# An Architecture-Altering and Training Methodology for Neural Logic Networks: Application in the banking sector

Athanasios Tsakonas[1] and Georgios Dounias[2]

[1]*Department of Production and Management Engineering, Demokritus University of Thrace,12 Vas.Sofias St.,Xanthi,Greece*

[2]*Department of Financial and Management Engineering, University of the Aegean, 31 Fostini St., Chios, Greece*

**Abstract.** Artificial neural networks have been universally acknowledged for their ability on constructing forecasting and classifying systems. Among their desirable features, it has always been the interpretation of their structure, aiming to provide further knowledge for the domain experts. A number of methodologies have been developed for this reason. One such paradigm is the neural logic networks concept. Neural logic networks have been especially designed in order to enable the interpretation of their structure into a number of simple logical rules and they can be seen as a network representation of a logical rule base. Although powerful by their definition in this context, neural logic networks have performed poorly when used in approaches that required training from data. Standard training methods, such as the back-propagation, require the network's synapse weight altering, which destroys the network's interpretability. The methodology in this paper overcomes these problems and proposes an architecture-altering technique, which enables the production of highly antagonistic solutions while preserving any weight-related information. The implementation involves genetic programming using a grammar-guided training approach, in order to provide arbitrarily large and connected neural logic networks. The methodology is tested in a problem from the banking sector with encouraging results.

## 1 Introduction

Being a methodology that aims to the integration between artificial neural networks and AI's rule-based systems [10], the neural logic networks paradigm [16] has been developed and applied nowadays, in a number of domains [9]. Neural logic networks have been proved successful when used in the AI framework, by providing network representations for nearly every logical rule base. The potential of using neural logic networks within the computational intelligence framework has been proposed since their first presentation [16]. Various training methods have been developed since then. In [16], a training methodology related to back-propagation was proposed. Later, the Supervised Clustering and Matching (SCM) algorithm [15] was introduced.

Tsakonas A. and Dounias G. (2005).
An Architecture-Altering and Training Methodology for Neural Logic Networks: Application in the banking sector.
In *Proceedings of the 1st International Workshop on Artificial Neural Networks and Intelligent Information Processing*, pages 82-93
DOI: 10.5220/0001192100820093

However, when used within the CI framework, the extracted networks almost always resulted into non-interpretable forms, eliminating this way their potential advantage over common neural network models. These early training methodologies used refinements of the edge weights, which resulted in non-interpretable solutions. This drawback led the research to alternative solving methodologies such as the genetic programming [1]. However, their system provided a model that was capable of producing only a limited set of neural logic network representations [1], which were actually neural logic binary trees. The latter problem is solved in the methodology of this paper, by providing indirect representation of neural logic network architectures within the genetic programming framework. The indirect representation is based on the cellular encoding [2], and is applied into the genetic programming [6] using a context-free grammar system. The problem where the system is applied comes from the banking sector and involves the loan management for enterprises.

The paper is organized as follows. In section 2 we introduce the theoretical background of the neural logic networks and the genetic programming framework. The paragraph covers also a review in grammar-guided methodologies for genetic programming and the cellular encoding advances for connectionist systems representation within genetic programming individuals. Section 3 contains the design and the implementation description of the proposed system. In section 4 we include the description of the problem domain, our system configuration and the obtained results together with a discussion. Finally, section 5 contains our conclusion regarding this work and proposes future directions for this domain.

