CROATICA CHEMICA ACTA CCACAA **68** (1) 29–38 (1995)

ISSN 0011-1643
CCA–2209

Conference Paper

Adaptation of Neural Networks Using Genetic Algorithms

Tin Ilakovac

Ruđer Bošković Institute,
POB 1016, HR-41001 Zagreb, Croatia

Received July 25, 1994; revised October 10, 1994; accepted December 01, 1994

This paper proposes an approach to the problem of adaptation of neural networks (NN) to arbitrary tasks. It is well known that the functional properties of a NN depend on its construction: on topological structure, learning and activation methods, and signal output. A definition language is developed for describing various constructions of NNs in the shape of strings. This paper uses a model of a neuron which has a receptive field and adaptable learning, activation and signaling, while the NN model consists of interconnected layers allowing feedforward, feedback and lateral connections with a single input and ouput layer. Adaptation of NNs is done with a genetic algorithm (GA) using crossover, mutation, and proportional selection operators on a population of strings that represent NNs. These strings (and their respective NNs) are evolved until they find solutions to given tasks which are defined as objective functions. The paper proposes a solution to »deception«, an important problem concerning GA's convergence: a strict hierarchy in the description of NNs based on ordered expression which decreases the probability of dual representations. This approach can develop autodidactive NNs.

INTRODUCTION

Applications of neural networks (NN) can be loosely divided into two groups: optimization of complex problems in multidimensional spaces and robust associative or classifying signal processing. The problem that is always encountered is how to choose an appropriate construction of a NN which will be able to solve the required task. The construction of an NN is given by its topology, activation and learning method, as well as signal output. Wrongly chosen NN construction cannot solve the given task, or solve them only partially.

Automatic adaptation of NN construction to a given task enables the adapted NN to completely solve the task, but also to minimize the size of NN, thus shortening the time necessary for learning and, finally, the time of its response to stimuli. In this work, genetic algorithms (GA) are used to find NN constructions which are capable of solving arbitrary problems.

GA can converge very fast to the complex space regions which contain good solutions of the task due to their implicit parallelism, which is based on simultaneous estimation of a vast number of subsolutions and the ability to combine these solutions.

This paper develops a specific language that describes the model of an individual neuron and the topology of an arbitrary NN. Here, it will be called *description language*. It is used to generate strings that define the constructions of arbitrary NNs (the strings represent »genotypes« of NNs). The strings are translated to realizations of NNs using an *interpretation function* (the realization presents the »phenotype« of the NN).

The second chapter describes genetic algorithm operators and their characteristics, and explains the reasons for GA convergence. The third chapter gives a general description of NNs, followed by the model used in this work. The fourth chapter delves into the problem of deception that plagues interpretation functions, and proposes the type of language that describes arbitrary NNs while decreasing the occurrence of this problem. The fifth chapter concludes with descriptional abilities of the proposed language and the expected results of this combined optimization approach.

GENETIC ALGORITHMS

Genetic algorithms are based on the principles of biological evolution which were found by Charles Darwin and defined as *»the survival of the fittest«*.

The process of evolution works on a group of individuals that make a *population*. Members of the population compete in accomodating to the environment. Competition is the basis of the *natural selection*: nonaccommodated members are rejected, and accommodated members are reproduced, creating offspring that take the place of rejected members. *Reproduction* is the mixing of characteristics of pairs of good members to produce new members that possess some characteristics of one parent, and other characteristics of the other parent. The second way of changing the population is called *mutation*. Mutation randomly chooses a member of the population and randomly changes some of its characteristics.[1]

The purpose of GA is to change the structure of some modifiable program by changing the genetic code which represents it. The main problem is to find a convenient genetic code that represents the modifiable program in a satisfactory way.

The members of the population are represented by genetic codes in the form of strings. Researchers most often use strings consisting of binary values, but sometimes the strings have different interpretations.[3] Each member of the population is represented by one string, and the group of all strings makes up the population.[2]

The GA procedure has the following steps:

1. The function which defines the environment imposed on the strings has to be determined and the results which this function gives for each string are called *fitness*.
2. An initial population of strings has to be generated.
3. The strings are evaluated by the fitness function, and the *selection* is imposed on the population based on the fitness factor of each string: bad strings are rejected, and the best ones are chosen for reproduction with the probability proportional to their fitness.

