Proceedings of ECAI International Workshop on Neural-Symbolic Learning and Reasoning NeSy 2006

A.S. d'Avila P. Hitzler G Tamburrini

Department of Computing

City University

Technical Report Series

TR/2006/DOC/02

ISSN 1364-4009

NeSy 2006

2nd International Workshop on Neural-Symbolic Learning and Reasoning

In collaboration with the 17th European Conference on

Artificial Intelligence ECAI 2006

http://www.neural-symbolic.org/NeSy06/

http://ecai2006.itc.it/cda/aree/index.php

Riva del Garda, Italy

29 August 2006

NeSy'06 Programme

9.00 Opening

9.00 – 10.00 Keynote: *Marco Gori*, University of Siena, Italy
Heat Kernel Learning Machines for Symbolic Problems

10.00 – 10.30 coffee break

10.30 – 11.00 Evidence Based Reasoning in Classifier Hierarchies
Rebecca Fay, Friedhelm Schwenker and Günther Palm

11.00 – 11.30 Using Activation Spreading for Ontology Merging
Miloslaw Frey

11.30 – 12.00 On the Capacity of Unsupervised Recursive Neural Networks for Symbol Processing
Barbara Hammer and Nicolas Neubauer

12.00 – 12.15 Position paper: Towards a Dynamic Assessment of Formal Language Complexity
Andre Grüning

12.15 – 14.00 lunch break

14.00 – 14.30 An Approach to Language Understanding and Contextual Disambiguation in Human-Robot Interaction
Heiner Markert and Günther Palm

14.30 – 15.00 Active Sonar Target Identification using Evolutionary Neural Logic Networks
Athanasios Tsakonas, Georgios Dounias and Nikitas Nikitakos

15.00 – 15.30 Construction of Neurules from Training Examples: A Thorough Investigation
Jim Prentzas and Ioannis Hatzilygeroudis

15.30 – 16.00 coffee break

16:00 – 16.30 Towards the Integration of Abduction and Induction in Artificial Neural Networks
Oliver Ray and Artur d'Avila Garcez

16.30 – 17.30 Keynote: *Stefan Wermter*, University of Sunderland, UK
Hybrid Intelligent Systems and Cognitive Robotics

17.30 ECAI 2006 Opening

Table of Contents

Organising Committee

Artur Garcez is a Senior Lecturer at the Department of Computing at City University, London. He is the author of about 60 publications on Machine Learning and the integration of Logics and Neural Networks. His research has evolved from the theoretical foundations of Neural-Symbolic systems to their application in Bioinformatics and Software Engineering. Dr. Garcez is an author of the book Neural-Symbolic Learning Systems: Foundations and Applications, published by Springer-Verlag in 2002, and of the forthcoming book Connectionist Non-Classical Logics, to be published in 2006. He is an area scientific editor (Logics and Neural Networks) of the Journal of Applied Logic, Elsevier, an area editor (Reasoning and Learning) of the Journal of Logic and Computation, Oxford University Press, and a member of the advisory board of the Cognitive Technologies book series, Springer-Verlag. He has organized the First International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'05, at IJCAI-05, has served and serves on the committees of a number of international conferences and workshops, and has acted as a reviewer for a number of international journals on Logic and Artificial Intelligence. He is grateful to The Royal Society for their financial support to the NeSy workshop series. He is a member of the British Computer Society, the City and Guilds College Association and a Visiting Research Fellow at the Department of Computer Science, King's College London. He holds an M.Eng. in Computing Engineering, an M.Sc. in Computing and Systems Engineering and a Ph.D. (D.I.C.) in Computing. For more information, please see http://www.soi.city.ac.uk/~aag

Pascal Hitzler is Assistant Professor at the Institute for Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe in Germany. His research record lists over 80 publications in such diverse areas as neural-symbolic integration, knowledge representation and reasoning, lattice and domain theory, denotational semantics, and set-theoretic topology. He is Programme Co-Chair at the 14th International Conference on Conceptual Structures, ICCS06, in Aalborg, Denmark, and co-organiser of the Workshop OWL – Experiences and Directions OWLED06 in Athens, GA. He has recently organized the First International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'05 at IJCAI-05 and the Workshop on Reasoning on the Web, RoW06, at the 2006 World Wide Web Conference, WWW06, in Edinburgh. He is currently editing a book on the workshop topic which is to appear in the Springer series on Computational Intelligence. He is currently writing a german textbook on Semantic Web which will be published by Springer, and he teaches in the Semantic Web Academy Karlsruhe. He is involved in a number of large-scale EU projects including NeOn, SEKT and KnowledgeWeb. He leads AIFB's activity in the German lead project SmartWeb, funded by the German Ministry for Education and Research (BMBF). He serves as a reviewer for international journals, conferences, and research project applications. He has also been an organizer of international enhancement programmes for highly skilled students in Mathematics and Computer Science, and has served as an editor for several books in this area. For more information, please see http://www.pascal-hitzler.de

