: 1) During initialization process – a number of individuals are randomly selected and modifications of *PP* is applied. 2) During evolution matrix *PP* is modified in selected generations (Mod_t): $Mod_{t+1} = Mod_t \cdot \alpha + \beta$, 3) *PP* is modified if all phenotypes in the population are similar but genotypes are different.

4. Simulation results

We widely tested proposed approach, experiments were made for the tasks: 1. XOR problem, 2. Autoencoder, 3. Thermometer, 4. Addition of binary numbers (two bits), 5. Digits recognition. The same experiments were made using classic genetic algorithm **CGA** and **GAPP**, so we can compare efficiency of both methods. We focus on the influence of *PP* matrix on the algorithm efficiency. Maximal number of hidden neurons (**noHi**) is always assumed. The real time of achieving solution is comparable for **CGA** and **GAPP**, only in 4th task we observe differences. Usually α=1 and β=5.

XOR problem. Problem requires a **NN** with two inputs, one hidden neuron and one output neuron. We tested population of 20 to 200 individuals (with step 20), repeating each simulation 20 times. 1000 generations is assumed as maximal time of evolution ($\mathbf{T_{max}}$).

For **noHi**=1, and $\mathbf{T_{max}}$=1000, **CGA** has problem with finding solution (only one or two for 20 runs) while **GAPP** gives good results. **CGA** works better (i.e., comparably with **GAPP**) for **noHi**=3. **GAPP** almost in all simulation runs found optimal network before 1000 generations (Figure 2). Big differences are seen in Figure 3, where a means number of generations are presented – for **CGA** only successes runs are taken into account. **GAPP** works better.

Autoencoder problem. Four inputs, always one of them is stated to 1, two hidden neurons, and four output neurons – we require that outputs are equal to inputs. Inputs must be coded by two hidden neurons and decoded to four outputs.

Assumed parameters: **noHi**=2, keep best individual, population size – 20 to 200 (step 20), $\mathbf{T_{max}}$=1000, 20 runs for each experiment.

This problem is simpler for genetic algorithms, both **CGA** and **GAPP** have 100% efficiency (solution was found in all runs). A means number of generations (Figure 4) allows

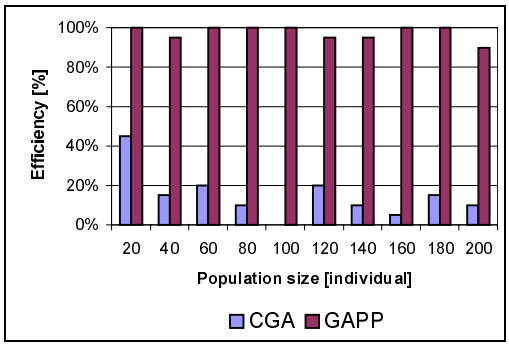

Figure 2. Efficiency of CGA and GAPP in XOR experiment

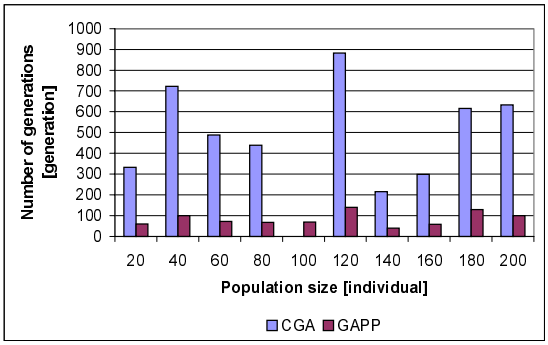

Figure 3. Solution search by CGA and GAPP in XOR problem

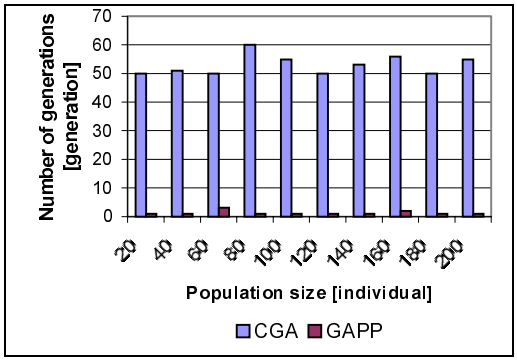

Figure 4. Rate of evolution for Autoencoder problem (CGA and GAPP)

us to say that **GAPP** is quicker than **CGA**. **GAPP** finds solution immediately due to modification of *PP* during initialization process.

Thermometer problem. Network has 3 inputs – binary coded integer number from 0 to 7, and 7 outputs. Decimal value of inputs indicates a number of fired outputs (sequentially placed from the first – it simulates analog thermometer). Algorithms **CGA** and **GAPP** have some problems with this task. The three types of tests were made: **CGA**, **GAPP**, and **GAPP** only with initial modification of *PP* matrix. Assumed parameters: **noHi**=5, keep best individual, population size – 20 to 200 (step 20), $\mathbf{T_{max}}$=2000, 20 runs for each experiment.

The genotype in **GAPP** is shorter than in **CGA**. All tests suggest that the role of dynamic modification of *PP* matrix is significant (Figure 5). Required number of generations varies from 300 to 520 for **GAPP** with dynamic changes of *PP*, from 460 to 890 for **GAPP** with only initial modification of *PP*, and from 520 to 720 for **CGA**.