4. Reproduction is preformed by crossover: a pair of strings are aligned along each other, a random point is selected, and the tails of strings after this point are interchanged.

5. The offspring produced by reproduction replace the rejected strings.

6. Mutations are occasionally performed to prevent the population from becoming uniform as a result of approaching the region of an optimum.

7. The algorithm continues at step 3 until a satisfactory string is found.

The properties and the convergence of GA have recently attracted a lot of attention. Implicit parallelism is proposed as the main reason for a fast convergence.[4] It has been found that this parallelism and the capability of combining strings that contain partial solutions result in two properties:

- The frequency of sampling a particular subspace is proportional to the probability of a good solution in that subspace
- The number of strings in a particular subspace grows at a rate proportional to the mean value of fitness in the subspace

The reason for such properties is that a string belongs to all the regions of the problem space where its substrings belong. Big regions of the problem space are represented by strings with a small proportion of defined bits and a big proportion of undefined bits. Therefore, these regions are sampled by a vast majority of the population members. An example of such a string is 10 * * * * * * with two leading bits defined, and the rest undefined.

Crossover selects some random point, which can be the one contained in a substring belonging to a good region of the problem space. In that case, the produced offspring will leave the good region and enter another, probably worse region, thus spoiling the effect of implicit parallelism on the convergence. The probability of a substring leaving some region depends on the distance between the bits of this substring. For example, a string 10 * * * * has a chance 1/5 to leave region 10 as there are 5 points where crossover can take place. Therefore, the substrings that are consecutive and short (which are called *building blocks*) will tend to stay intact through the crossover operation. These substrings will be inherited by the offspring with the probability proportional to the fitness value of the string that contains them. As there is a vast number of building blocks in a string, the implicit parallelism is maintained.

An important point is that GA is not an optimization method that converges to some optimum by »looking« at the gradient in its momentary local neighbourhood, but rather by sampling the problem space in a sophisticated way. Thereofre, GA converges fast to some of the good regions of the problem space, but finds slowly the exact point of optimum in that region. Consequently, optimum finding methods like the gradient method should be used for the final convergence.[5]

Elitist selection,[6] where the best member is always retained, assures the convergence of GA to the global optimum while a canonical GA, where this property does not hold, has no convergence at all but has a tendency to repeatedly finding the global optimum.[7]

THE MODEL OF THE NEURON AND THE NETWORK

NN represent a crude simulation of the neural systems in living beings. One NN consists of a network of interconnected neurons having multiple inputs (dendrites) and a single output (axon). Through inputs the neuron receives signals from neighbouring neurons, multiplies signals with a specific weighting factor, and most often sums up the products. The sum represents the activity of the neuron, and the activity produces the signal output:[8]

$$a_i = \sum_j w_{ij} s_j$$

$$s_i = S(a_i)$$

where a_i is the activation of the i-th neuron, W_{ij} is the weighting factor for the signal emanating from neuron j and sinking into neuron i, and s_i is the result of the signal function S working on the activation of neuron i. It is interesting to note that, with respect to signal processing, the activation function is a scalar product of input signals and their respective weights, and the scalar product is the main mathematical tool for digital signal processing.[9] Signal function is either linear or limiting between the strongest and the weakest allowed values. If it is a limiter, it can be of a threshold type, or a sigmoid type which has a continuous first derivative (important for some types of learning). Sigmoid is defined as $F(a_i) = 1/(1 + \exp(-a_i))$.

The behaviour of NN is a consequence of the collective behaviour of single neurons, but differs greatly from the single neuron behaviour. NN consisting of three layers (input, hidden, and output) organizes its middle (hidden) layer to represent the input data in a self-specific way. Removing one neuron from this layer will not destroy the response of the NN to the same input data, but will gracefully reduce the quality of the output.[10] This is a consequence of the distributed representation of the input data in the hidden layer.