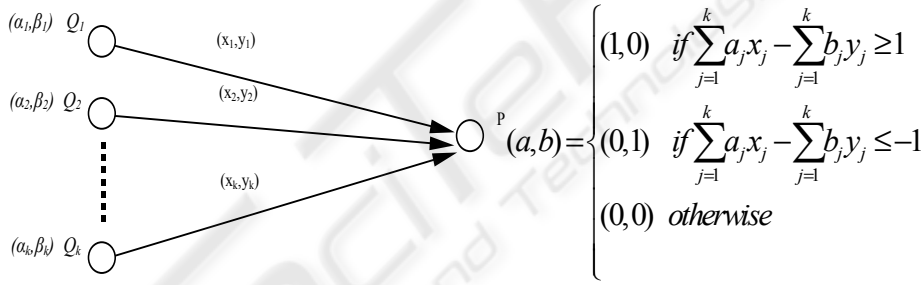



**Fig. 1.** Neural logic network and its output.

## 2 Background

### 2.1 Neural Logic Networks

A neural logic network is defined as a finite directed graph. It consists of a set of input nodes and an output node. In its 3-valued form, the possible value for a node can be one of three ordered pair activation values (1,0) for true, (0,1) for false and (0,0) for don't know. Every synapse (edge) is also assigned a an ordered pair weight (x,y) where x and y are real numbers. An neural logic network and its output value (a,b) of node P is shown in *Figure 1*. It is possible to map any rule of conventional knowledge into a neural logic network by using different sets of weights, which enable the representation of different logical operations. In *Figure 2*, some examples

of these logical operators and their implementation into neural logic networks are presented. The neural logic networks have not applied in many domains within the CI framework, although they are powerful by their definition. The main reason can be located in the fact that for the known training methodologies [16], [15], the adjustment of the synapse weights eliminates the ability to interpret the extracted solution into a number of logical rules, thus depriving these networks from their valuable feature. In *Figure 3(a)* it is shown an easily interpretable neural logic network, while in *Figure 3(b)*, a network with adjusted synapse weights fails to be interpreted. Some steps for the preservation of the interpretability have been performed by [1], which however proposed a system that lacks the ability to express arbitrarily large and connected neural logic networks. In *Figure 4(a),* a network that is produced by the methodology of by [1] is shown. In *Figure 4(b),* it is shown a neural logic network, which performs the fundamental logical operation of XOR, and it cannot be represented using the direct encoding of [1]. In order to make the networks able to handle any real value (i.e. not only 3-valued pairs), the fuzzy extension for neural logic networks has been proposed [16], a design that is also adapted here for the needs of the problem examined in this paper.

### 2.2 Genetic Programming

Genetic programming [6] is an evolutionary computation approach, which in its canonical form enables the automatic generation of mathematical expressions or programs. Genetic programming retains a significant position among successful evolutionary computation approaches due to its valuable characteristics, such as the

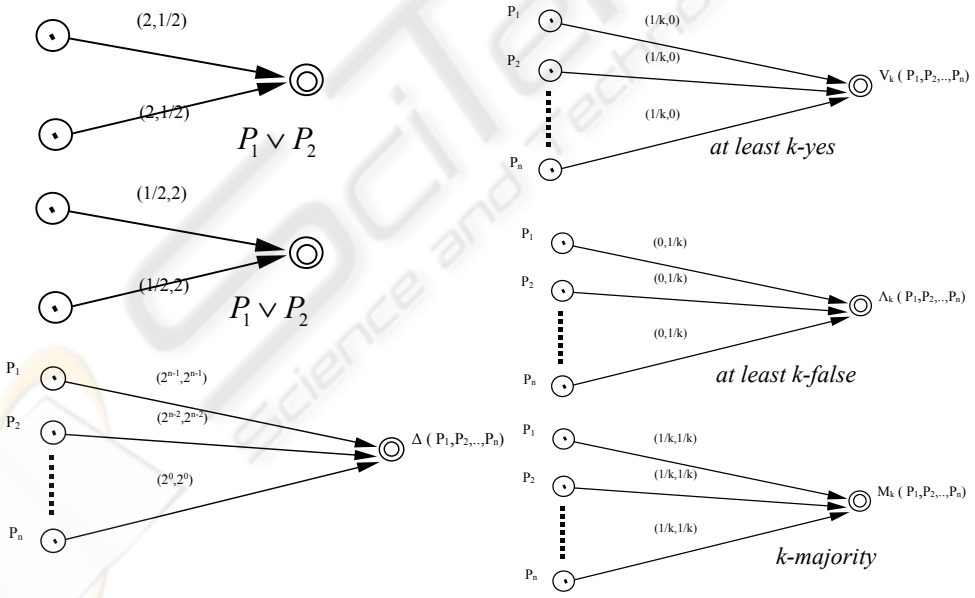



**Fig. 2.** Example logical operations in neural logic networks

flexible variable-length solution representation and the absence of population convergence tendency. In most implementations, a population of candidate solutions is maintained, and after a generation is accomplished, the population increases its fitness for a given problem. The genetic operators that are mostly used in these algorithms are reproduction, recombination (crossover) and mutation. The first operation copies an individual without affecting it, the recombination exchanges genetic material between two individuals and mutation alters a part of a randomly selected genetic material. A genetic programming training cycle usually includes the following steps:

1. Initialise a population of individuals at random.
2. Evaluate randomly an individual and compare its fitness to other (this fitness determines how closely is an individual to the desired goal).
3. Modify an individual with a relatively high fitness using a genetic operator.
4. Repeat steps 2-3 until a termination criterion is met.