Guglielmo Tamburrini is Professor of Logic and Philosophy of Science at the Università di Napoli Federico II. Formerly associate professor of Logic and Philosophy of Science at the University of Pisa (1998-2004), and researcher at Istituto di Cibernetica of CNR (Italian National Council of Research, 1984-1998). He has recently organized a special session on Epistemology of Computing at the First Computability in Europe Conference (Amsterdam, June 2005), co-organized the International Workshop on Models of Computation and Natural Processes (Bologna, June 2005), and the International Workshop on the confluence of ideas in 1943: 60 years of machine-oriented investigations of intelligent behaviour (Pisa, November 2003). He is coordinating the two-year (2005-07) EU CA project ETHICBOTS: Emerging techno-ethics of human interactions with bionic, robotic and communication systems, which involves research groups from 5 European countries. Current research interests chiefly concern real-time reasoning and perception for robotic systems, and the methodology of AI and the cognitive (neuro-)sciences. More closely related to the NeSy workshop series is his work on neural-symbolic integration in perceptual systems, robotic control, and knowledge-based systems. Guglielmo Tamburrini is the author of about 70 publications, has served as reviewer for international journals and conferences in cognitive science and philosophy of science, and has given invited talks at conferences in Austria, Brazil, Italy, Spain, and the UK. For more information, please see http://www.cibernetica.unina.it/tamburrini.php

NeSy Programme Committee

Introduction

The importance of the efforts to bridge the gap between the connectionist and symbolic paradigms of Artificial Intelligence has been widely recognised. The merging of theory (background knowledge) and data learning (learning from examples) in neural networks has been indicated to provide a learning system that is more effective than purely symbolic or purely connectionist systems, especially when data are noisy.

The above results, which are due also to the massively parallel architecture of neural networks, contributed to the growing interest in developing Neural-Symbolic Learning Systems, i.e. hybrid systems based on neural networks that are capable of learning from examples and background knowledge, and of performing reasoning tasks in a massively parallel fashion. Typically, translation algorithms from a symbolic to a connectionist representation and vice-versa are employed to provide either (i) a neural implementation of a logic, (ii) a logical characterization of a neural system, or (iii) a hybrid system that brings together features from connectionism and symbolic Artificial Intelligence.

However, while symbolic knowledge representation is highly recursive and well understood from a declarative point of view, neural networks encode knowledge implicitly in their weights as a result of learning and generalisation from raw data. The challenge for neural-symbolic systems, therefore, is to combine neural networks' robust learning mechanisms with symbolic knowledge representation, reasoning, and explanation capability in ways that retain the strengths of each paradigm.

This workshop brings together researchers in the fields of neural-symbolic integration, neural computation, logic and artificial intelligence, and computational neuroscience, as well as experts in robotics and semantic web applications of neural-symbolic systems. The workshop aims to focus on principled ways of integrating neural computation and symbolic artificial intelligence w.r.t. knowledge representation, reasoning, learning, and knowledge extraction. Towards this goal, the papers in the workshop address all facets of neural-symbolic integration, including:

- The representation of symbolic knowledge by connectionist systems;
- Integrated neural-symbolic learning approaches;
- Extraction of symbolic knowledge from trained neural networks;
- Integrated neural-symbolic reasoning;
- Biological inspiration for neural-symbolic integration;
- Applications in robotics and semantic web.

The provision of integrated systems for robust learning and expressive reasoning has been identified recently by Leslie Valiant as a key challenge for computer science for the next 50 years (Journal of the ACM, Vol. 50, 2003). Neural-Symbolic integration can rise to this challenge. The area has now reached maturity, as indicated by books recently published in the subject, journals dedicated scientific areas on logic and neural networks, research projects, and a book series dedicated to the integration of symbolic and sub-symbolic computation. There have been isolated workshops in the area in the past, and it is now time for a regular workshop series to serve as a focal point for the community. We hope *Neural-Symbolic Learning and Reasoning* will serve this purpose. We hope it will also become a source for further collaboration between researchers working in the area.

We would like to take this opportunity to thank the members of the programme committee who helped in reviewing and selecting the papers submitted to the workshop, our invited speakers, Prof. Marco Gori and Prof. Stefan Wermter, the authors of the papers submitted to the workshop, and the ECAI-06 workshop chair, Prof. Toby Walsh, for his assistance in the organisation of the workshop.