Most researchers build their NN to consist of a few layers,[11,12] and the NN model that is used in this paper enables the same. The layers are 2D planes interconnected into a network. There is one input and one output layer. All layers can receive inputs from a few layers, except the input layer which may not be connected to any layer (including itself). The layers may be connected in a cyclic way, and may signal to themselves (this is called a lateral connection). The activation function is not explicitly given but through the topology of the whole construction. The operations between the layers are $\oplus$ and $\otimes$, where $\oplus$ is standard commutative addition of two or more values, and $\otimes$ is a noncommutative blocking operator where the first operand, if active, blocks all the following operands, thus preventing them from influencing the activity of the receiving neuron.

To define the construction of interconnected layers, it is necessary to define the size of each layer with two numbers and how the layers are interconnected.

Figure 1 shows an example of interconnected layers. In this example, layer **F** is excited by layer **C**, but this excitation can be blocked by layer **D**. Layer **F** excites itself laterally, and excites the output layer **B**.

Each layer consists of neurons which are clones of each other: they have the same activation function, the same weights, the same signal function, and the same

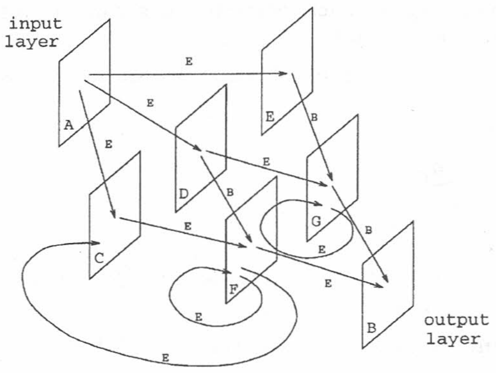

Figure 1. Example of an NN model with feedforward, feedback and lateral connections. Connections are exciting or blocking. There must be exactly one input and one output layer.

learning function. The layers have different and specific functions – a property of diversification that has been found in biological systems.[14] One neuron does not necessarily see the whole neighbouring layer, but only a region of it which is called *receptive field*. This means that a neuron is connected to those neurons in the neighbouring layer which are in its receptive field. This resembles the way in which the real neurons in biological system are interconnected.[13] The neurons belonging to one layer have different positions of the receptive fields – they are shifted one to another. All neurons belonging to a layer see all the neurons of its neighbouring layer, but a single neuron sees only a part of them, as shown in Figure 2.

A neuron sees its receptive field through a mask of weights. All neurons in a layer have the same mask of weights. The influence of a neighbouring layer on one neuron is the scalar product of the signals from the receptive field and the receptive weights in the mask of weights.

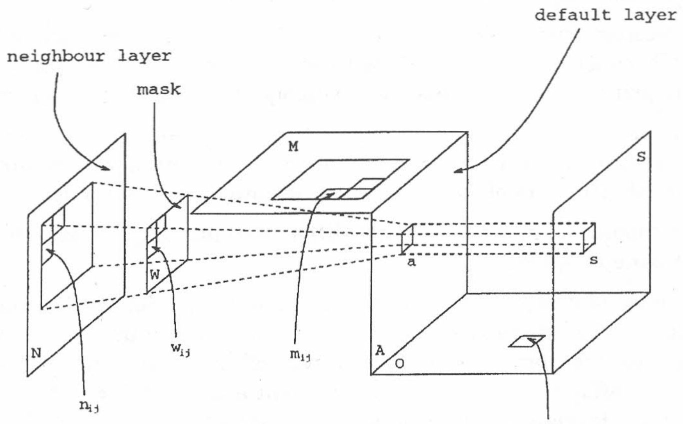

Figure 2. Model of the neuron as a part of a layer (n receptive field, w mask of weights, a activation, s signal output, m previous activation, o previous signal output).

The signal function maps the activation to the signal output by one of the functions: linear, threshold, or sigmoid.

A single neuron has a very limited scope of information which it can use for learning:

- its own activation (a_i)
- its own signal output (s_i)
- the weight mask (w_{ij})
- the signal outputs of neurons in the receptive field (n_{ij})
- its own signal output in the last iteration (o_i)
- the signal outputs of neurons in the receptive fields in the last iteration (m_{ij})

This model allows any kind of learning that is a linear function of these variables:

$$w_{ij} = W(w_{ij}, n_{ij}, a_i, s_i, m_{ij}, o_i) .$$

To define the whole complex, it is necessary to add the following definitions for neurons of each layer: the size of the weighting mask (receptive field), the learning function for the mask, and finally the signal function.