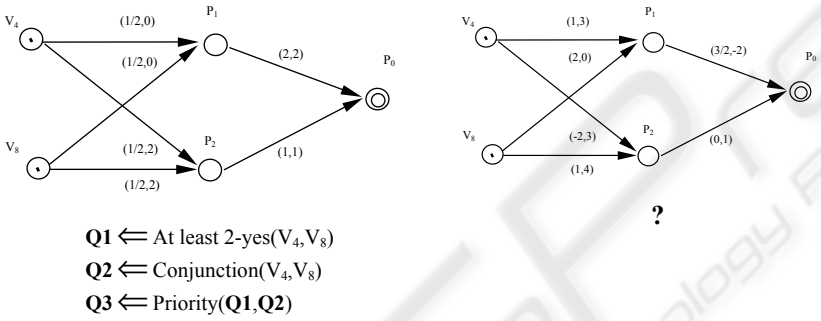



**Fig. 3.** (a) Interpretation of a neural logic network into logical rules. (b) a network with refined edge weights that cannot be interpreted.

Common termination criterions are the accomplishment of a number of generations, the achievement of a desired classification error, etc.

As previously stated, the genetic programming has been proved an advance over traditional genetic algorithms due to its ability to construct functional trees of variable length. This property enables the search for very complex solutions that are usually in the form of a mathematical formula - an approach that is commonly known as

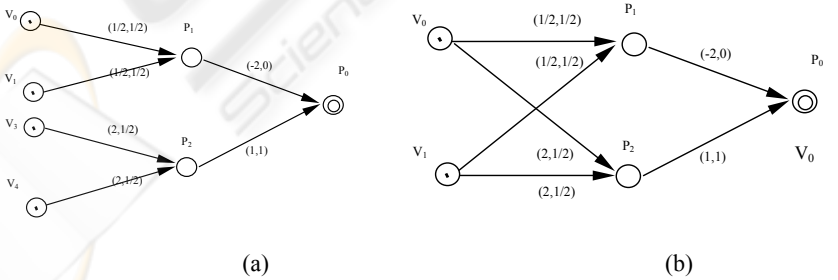
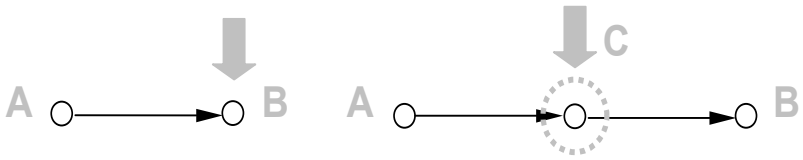



**Fig. 4.** (a) Tree-like neural logic networks as generated in [1]. (b) a neural logic network that performs the XOR operation needs the general structure of a finite graph and cannot be described directly by the approach of [1].

symbolic regression. Although later paradigms extended this concept to calculate any boolean or programming expression, the task of implementing complex intelligent structures into genetic programming functional sets in not rather straightforward. The function set that composes an intelligent system retains a specific hierarchy that must be traced in the genetic programming tree permissible structures. This writing offers two advantages. First, the search process avoids candidate solutions that are meaningless or, at least, obscure. Second, the search space is reduced significantly among only valid solutions. Therefore, when the syntax form of the desired solution is well defined, it is useful to restrain the genetic programming from searching solutions with different syntax forms [3], [8]. One of the advantageous methods to implement such restrictions is to apply syntax constraints to genetic programming trees, usually with the help of a context-free grammar [2], [5], [13]. The execution of massively parallel processing intelligent systems such as the neural logic networks within the genetic programming framework is not considered a straightforward task. In order to explore variable sized solutions, it is required the application of an indirect encoding system. The most common one is the *cellular encoding* [2], in which a *genotype* - a point in the search space- can be realised as a descriptive *phenotype* - a point in the solution space. More specifically, within such a function set, there are elementary functions that modify the system architecture together with functions that calculate tuning variables. Related implementations include encoding for feedforward and Kohonen neural networks [2], [4], [17] and fuzzy Petri-nets [18]. A similar technology, called *edge encoding*, developed by [7] is also today used with human competitive results in a wide area of applications.