Riva del Garda, August 2006

Artur d'Avila Garcez , Pascal Hitzler, Guglielmo Tamburrini

Keynote talk: Heat Kernel Learning Machines

Dr. Marco Gori
Dipartimento di Ingegneria dell'Informazione, Via Roma, 56, 53100 Siena – ITALY
marco@dii.unisi.it, htp://www-dii.ing.unisi.it/~marco/

Abstract:
A remarkable number of important problems in different domains (e.g. web mining, pattern recognition, biology ...) are naturally modeled by functions defined on graphical domains, rather than on traditional vector spaces. In this talk, I introduce a general framework for learning functions defined on graphical domains. Using the metaphor of heat propagation, I introduce the concept of heat kernel learning machines (HKLM), and show that they can approximate up to any degree of precision a class of functions, referred to as unfolding-equivalence functions (UEF) that turn out to be of interest in many real-world problems. I sketch the general architecture of the HKLM and discuss a neural network based computation at node level. The corresponding weights can be discovered from supervised examples using algorithms inspired to connectionist learning. The basic idea is to adopt a special dynamic behavior that arises from forcing a contraction map in the HKLM. In the extreme case, the parameters are shared amongst the nodes of the graphs, but one can group nodes so as to share the weights within the group. Interestingly, I show that the adoption of different weights for different classes of nodes makes it possible to extend the methodology to the case in which the function takes values on the arcs.

I give some very promising experimental results on a number of graph problems and for functions involved in link analysis, like PageRank. I also show that similar performance holds for extensions of PageRank in which the function also depends on the content of the page. I claim that the propagation of the relationships expressed by the arcs in the graphical domain reduces dramatically the sample complexity with respect to traditional learning machines, thus making HLKMs suitable to many large scale real-world problems (e.g. spam detection and complex page categorization).

Biography:
Marco Gori is professor of computer Science at the University of Siena. His research interests are in the field of artificial intelligence, with emphasis on machine learning. He is especially involved in the conception of new theories of learning in structured domains and in their applications to pattern recognition and mining the web. He has been the President of the Italian Association for Artificial Intelligence and is currently acting as the chairman of the Italian Chapter of the IEEE Computational Intelligence Society. Dr. Gori is a fellow of the IEEE.

Keynote talk: Hybrid Intelligent Systems and Cognitive Robotics

Professor Stefan Wermter

Chair for Intelligent Systems
School of Computing and Technology
University of Sunderland
St Peters Way
Sunderland SR6 0DD
United Kingdom

email: stefan.wermter AT sunderland.ac.uk
http://www.his.sunderland.ac.uk/~cs0stw/
http://www.his.sunderland.ac.uk/

Abstract:
There has been substantial progress in both hybrid intelligent systems and cognitive robotics in recent years. While in the past robots were most successful in traditional industrial environments, new generations of hybrid intelligent robotic systems are being developed which focus on higher cognitive capabilities, including reasoning, learning and language communication. In this talk we will give an overview of learning neural robots from a perspective of integrative hybrid intelligent systems and illustrate some new developments including also examples under development in the Centre for Hybrid Intelligent Systems at the University of Sunderland (www.his.sunderland.ac.uk).

Biography:
Professor Stefan Wermter holds the Chair in Intelligent Systems at the University of Sunderland, UK and is the Director of the Centre for Hybrid Intelligent Systems. His research interests are in Intelligent Systems, Neural Networks, Cognitive Neuroscience, Hybrid Systems, Language Processing, and Learning Robots. He has an MSc from the University of Massachusetts, USA and a PhD and Higher Doctorate (Habilitation) from the University of Hamburg, Germany, all in Computer Science and was a Research Scientist at Berkeley, USA before joining the University of Sunderland. Professor Wermter has written or edited five books and published about 130 articles on this research area, including books like "Hybrid Connectionist Natural Language Processing", "Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing", "Hybrid Neural Systems" and "Emergent Neural Computational Architectures based on Neuroscience".

Evidence Based Reasoning in Classifier Hierarchies

Rebecca Fay and **Friedhelm Schwenker** and **Günther Palm**[1]

Abstract. Hierarchical neural networks naturally combine subsymbolic information processing with symbolic information as they consist of several neural classifiers which provide hierarchically structured knowledge. This knowledge implies a particular uncertainty which is indicated by the magnitude of the classifier outputs. There are different ways to combine this expert knowledge to a collective output. Two different methods are evaluated in this paper: a method similar to the decision tree approach and an evidence theoretic approach utilising Dempster-Shafer theory. The proposed approaches have been evaluated using three different data sets and two different types of classifiers. It was shown that the evidence theoretic approach yields improved classification performance.

1 INTRODUCTION

Hierarchical relationships among objects occur rather often. In particular, hierarchical grouping of similar objects seems reasonable. This similarity can refer to different characteristics such as functionality or appearance of objects.

Hierarchical neural networks consist of multiple classifiers arranged in a hierarchical manner where the individual classifiers provide evidence at different levels of abstraction, i.e. the individual classifiers give results for not necessarily single classes but sets of classes. The evidence provided by the single classifiers represents measures for the likelihood of a given sample to belong to a certain class or group of classes.

Thus a hierarchical neural network can be interpreted as a group of hierarchically arranged experts which make hierarchically structured statements, i.e. experts at higher levels of the hierarchy make rough decisions concerning comprehensive groups of classes and experts at low levels provide detailed information about few single classes.