THE DESCRIPTION LANGUAGE

Let us now turn to the language that is used to define arbitrary NN constructions. The main problem concerning the definition of such a language is called *deception*.[16] Deception is the possibility that the fitness of a string is not correlated to the fitness of its component substrings. The results of deception can be that two strings with high fitness produce an offspring string that, when translated to a NN construction by an *interpretation function*, gets a lower fitness value than its parents. Three reasons may cause this peculiar problem:[15]

1. The interpretation function is »many to one«, allowing a number of strings to produce the same NN construction. Two NN with identical construction and weights may produce offspring containing repeated components instead of complement parts.
2. The parent topologies are identical, but the weights are not: the distributed representations may differ between NN which have similar behaviours, and offspring may inherit parts of distributed representations that do not fit each other.
3. The parent topologies and the weights differ: parents may be genetically too far to produce sane offspring.

The first cause of deception comes as a result of descriptions of NN constructions that do not have a well-defined sequence, in which case the interpretation function has the »many to one« property: crossover will often interchange parts of parent strings producing offspring which do not represent a whole NN construction. Therefore, the language developed here describes the characteristics of a NN construction in well-defined places, sequentially and hierarchically. A string is divided into functionally separated substrings. These substrings describe NN in the following order: topology, activation method, learning method, signal function, layer sizes, and finally constant values. Hierarchically higher substrings call for the definition of their components in the hierarchically lower substrings.

The description of NN construction does not include the weights between neurons, so the second cause of deception does not apply to our GA. The development of weighted connections between the neurons of a member is done by the NN learning method of this member during its life, but the weights are lost in the reproduction process. This accentuates the structure of NN with its specific selforganizing properties.

The third cause of deception cannot be easily avoided, and perhaps should not be avoided at all. Complex problems can always be solved in various ways, and groups of member strings that cannot have good offspring represent such different solutions. A separate »breeding« of those groups can relax the third cause of deception.

Each component of the NN construction has to be declared and its value has to be defined. A component belongs to a certain type with a specific name. It has a code number that discriminates it from the other components of the same type. The components are declared in the following way:

$ [code] – layer
^ [code] – size (2D)
~ [code] – type of mask
# [code] – learning method
? [code] – signal function
! [code] – constant

The NN construction is defined in blocks, each block being a substring of the whole string. A block has a start and a stop code, and all components of the same type are sequentially declared and defined between the start and stop codes of a block. The number of components in a block is not limited (apart from computer imposed limitations). Components are divided by a dot and surrounded by parentheses. Generally, a block looks like:

$$|\Diamond|\,(\ ).(\ ).(\ ).\ etc.\ |E|$$

where ◊ denotes one of the following: T,S,M,V,L,K,E, and the blank spaces between the parentheses denote components of the same type.

Each block defines a group of components of the following types:

- $|T|$topology
- $|S|$layers
- $|M|$masks
- $|V|$layer sizes
- $|L|$learning methods
- $|K|$constants
- $|E|$end

Now, let us give an informal specification of the language. The blocks describe the components in the following way:

$|T|$ ($\$code\ \langle\ \tilde{}code\rangle = \$code \odot \$code \odot \cdots \odot \$code$). *etc* $|E|$
$|S|$ ($\$code = \hat{}code$). *etc* $|E|$
$|M|$ ($\sim code = \hat{}code, \#code$). *etc* $|E|$
$|V|$ ($\hat{}code = !code, !code$). *etc* $|E|$
$|L|$ ($\#code$ = [*arithmetic expression containing*: $a, w, n, m, o, s, +, -, *, \div$, (,)]). *etc.*$|E|$
$|K|$ ($!code$ = *value, change indicator*). *etc.* $|E|$

($\odot$ denotes either excitation $\oplus$ or blocking $\otimes$)

A simple example will give a better insight into the usage of the description language. The example presents a two layered network consisting of an input layer and an output layer where the output layer has a receptive field through which it »sees« the input layer. The learning method is the simplest one: Hebbian learning, where the change of a weight is proportional to the activation of the neuron times the weight itself. Learning changes weights of the masks confronting receptive fields. The size of the input layer is 200 × 200, the output layer is 100 × 100, the receptive field is 9 × 9, and the rate of learning is 3. Finally, the change factor of the constants is set to 4, which denotes no change.