## 3 Design and Implementation

As mentioned in the previous paragraph, the characteristic feature of neural logic networks should be the ability to interpret any network architecture into a set of logical rules. For this reason, the *cellular encoding* is used in our model to represent the candidate solutions into genetic programming trees. One cellular encoding scheme includes (I) functions for architecture altering and (II) functions for parameter tuning. The functions for parameter tuning have common properties with the usual genetic programming functions, which operate as procedures or program elements. The functions that are used for architecture altering however are not used in standard genetic programming systems. They comprise a function set that alters an embryonic neural logic network, by entering nodes sequentially or in parallel onto an initial (elementary) neural logic network, in order to form the final / desirable architecture. Hence, among the architecture altering functions we may discriminate between (I-a) functions that enter a node serially, and (I-b) functions that enter a node in parallel. The problem that has arisen during the prime implementations of cellular encoding concerns the grammar description, which enabled the existence of networks without inputs [2], a situation that can easily lead into premature population convergence.




(a) (b)

**Fig. 5.** Application of the function S1 on the B node.


(a) (b)

**Fig. 6.** Application of the function P1 on the B node.

Thus, in order to be able for a system to include at least one input, we incorporated different functions for the architecture altering on the system inputs, than those used for the architecture altering on internal nodes. Hence, we may further divide the architecture altering functions into two additional sub-classes, (I-1) functions that are applied on the system inputs and (I-2) functions that are applied on internal nodes. To conclude with, we use a function that enters a node in serial to an input node (S1), a function that enters a node in parallel to an input node (P1), a function that enters a node in serial to an internal node (S2) and, finally, a function that enters a node in parallel to an internal node (P2). In *Figure 5*, we demonstrate the operation regarding function S1. The application of this function on the B node in *Figure 5(a)*, results in the construction of the network shown in *Figure 5(b).* The grey-coloured arrow shows the running cursor, which marks the point from which any further network expansion will occur. *Figure 6* illustrates the operation of P1 function. Application of this function on the B node in *Figure 6(a)*, results to the network shown in *Figure 6(b)*. *Table 1* depicts in short the functions that we used in order to describe the evolutionary neural logic networks. The system grammar is presented in *Table 2*. Initial symbol (root) of a tree can be a node of a type <PROG>. The logical functions that construct in this work the operator set for the neural logic networks expressed in our system are shown in *Table 3*.

**Table 1.** GP- function set for neural logic networks.

| Class | Function | Operation |
|---|---|---|
| Architecture altering | P1 | Enters a node in parallel to an input node |
| | S1 | Enters a node in serial to an input node |
| | P2 | Enters a node in parallel to an internal node |
| | S2 | Enters a node in serial to an internal node |
| | PROG | Initial function. Creates the embryonic network |
| Parameter tuning | CNR | Applies logical operator based on the CNRSEL and K values |
| | LNK | Cuts links based on the NUM and CUT values |
| | IN | Enters an input parameter value into the network |
| | NUM | Supplementary to the LNK function, selects the link to cut |
| | CUT | Supplementary to the LNK function, determines whether the link will be cut or not |
| | CNRSEL | Supplementary to the CNR function, its value determines the operator that will be applied to a node |
| | K | Supplementary to the CNR, function, its value determines the logical operator's parameter (determined by CNRSEL). |