There are diverse ways of obtaining a collective result on the basis of the opinions of the various experts. One way is to attain the result in stages by propagating the decision down the hierarchy, i.e. the decision is delegated hierarchically. At each level a selected expert makes a decision at his level of detail and based on this decision he chooses the expert at the next level who has to make a more detailed decision. Thus the decision is propagated down the hierarchy until the final expert at the lowest level conclusively decides what the result is. Thereby not all experts are consulted but only those experts on the path which emerged. Another way is to incorporate the opinions of all experts and to combine them to one conclusive result. The integration of the different expert opinions is a form of reasoning.

A suitable approach for combining hierarchically structured knowledge incorporating uncertainty is the well-established Dempster-Shafer evidence theory. It provides means of dealing with information provided at different levels of abstraction without enforcing to assign information at a more detailed level than is justified. Moreover, it offers a possibility to represent lack of knowledge and doubt. The first characteristic facilitates the dealing with the hierarchical information provided by the classifier hierarchy. The latter property accounts for the necessity of the individual classifiers to be able to state that a given sample belongs to an unknown class. This is essential as not all classifiers within the hierarchy provide information about all classes, but only deal with a specific subset of classes and thus are likely to have to give results for classes they have no knowledge about.

2 METHOD

In this paper a method to combine the results of multiple hierarchically arranged classifiers utilising evidence based reasoning is presented. This method is compared with a simple decision-tree-like approach for retrieving the classification results. Both approaches are applied to the same hierarchy, i.e. the hierarchy generation and training is the same for both methods.

In the following the main components of the proposed approach are presented. Hierarchical neural networks are briefly introduced and the two methods for evaluating the hierarchy are explained.

2.1 Dempster-Shafer Evidence Theory

Dempster-Shafer evidence theory [4, 5, 16] is a mathematical theory of evidence and plausibility reasoning. It provides means of representing and combining measures of evidence. Major advantages of this theory are the possibility to differentiate between ignorance and uncertainty, the ability to easily represent evidence at different levels of abstraction and the possibility to combine evidence from different sources. In the following the basic concepts of the Dempster-Shafer evidence theory relevant for the proposed approach are briefly explained.

Let Ω be a finite set of q mutually exclusive atomic hypotheses $\Omega = \{\theta_1, ..., \theta_q\}$ called the *frame of discernment* representing the universe of discourse and let 2^Ω denote the power set of Ω.

A *basic probability assignment* or *mass function* m over a frame of discernment Ω is a function $m : 2^\Omega \mapsto [0, 1]$ that satisfies the following two conditions:

$$\begin{aligned} m(\emptyset) &= 0 \\ \textstyle\sum_{A \subseteq \Omega} m(A) &= 1 \end{aligned} \tag{1}$$

The mass $m(A)$ specifies the belief in hypothesis A which does not need to be atomic, but can be a set of atomic hypothesis. In that case $m(A)$ reflects ignorance in so far as it is not possible to further subdivide this belief in A among the subsets of A. Thus the mass $m(A)$ specifies the degree of belief that is assigned to exactly the set $A \subseteq \Omega$ and not to any subset of A.

[1] University of Ulm, Germany, email: {rebecca.fay, friedhelm.schwenker, guenther.palm}@uni-ulm.de

Two basic probability assignments m_1 and m_2 from two independent sources can be combined via Dempster's combination rule, the so called *orthogonal sum* $m_{1,2} = m_1 \oplus m_2$ which is defined as:

$$m_{1,2}(C) = K^{-1} \cdot \sum_{A,B:A\cap B=C} m_1(A) \cdot m_2(B), \quad \forall C \neq \emptyset \quad (2)$$

where K is a measure for the conflict between the two sources. The conflict K is defined as:

$$K = 1 - \sum_{A,B:A\cap B=\emptyset} m_1(A) \cdot m_2(B) = \sum_{A,B:A\cap B\neq\emptyset} m_1(A) \cdot m_2(B) \quad (3)$$

The orthogonal sum $m_1 \oplus m_2$ does only exists if $K \neq 0$ and the result $m_{1,2}$ is then a basic probability assignment. Otherwise the two sources are said to be totally contradictory.

Within the transferable belief model [18], an interpretation of the Dempster-Shafer theory of evidence, positive masses can be assigned to the empty set $\emptyset$ entailing unnormalised belief functions [17]:

$$m_{1,2}(C) = \sum_{A,B:A\cap B=C} m_1(A) \cdot m_2(B), \quad \forall C \subseteq \Omega \quad (4)$$

A high value for the mass of the empty set $\emptyset$ indicates a high conflict between the sources.

2.2 Classifier Hierarchies

Hierarchical neural networks consist of multiple neural classifiers which are arranged hierarchically and realise a hierarchical output space decomposition. The classification process is decomposed into several stages utilising coarse to fine classification.