$|T|$ ($\$A = 0$). ($\$B = \$A\tilde{}w_1$). $|E|$
$|S|$ ($\$A = \hat{}s_0$). ($\$B = \hat{}s_1$). $|E|$
$|M|$ ($\tilde{}w_1 = \hat{}s_2, \#l_0$). $|E|$
$|V|$ ($\hat{}s_0 = !k_0, !k_1$). ($\hat{}s_1 = !k_2, !k_3$). ($\hat{}s_2 = !k_4, !k_5$).
$|L|$ ($\#l_0 = !k_6 \times w \times a$). $|E|$
$|K|$ ($!k_0 = 200, 4$). ($!k_1 = 200, 4$). ($!k_2 = 100, 4$). ($!k_3 = 100, 4$). ($!k_4 = 9, 4$).
($!k_5 = 9, 4$). ($!k_6 = 3, 4$).

The last question to be tackled is how to improve the rate of evolution of a population where the members have been described by the decription language rules. If the whole population has a very low fitness value, then the information contained in strings representing the members can be severely changed without much damage. In such a case, the rate of evolution should be increased. The factors that are considered here and which are capable if changing this rate are the selected points of crossover, the sliding of the constants, the amount of genetic information, and the mechanism for gene repair.

The crossover points can be selected completely at random, allowing for the greatest variability of the offspring but also maximizing the probability that the offspring cannot code NN constructions. To increase the probability of getting derivable offspring, it is necessary to restrict the crossover points to certain allowable locations. The most secure locations are the points between strings. A less stringent restriction allows the points between expressions contained in a block. Finally, pairs of equal operations signs like »=« and »$\odot$« could be allowed for crossover points.

The constants used by a NN define its construction and behaviour. If constants change by a small amount form generation to generation (a process that we call »*sliding*«), then the NN's characteristics that depend on constants will change more rapidly than only by crossover and mutations. The sliding has to be stopped when NN gets higher fitness results.

A string contains a certain amount of »genes«, or information how to construct a NN. Obviously, some »genes« have to be used for the construction, but some do not have to be used. Those which are not used represent a pool of genes and may enter the construction of NN in some later generation. New genes may be added to the pool by a mechanism called *gene generator*. Gene generator produces genes by means of a random process and they are inserted into strings. The pool of genes improves the variability of a NN construction.

A *gene repair* mechanism can be run after each genetic operation. Genetic operators alter the information carried by strings in an unpredictable way. The purpose of genetic repair mechanism is to detect »insane« genes and to translate each one to its nearest meaningful gene if that is possible, otherwise to reject that gene or the complete string containing it. The random results of the *gene generator* are good candidates for a gene repair mechanism as they are often prone to mending or discarding. Also, the sliding constants can be conceived as deviant genes, and should be mended to become nonsliding as soon as their NNs get higher fitness results. Therefore, the gene repair mechanism has to be activated on them.

CONCLUSION

This paper develops a language called *description language* for defining the contruction of a wide variety of models of neural networks. The strings that are products of description language are well adapted to modification with genetic algorithms. The genetic algorithms that are used are based on previous works in the field of evolutional programming and improved by some additional mechanisms.

Description language has been developed to be able to define some already known and well studied NN models which are based on unsupervised learning. This language enables the construction of multilayer NN with feedforward and feedback connections between the layers, and lateral connections inside a layer. A single neuron has a particular receptive field for every neighbouring layer. All neurons in one layer are the same, excluding the positions of their receptive fields. The method of activation and learning as well as the signal function are selected by GA.

The point of this work is twofold: the usage of a string-generating description language for definition of NN constructions, and a convergence method that consists of a genetic algorithm and a neural network learning. After finding an appropriate construction of NN (by locating the subspace where the good solutions are situated), it is up to the NN to converge to the local optimum of that subspace. This method divides the whole convergence into two parts: approaching the region of the solution with GA, and finding the exact position of the solution with NN.