**Table 2.** Context-free grammar for neural logic networks.

| | | |
|---|---|---|
| <PROG> | : = | **PROG** <PLACE1><SYNAPSE> |
| <PLACE1> | : = | **S1** <PLACE1><SYNAPSE><PLACE2><br>\| **P1** <PLACE1><PLACE1><br>\| **IN** |
| **IN** | : = | Data attribute (system input) |
| <PLACE2> | : = | **S2** <PLACE2><SYNAPSE><PLACE2><br>\|**P2** <PLACE2><SYNAPSE><PLACE2><br>\| **E** |
| **E** | : = | ∅ |
| <SYNAPSE> | : = | **LNK** <NUM><CUT><SYNAPSE><br>\| **CNR** <CNRSEL><K> |
| <NUM> | : = | **NUM** |
| <CUT> | : = | **CUT** |
| <CNRSEL> | : = | **CNRSEL** |
| <K> | : = | **K** |
| **NUM** | : = | Integer in [1,256] |
| **CUT** | : = | Integer in [0,1] |
| **CNRSEL** | : = | Integer in [0,10] |
| **K** | : = | Integer in [0,9] |

## 4 Results and discussion

The proposed system was applied in a banking sector aiming to both provide high classification rates and the ability to interpret the extracted solution. This data is acquired by an Australian bank [12]. The features in this data are given as simple elements and their interpretation is not known, since this data set has been considered as classified information. However, the application of our methodology to this problem may provide useful conclusions regarding the system's effectiveness, since enough successful applications of other approaches exist [11], [14], in related literature. The data is consisted of 688 records from which 344 were randomly used as training set, 172 as validation set and another 172 as test set (unknown data). There are 67 missing values in total, which were substituted with zeroes. *Table 4* presents the related attributes and the encoding that was used. This data is primarily composed by discrete attributes, which are encoded into independent binary features for further processing by our system. The percentage of cases for which the application was finally accepted, reaches 44.5 % of the total (307 records). A total of 37 records have one or more missing values (5% of total data). After the training process was accomplished (200 genetic programming generations), our algorithm generated the neural logic network shown in *Figure 7*.

**Table 3.** Implemented logical functions.

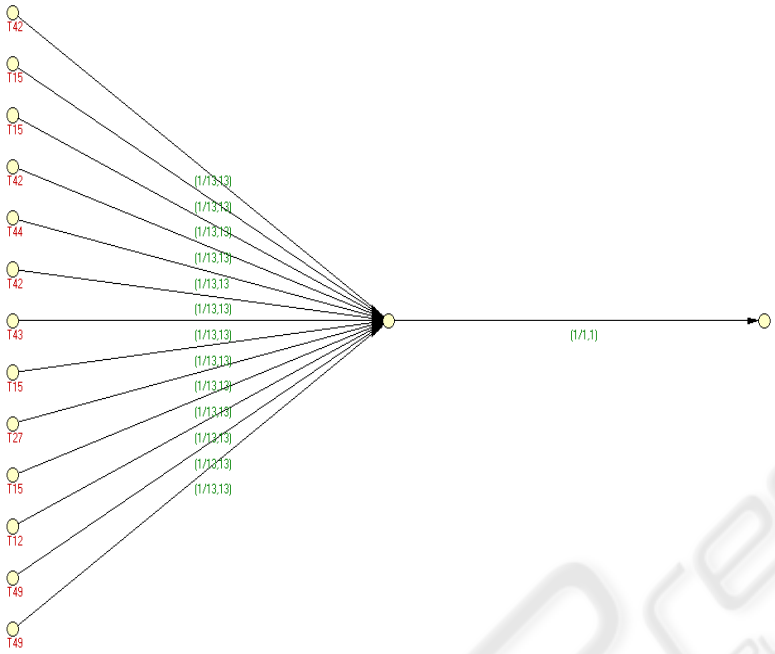
| |
|---|
| Conjunction (AND) |
| Disjunction (OR) |
| Priority |
| At least k-true |
| At least k-false |
| Majority influence |
| Majority influence of k |
| 2/3 majority |
| Unanimity |
| IF-Then (Kleene' s model) |
| (logical) Difference |
| Exclusive OR (XOR) |
| Negative exclusive OR (XNOR), or equivalence (EQV) |
| Negative conjunction (NAND) |
| Negative disjunction (NOR) |
| Exactly k-true |