The hierarchies [6] are generated by unsupervised k-means clustering with the objective of grouping similar classes together, i.e. classes that are similar with respect to the features used.

The basic idea of hierarchical neural networks is the decomposition of a complex classification problem into several less complex problems. This yields hierarchical class grouping. The hierarchy emerges from recursive partitioning of the original set of classes C into several disjoint subsets C_i until subsets consisting of single classes result. C_i is the subset of classes to be classified by node i, where i is a recursively composed index reflecting the path from the root node to node i. The subset C_i of node i is decomposed into s_i disjoint subsets $C_{i,j}$, where $C_{i,j} \subset C_i$, $C_i = \cup_{j=0}^{s-1} C_{i,j}$ and $C_{i,j} \cap C_{i,k} = \emptyset$. The total set of classes C is assigned to the root node $C_0 = C$. Consequently nodes at higher levels of the hierarchy discriminate between larger subsets of classes whereas nodes at the lowest level classify between single classes. From the application of this divide-and-conquer strategy emerge several simple classifiers, that can be amended much more easily to the decomposed simple classification tasks than one classifier could be adapted to the original complex classification task.

An example of a classifier hierarchy is shown in figure 1. The nodes within the hierarchy represent individual neural networks. Different types of classifiers can be used. We chose radial basis function networks and fuzzy k-nearest neighbour classifiers. A three phase learning algorithm [15] was chosen to train the radial basis function networks.

Hierarchical neural networks naturally provide a link between between symbolic information and sub-symbolic information processing. Feature vectors representing sub-symbolic information are used for the classification, whereas symbolic knowledge is made available concomitantly via the information about the affiliation to certain subsets of classes. Thus the hierarchy does not only provide the information to which class a given sample most likely belongs but also the information to which subsets of classes this sample belongs. The usage of neural networks, fuzzy or probabilistic classifiers allows the representation of uncertainty of the membership to these classes or groups of classes since the original output of the neurons is not discrete but continuous.

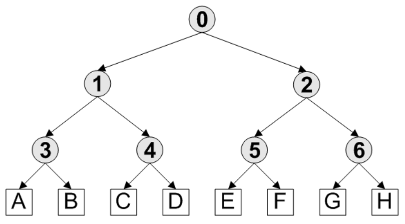


Figure 1. Classifier hierarchy for the classification of eight classes (A, B, C, D, E, F, G, H). Each node within the hierarchy represents a neural network which is used as a classifier. The end nodes represent classes.

2.3 Retrieving the Classification Result in a Decision-Tree-Like Manner

A simple and fast way to obtain the classification result is to evaluate the hierarchy similar to the retrieval process in decision trees where a path from the root node of the hierarchy to the leaf node that specifies the resulting class is determined. Starting with the root node the classification results of the individual classifiers are used to decide which classifier at the next lower level will be looked at next, i.e. to which successor node the decision will be delegated. Classifier i that discriminates between s_i disjoint subsets $C_{i,j}$ decides to which of these subsets C_{i,j^*} the presented sample most likely belongs. As a result the j^*th successor node is the next classifier looked at. This is successively repeated until a leaf node is reached. This evaluation method only considers a subset of the classifiers within the hierarchy. Figure 2 visualises this decision process and shows which classifiers are involved.

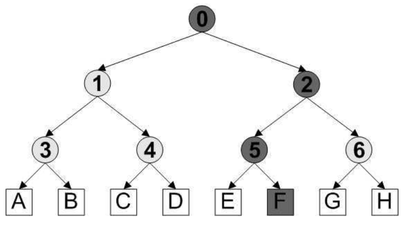


Figure 2. Retrieval of the classification result analogous to decision trees. A path through the hierarchy is determined leading to the resulting class. The highlighted path (in dark grey) shows the nodes activated during the classification of a sample that is classified as class F.

If for a given task only intermediate results are of interest, e.g. whether a sample belongs to a certain subset of classes, it is not necessary to follow through the complete decision process until the final class is obtained, but the process can be aborted at an earlier level.

This methods features a simple way of combining the results of multiple classifiers. It yields good classification results in rather short classification time, but a major disadvantage is the missing ability to correct misclassifications that occur at higher levels of the hierarchy. Hence it would be beneficial not only to take a single path within the hierarchy into account but to consider all classifiers of the hierarchy.

2.4 Evidence Based Reasoning in Hierarchical Neural Networks

A more complex way of combining the results involves all classifiers of the hierarchy. The sample to be classified is presented to all classifiers within the hierarchy and the individual results are then combined to one collective result. The strengths of the individual results are incorporated by this method. The combination is performed utilising Dempster-Shafer theory of evidence.

Figure 3 depicts which classifiers are considered for this decision process.