In principle, this approach can give answers to the following questions:

1. Are the known NN models reachable by using GA?
2. Is it possible to improve these models?
3. How well can we imitate the biological visual system?
4. Can we use developed models for 2D signal processing?

REFERENCES

1. E. Mayr, *The Growth of Biological Thought: Diversity, Evolution and Inheritance*, Cambridge, MA: Belknap press, 1988.
2. D. E. Goldbeerg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley Inc., 1989.
3. J. Antonisse, »A new interpretation of schema notation that overturns the binary encoding constraint,« *Proc. of the Third International Conf. on Genetic Algorithms*, J. D. Schaffer (Ed.), San Mateo, CA. Morgan Kaufmann Publishers, 1989. pp. 86–91.
4. J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press, 1975.
5. H. Muhlenbein, »Parallel genetic algorithms, population genetics and combinatorial optimization,.« in *Proc. of the Third Int. Conf. on Genetic Algorithms*, J. D. Schaffer (Ed.), San Mateo, CA: Morgan Kauffman, 1989. pp.416–421.
6. J. J. Grefenstette, »Optimization of control parameters for genetic algorithms,« *IEEE Trans. Sys. Man and Cybern.* 1986, Vol. 16, No.1, pp. 122–138.
7. G. Rudolph, Convergence analysis of canonical for genetic algorithm,» *IEEE Trans. Neural Networks*. 1994, Vol. 5, pp. 96–101.
8. B. Kosko, *Neural Networks and Fuzzy Systems*, Prentice-Hall, 1992.
9. A. V. Oppenheim and R. V. Schafer, *Discrete-Time Signal Processing,«* Prentice-Hall, 1989.
10. G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, »Distributed representation,« in *Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Vol. 1: Foundation*, MA: MIT Press, 1986.
11. R. Linsker, »From basic network principles to neural architecture: Emergence of spatial-opponent cells,« *Proc. Natl. Acad. Sci. USA*, 1986, Vol. 83, pp.7508–7512.
12. K. Fukushima, »A neural network for visual pattern recognition,« *IEEE Computer*, 1988. pp. 65–74.
13. D. B. Hubbel, and T. N. Wiesel, »Brain mechanisms of vision,« *Sci. American* 1979, vol. 241, pp. 150–162.
14. S. Zeki and S. Ship, »The functional logic of cortical connections, *Nature*, vol. 335, 1988. No. 6188, pp. 311–317.
15. P. J. Ageline, G. M. Saunders, and J. B. Pollack, «An evolutionary algorithm that constructs recurrent neural networks,« *IEEE Trans. Neural Networks,* 1994. Vol. 5, pp. 54–65.
16. D. E. Goldberg, »Genetic algorithms and Walsh functions: Part 2, Deception and its analysis«, *Complex Systems*, 1989. Vol. 3, pp. 153–171.

SAŽETAK

Prilagodba neuronskih mreža s pomoću genetskih algoritama

Tin Ilakovac

Opisan je pristup adaptaciji neuronskih mreža proizvoljnim zadacima. Dobro je poznato da radne značajke neuronskih mreža ovise o njihovoj konstrukciji: topološkoj strukturi, metodi učenja i aktivacije i izlaznoj signalnoj funkciji. Razvijen je opisni jezik kojim se različite konstrukcije neuronskih mreža opisuju u obliku slovčanog niza (string). Uporabljeni model neurona ima receptivno polje, adaptivno učenje, aktivaciju i signalnu funkciju, a cjelokupna neuronska mreža sastoji se od slojeva koji dopuštaju unaprijedne, povratne i postranične veze s jednim ulaznim i jednim izlaznim slojem. Adaptacija neuronskih mreža izvodi se genetskim algoritmom upotrebljavajući križanje (crossover) , točkastu mutaciju i selekciju kao operatore nad populacijom nizova koji reprezentiraju neuronske mreže. Ti se nizovi evoluiraju prema rješenju zadanog problema. Predlaže se rješenje takozvanog decepcijskog (deception) problema koji uzrokuje nekonvergentnost genetskog algoritma: uvodi se hijerarhija opisa neuronskih mreža s uređajem izraza kojim se smanjuje vjerojatnost udvostručenih reprezentacija. Tim pristupom moguće je razvijati neuronske mreže bez učitelja.