**Table 4.** Feature description for the Australian bank credit-scoring problem.

| Variable | Values | Encoded features |
|---|---|---|
| T1-T3 | discrete (b, a, ?), 12 missing values | 1 (discrete) → 3 binaries, 1 of 3 (b="1 0 0", a="0 1 0", ?="0 0 1") |
| T4-T5 | continuous (13.75...80.25) | 1 (continuous) → 2 ( 1 continuous 0..1, 1 binary, 1: absent 0: non-absent) |
| T6 | continuous (0...28) | 1 (continuous) → 0..1 |
| T7-T11 | discrete (u, y, l, t, ?), 6 missing values | 1 (discrete) → 5 binaries, 1 of 5 |
| T12-T15 | discrete (g, p, gg, ?), 6 missing values | 1 (discrete) → 4 binaries, 1 of 4 |
| T16-T30 | discrete (c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff, ?), 9 missing values | 1 (discrete) → 15 binaries, 1 of 15 |
| T31-T40 | discrete (v, h, bb, j, n, z, dd, ff, o, ? ), 9 missing values | 1 (discrete) → 10 binaries, 1 of 10 |
| T41 | continuous (0...28.5) | 1 (continuous) → 0..1 |
| T42 | binary | 1 (binary) → 0..1 |
| T43 | binary | 1 (binary) → 0..1 |
| T44 | continuous (0...67) | 1 (continuous) → 0..1 |
| T45 | binary | 1 (binary) → 0..1 |
| T46-T48 | discrete (g, p, s) | 1 (discrete) → 3 binaries, 1 of 3 |
| T49-T50 | continuous (0...2000),13 missing values | 1 (continuous) → 2 ( 1 continuous 0..1, 1 binary, 1: absent 0: non-absent) |
| T51 | continuous (0...10000), 13 missing values | 1 (continuous) → 0..1 |

The classification rate of this solution in the test set (unknown data) reaches 89.53% (154/172) which is higher than those reported in literature [12]. The corresponding classification rates in the training and the validation set were 85.47% (294/344) and 87.21% (150/172) respectively. The generated neural logic network can be described by the following extremely simple decision rule:

Q ← *Conjunction* (AND) (T12, T15, T15, T15, T15, T27, T42, T42, T42, T43, T44, T49, T49)




(CNLN (P1 (P1 (In T42) (P1 (In T15) (P1 (In T15) (P1 (P1 (In T42) (P1 (In T44) (P1 (P1 (P1 (In T42) (P1 (In T43) (P1 (In T15) (In T27)))) (In T15)) (In T12)))) (In T49)))))) (In T49)) (Rule 0 0))))

**Fig. 7.** Neural logic network (description and representation) generated for the Australian credit-scoring problem.

The main conclusions regarding this extracted decision-rule follow:

· Only one simple attribute conjunction (AND operation) is contained in the resulting network, a result showing that no complex rules should necessarily be expected for building up an effective decision – making strategy from the bank's viewpoint, regarding credit applicants' evaluation.

· Getting into greater detail, attribute (feature) T15 seems to be of significant importance for the overall decision-making process. This feature is binary. The initial data feature from which T15 was derived, receives values from the set {g, p, gg, ?}, with T15 corresponding to the last value.

· Feature T42 is also of significant importance for the decision-making process.

**Table 5.** Comparison among various approaches for the Australian bank data [14].

| Methodology | Classification rate in unknown data (%) |
|---|---|
| Cal5 | 86.9 |
| Itrule | 86.3 |
| LogDisc | 85.9 |
| Discrim | 85.9 |
| Dipol92 | 85.9 |
| Radial | 85.5 |
| Cart | 85.5 |
| Castle | 85.2 |
| Bayes | 84.9 |
| IndCart | 84.8 |
| BackProp | 84.6 |
| C4.5 | 84.5 |
| Smart | 84.2 |
| BayTree | 82.9 |
| KNN | 81.9 |
| Ac2 | 81.9 |
| NewId | 81.9 |
| LVQ | 80.3 |
| Alloc80 | 79.9 |
| Cn2 | 79.6 |
| QuaDisc | 79.3 |
| Default | 56.0 |
| This work (NLN) | 89.5 |

· Another significant feature is the T49, which in the initial data set receives values within the range [0,2000].

The Australian bank data problem is particularly interesting, since it enables the comparison of our model with a number of competitive statistical and intelligent approaches. *Table 5* presents our results as compared with 22 other competitive approaches presented in [14]. Among the provided methodologies of *Table 5*, the proposed approach succeeded in obtaining the highest classification score using the specific neural logic network.