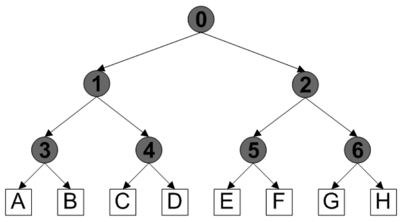


Figure 3. Retrieval of the classification result utilising Dempster-Shafer evidence theory. All classifiers are considered when calculating the classification result.

The application of the Dempster-Shafer evidence theory requires in a first step the calculation of basic probability assignments m_j from the outputs of the individual classifiers within the hierarchy. As not all neural classifiers produce output values that fulfil the requirements for basic probability assignments (equation 1) a transformation of the outputs might be required. The output values of fuzzy k-nearest neighbour classifiers $\Xi_i(x)$ satisfy the conditions for basic probability assignments as the class memberships fulfil the conditions $\Xi_i(x) \in [0,1]$ and $\sum_{i=1}^{l} Xi_i(x) = 1$ whereas the output of radial basis function networks $z_i(x)$ does not necessarily do so. To enforce the fulfilment of the condition $z_i(x) \in [0,1]$ a ramp function

$$\Theta(z_i(x)) = \begin{cases} 0, & x < 0 \\ x, & 0 \leq x \geq 1 \\ 1, & x > 1 \end{cases} \qquad (5)$$

is is applied to the classifier output setting all negative values to zero and all values greater than 1 to 1. This is justified insofar as only a negligible number of output values violate this condition. In order to account for ignorance which is represented by low classifier outputs the difference to one is assigned to Ω. If the sum of the classifier outputs is equal to or greater than one nothing is assigned to Ω. In this case the output is then normalised to sum up to one. Hence in either case the condition $\sum_{i=1}^{l} m_j(i) = 1$ is satisfied. These transformations are applied if necessary to the outputs of all classifiers and then the resulting basic probability assignments m_j of all classifiers are combined using the orthogonal sum without normalisation (equation 4).

According to the structure of the hierarchy each classifier provides evidence for the specific subsets of Ω between which the respective classifier discriminates as well as for Ω. In case of ignorance strong evidence is assigned to Ω.

Additionally, a discounting technique is used propagating classifier responses top down. Thus classifier responses along pathes that at a higher level contain a classifier which showed low responses are weakened strongly whereas pathes below classifiers with strong output are hardly weakened. The discounting is realised by successively multiplying the classifier responses with the classifier output of the respective predecessor node. Hence no discounting is applied to the root node. The discounting accounts for the fact that within the hierarchy there are a not negligible number of classifier that have to provide results for samples belonging to classes they have not been trained with. Hence low classifier responses, as would be desired, cannot be guaranteed in that cases. The discounting thus weakens insular strong responses, which are likely to be caused by a classifier that has been presented a sample of an unknown class. In contrast if only one classifier within a specific path shows a low response but all other classifiers responses are high this leads only to a moderate attenuation. The discounting is applied directly after the transformation of the classifier outputs to basic probability assignments. As a multiplication with the discounting factors $d_i \in [0,1]$ decreases the basic probability assignments if $d_i < 1$, their sum is then smaller than one $\sum_{j=0}^{s_i-1} d_i m_i(C_{i,j}) < 1$. The difference to one originating from this is then assigned to Ω: $m_i(\Omega) = 1 - \sum_{j=0}^{s_i-1} d_i m_i(C_{i,j})$.

3 RESULTS

The proposed approach was evaluated by means of 10 runs of 10-fold cross-validation experiments on three different data sets and with two kinds of classifiers. The data sets used were the Columbia Object Image Library (COIL-20) [13] data set consisting of 20 objects and 72 grey value images per object, the Letter Recognition Image Data [8] comprising 26 letters and 20000 samples in total and the handwritten STATLOG digits data set [9] containing 10 digits and 1000 samples per digit. From the images of the COIL-20 data set orientation histograms [7, 3] were extracted as features for the object recognition. As classifiers radial basis function networks and fuzzy k-nearest neighbour classifiers were used. The usage of the latter is motivated by the simplicity and the low training effort of this approach as well as by the fact that no parameters except k need to be optimised for this type of classifier.

The approach was not only evaluated on automatically generated hierarchies but also on hierarchies that were manually created grouping classes in a plausible manner such that meaningful groups emerge and the classes within one group bear resemblance to each other. Figure 4 depicts the two hierarchies for the COIL-20 data set generated in the described ways.

In all experiments the evidence theoretic approach yield at least the same if not better classification results compared to the decision-tree-like method. The results for the automatically generated hierarchies even show a significant difference on all three data sets.

The results for the automatically and manually generated hierarchies are visualised in figure 5 and 6 respectively by means of box plots and error bars. The tables 1 and 2 list the classification rates for the different experiments performed on the automatically and manually generated hierarchies respectively.