Addition of binary numbers. Network has four inputs (two bit for two binary numbers) and three outputs, outputs must give binary coded sum of two input numbers. Assumed parameters: **noHi**=15, keep best individual, population size – 20 to 200 (step 20), $\mathbf{T_{max}}$=4000, 20 runs for each experiment. **CGA** strongly depends on the size of evolving populations (Figure 6) but **GAPP** works well for small populations. A means number of generations varies from 700 to 1200 for **GAPP** and from 1200 (population size=200) to 3400 for **CGA**. Real time of computations (population size=100) is 16 sec. for **CGA** and 48 sec. for **GAPP** (for 200 individuals it is respectively: 33 sec. and 63 sec.). For larger populations the real time will be near

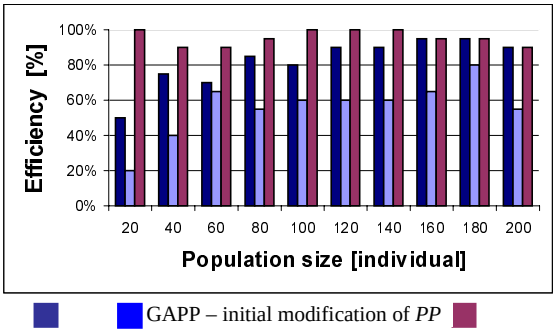

Figure 5. Efficiency of algorithms in *Thermometer problem*

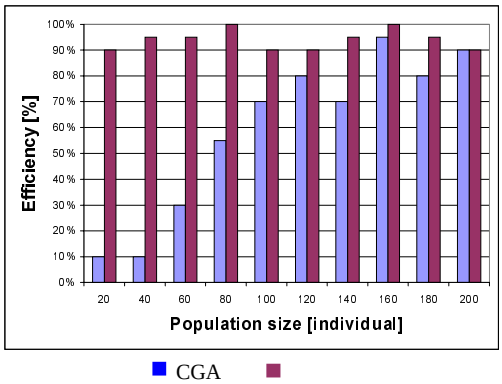

Figure 6. Efficiency of CGA and GAPP – adding binary numbers

the same for both algorithms. **GAPP** works well independently of population size, a means number of generations is about 500, **GAPPP** with population of 20 individuals gives better results than **CGA** with 200 individuals, but real times of computations are similar.

Digits recognition. Obtained results suggest similar conclusions as in *Thermometer problem*. The role of dynamic modification of *PP* matrix is significant.

5. Summary

In the paper we present very shortly results obtained using proposed **GAPP** as a tool of **NNs** design, because a length of the paper is limited. All results are widely described in (Rynkiewicz 2000).

Only a short time of processing single generation of evolution is an advantage of **CGA**. But the efficiency of **CGA**, measured by a percentage of successful runs, is very poor, especially for small populations, **GAPP** is quite independent of population size. **GAPP** consumes more time for single generation but small populations can be evolved with good results.

The frequency of *PP* modification during evolution influence on the tempo and mode of evolution. If modifications are made very often, genetic algorithm has no time to search space of solutions. On the other side, rarely modifications cause that the population size starts play the role. Experiments show that *PP* modifications play significant role in searching process.

The main problem with our method is the way of *PP* modification. Developing the more "intelligent" modification process would be very profitable.

References

ALTENBERG L., *Evolving better representation through selective genome growth*, Proceedings of the IEEE 1994 World Congress on Computational Intelligence, pp. 182-187, 1994.

BORNHOLDT S., GRAUDENZ D., *General Asymmetric Neural Networks and Structure Design by Genetic Algorithms*, Neural Networks, **5**, pp. 327-334, 1992.

EGGENBERGER P., *Creation of Neural Networks Based on Developmental and Evolutionary Principles*, Proceedings of the International Conference on ANNs (ICANN 97), Switzerland, 1997.

FREEMAN J.A., SKAPURA D.M., *Neural Networks, Algorithms, Application and Programming Techniques*, Addison-Wesley Publishing Company, 1992.

GRUAU F., *Neural Network Synthesis using Cellular Encoding and Genetic Algorithm*, PhD Thesis, L'Ecole Superieure De Lyon, January 1994, (http://www.cwi.nl/~gruau/ gruau/ PhD94-01-E.ps.Z)

HARP S.A., SAMAD T., GUHA A., *Towards the genetic synthesis of Neural Networks*, Proceeding of the third International Conference on GAs, Morgan Kaufmann Publishers, Inc., 1989.

HOLLAND J.H., *Adaptation in Natural and Artificial Systems*, The Univ. of Michigan, 1975.

KUSCU I., Ch. TRONTON Ch., *Deisgn of Artificial Neural Networks using Genetic Algorithms: review and prospect*, Cognitive and Computing Sciences, 1994.

KWAŚNICKA H., *Obliczenia ewolucyjne w sztucznej inteligencji* (*Evolutionary Computations in Artificial Intelligence*), Oficyna Wydawnicza PWr., Wrocław, 1999.

MEDSKER L. R., *Hybrid Intelligent Systems*, Kluver Academic Publishers, Boston, 1995.

MILLER G.F., TODD P.M., HEDGE S.U., *Designing Neural Networks using Genetic Algorithms*, Proceeding of the third International Conference on GA, Morgan Kaufmann Publishers, Inc., 1989.

OSSOWSKI S., *Sieci Neuronowe* (*Neural Networks*), Oficyna Wydawnicza PW, Warszawa, 1994.

RYNKIEWICZ S., *Zaawansowane operatory genetyczne w projektowaniu sieci neuronowych* (*Advanced genetic operators in NN designing*), Master Thesis, PWr, Wrocław, 2000.

SCHAFER J.D., WHITLEY D., ESHELMAN L.J. *Combinations of Genetic Algorithms and Neural Networks: A Survey of the State*, Baltimore, Maryland 1992.