## 5 Conclusions and Further Research

Neural logic networks are a family of artificial neural networks designed with the aim to provide network interpretation into simple logical rules. Although successful in representing any set of logical rules into a network structure, the opposite has been proved unsuccessful. This result derived from the training procedures used so far which, based on edge tuning, destroyed the network's interpretation. Aiming to successfully use the neural logic networks family in the CI framework, this paper presented an architecture-altering and training methodology. This methodology overcomes the problems encountered in previous approaches and succeeds in producing highly accurate and interpretable neural logic networks. The methodology is tested in a problem from the banking domain with very encouraging results. It is acknowledged that further experiments are needed in similar or different domains in order to obtain a wider valuation of the examined methodology.

Further research includes the application of the system into more domains, especially from the financial sector where the neural networks have been proved an effective classifying and forecasting methodology and at the same time, the interpretation of a network has always been a desirable target. Moreover, the research will be driven to

implement higher order recursive neural logic network, in an attempt to develop neural logic networks for financial problems dealing with time-series data.

## References


1. Chia H.W-K., Tan C-L., 2001. Neural logic network learning using genetic programming, *Intl. Journal of Comp. Intelligence and Applications*, 1:4, pp 357-368
2. Gruau F., 1994. Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm, *Ph.D. Thesis, Ecole Normale Superieure de Lyon*, ftp:lip.ens-lyon.fr (140.77.1.11) pub/Rapports/PhD PhD94-01-E.ps.Z
3. Gruau F., Whitley D., Pyeatt L., 1996. A Comparison between Cellular Encoding and Direct Encoding for Genetic Neural Networks, in *Koza J.R., Goldberg D.E., Fogel D.B., Riolo R.L. (eds.), Genetic Programming 1996: Proceedings of the First Annual Conf.*, pp 81-89, Cambridge, MA, MIT Press
4. Hussain T., Browse R., 1998. Attribute Grammars for Genetic Representations of Neural Networks and Syntactic Constraints of Genetic Programming, *in AIVIGI'98:, Workshop on Evol.Comp*., Vancouver BC.
5. Janikow C.Z., 1996. A Methodology for Processing Problem Constraints in Genetic Programming, in *Computers Math.Applic*. Vol.32:8,pp 97-113.
6. Koza J.R., 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA, MIT Press.
7. Koza J.R., Keane M.A., Streeter M.J., 2003. Genetic Programming Human-Competitive Results, *IEEE Intelligent Systems*, May/June 2003, pp 25-31.
8. Montana D.J., 1995. Strongly Typed Genetic Programming, *Evolutionary Computation*, vol. .3, no. 2.
9. Quah T-S., Tan C-L., Teh H-H.,Sriniivasan B., 1995. Utilizing a Neural Logic Expert system in Currency Option Trading, *Expert Systems with Applications*, 9:2, pp 213-222.
10. Quah T-S., Tan C-L., Raman K., Sriniivasan B., 1996. Towards integrating rule-based expert systems and neural networks, *Decision Support Systems*, 17, pp 99-118.
11. Quinlan, J.R., 1987. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27, December: pp. 221-234.
12. Quinlan J.R. 1992. *C4.5: Programs for Machine Learning*, Morgan Kaufmann.
13. Ryan C., Collins J.J., O'Neil M. , 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language, in *W.Banzhaf, R.Poli, M.Schoenauer, T.C.Fogarty (Eds.), Genetic Programming*, Lecture Notes in Computer Science, Springer.
14. Statlog 2005. Statlog Use, Test of Australian Credit Scoring data, available online: http://www.liacc.up.pt/ML/statlog/datasets/australian/australian.use.html, date of last access:05/22/05.
15. Tan A-H., Teow L-N., 1997. Inductive neural logic network and the SCM algorithm, *Neurocomputing* 14, pp 157-176
16. Teh H-H., 1995. *Neural Logic Networks*, World Scientific.
17. Tsakonas A. Dounias G., 2002. A Scheme for the Evolution of Feedforward Neural Networks using BNF-Grammar Driven Genetic Programming, in *Proc. of Annual Workshop of European Network of Excellence on Smart Adaptive Systems 2002*, Eunite-02, Algarve.
18. Wong M.L., 2001. A flexible knowledge discovery system using genetic programming and logic grammars, *Decision Support Systems*, 31, pp 405-428