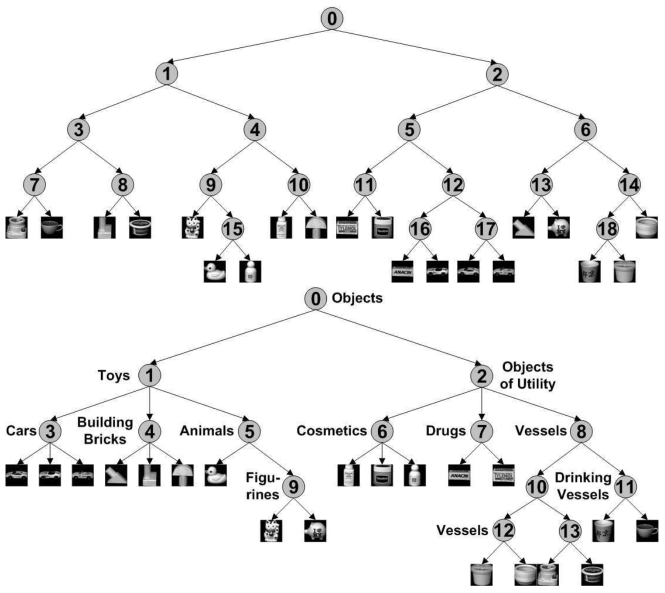


Figure 4. Hierarchies for the classification of the COIL-20 objects. The upper hierarchy was automatically generated by unsupervised k-means clustering, the lower hierarchy was manually created grouping objects in a plausible way such that meaningful groups result.

Data	RBF		3-NN		5-NN	
	DS	DT	DS	DT	DS	DT
Letters	$86.74\pm$ 0.79%	$85.45\pm$ 0.78%	$90.80\pm$ 0.61%	$89.22\pm$ 0.69%	$82.92\pm$ 0.68%	$81.30\pm$ 0.64%
Digits	$94.21\pm$ 0.74%	$93.18\pm$ 0.79%	$94.51\pm$ 0.55%	$94.20\pm$ 0.60%	$90.26\pm$ 0.82%	$89.85\pm$ 0.80%
COIL-20	$96.76\pm$ 1.58%	$95.39\pm$ 2.03%	$99.59\pm$ 0.47%	$99.26\pm$ 0.67%	$92.62\pm$ 1.62%	$92.12\pm$ 1.77%

Table 1. Classification rates for the different data sets on the test data for the Dempster-Shafer method (DS) and the decision tree method (DT) for the radial basis function network (RBF) and fuzzy k-nearest neighbour classifier (k-NN) on automatically generated hierarchies. The evidence theoretic approach outperforms the decision tree approach in all experiments.

A pairwise t-test based on repeated k-fold cross validation with a variance correction [2] to compensate the highly violated independence assumption, called corrected repeated k-fold cross validation test, was conducted to assess the results of the different experiments statistically.

The results of the t-test for the different experiments are listed in tables 3 and 4.

4 DISCUSSION

The evaluation of the classifier hierarchy by means of Dempster-Shafer evidence theory yields improved or at least the same classification results compared to the simple decision-tree-like evaluation method. Hierarchies automatically generated show more stable results than manually generated hierarchies, but the manually hierarchies also show good results.

Data	RBF		3-NN		5-NN	
	DS	DT	DS	DT	DS	DT
Letters	$86.51\pm$ 1.00%	$84.71\pm$ 0.93%	$90.72\pm$ 0.62%	$89.05\pm$ 0.72%	$82.65\pm$ 0.84%	$80.91\pm$ 0.61%
Digits	$94.10\pm$ 0.85%	$93.38\pm$ 0.90%	$94.64\pm$ 0.46%	$93.86\pm$ 0.63%	$89.98\pm$ 0.91%	$89.58\pm$ 0.97%
COIL-20	$96.07\pm$ 2.01%	$95.46\pm$ 2.16%	$99.05\pm$ 0.90%	$98.96\pm$ 0.88%	$92.19\pm$ 1.92%	$92.21\pm$ 1.84%

Table 2. Classification rates for the different data sets on the test data for the Dempster-Shafer method (DS) and the decision tree method (DT) for the radial basis function network (RBF) and fuzzy k-nearest neighbour classifier (k-NN) on manually generated hierarchies. The average classification rates of the Dempster-Shafer approach are mostly higher than the classification rates of the decision-tree method.

Data	RBF		3-NN		5-NN	
	t	p	t	p	t	p
Letters	9.28	$3.53e{-}10$	13.60	$4.12e{-}14$	15.04	$3.14e{-}15$
Digits	4.80	$9.70e{-}4$	3.77	0.0044	3.27	0.0096
COIL-20	4.70	$8.34e{-}6$	1.61	0.1181	2.53	0.0174

Table 3. Results of the corrected t-test for the different data sets on the test data comparing the Dempster-Shafer (DS) method and the decision tree method (DT) for the radial basis function network (RBF) and fuzzy k-nearest neighbour classifier (k-NN) on automatically generated hierarchies. The table gives the p-values as well as the t-value. The t-tests indicates that the evidence theoretic approach outperforms the decision tree approach significantly.

Data	RBF		3-NN		5-NN	
	t	p	t	p	t	p
Letters	9.93	$3.78e{-}6$	14.69	$1.35e{-}7$	11.79	$8.97e{-}7$
Digits	2.09	0.1050	3.53	0.0242	2.03	0.1124
COIL-20	1.25	0.2191	0.99	0.33	-0.10	0.92

Table 4. Results of the corrected t-test for the different data sets on the test data comparing the Dempster-Shafer (DS) method and the decision tree method (DT) for the radial basis function network (RBF) and fuzzy k-nearest neighbour classifier (k-NN) on manually generated hierarchies. The table gives the p-values as well as the t-value. The t-tests indicate that no significant differences between the classification results of the two different methods can be observed for all data sets.

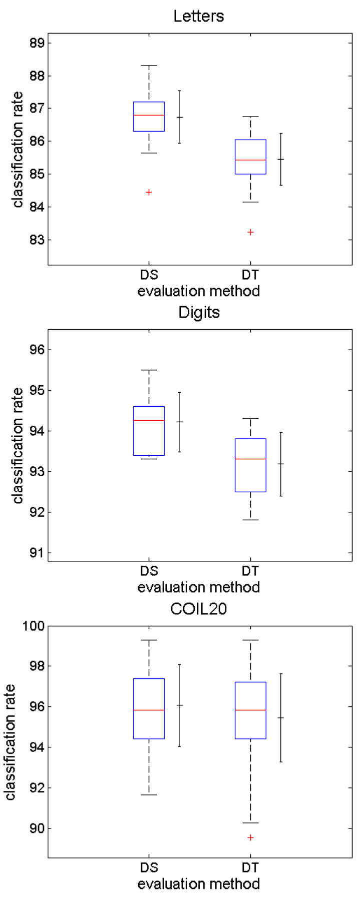


Figure 5. Classification rates for the three data sets (letters, digits, COIL-20) on the test data for the evidence based (DS) and the decision-tree-like (DT) approach on the automatically generated hierarchies. As classifier radial basis function networks were used. The box plots as well as the error bars indicate that Dempster-Shafer methods performs better than the decision tree method on all three data sets.

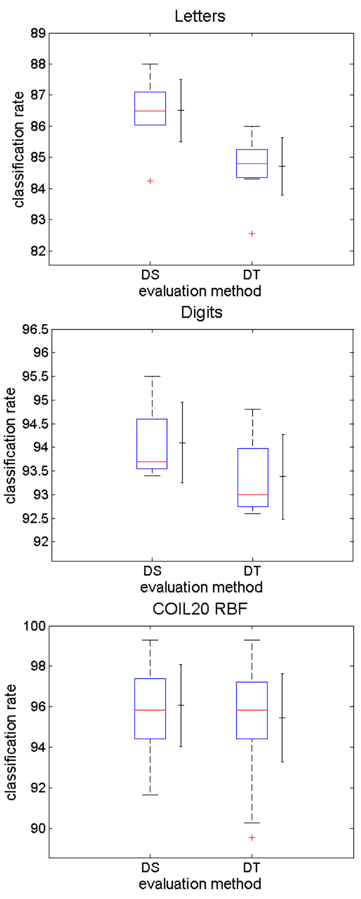


Figure 6. Classification rates for the three data sets (letters, digits, COIL-20) on the test data for the evidence based (DS) and the decision-tree-like (DT) approach on the manually generated hierarchies. As classifier radial basis function networks were used. The box plots as well as the error bars indicate that Dempster-Shafer methods yields the same or even better classification rates than the decision tree method on all three data sets.

A major drawback of the decision-tree-like evaluation method is the missing possibility to later on correct misclassifications that occured at higher levels of the hierarchy. Since the evidence based approach considers all classifiers within the hierarchy, misclassifications at higher levels of the hierarchy can be compensated for if the decisions made by the classifiers at the lower levels are correct. If the misclassification takes place at a leaf node, this wrong decision cannot be corrected any more. The Dempster-Shafer approach can also not compensate for misclassifications where the majority of the classifiers supports the wrong decision.

As all classifiers within the hierarchy need to be evaluated when using the evidence theoretic approach the advantage of the availability of intermediate classification outputs and the resulting savings of computation time, which the decision-tree-like method provides, do not apply. However, the Dempster-Shafer approach provides not only the resulting class but also a measure for the degree of membership of the presented sample to each class.

With regards to computation time the decision tree method outperforms the evidence theoretic approach as not all classifiers are considered and no additional calculations are required. Thus in time-critical applications the decision-tree-like method should preferably be used as it rather quickly yields good classification results.

Since the individual classifiers within the hierarchy can be evaluated independently of each other, all could be evaluated in parallel. Thus the difference in time on multi-processor machines is solely determined by the combination rule which is only slightly more complex for the evidence-based approach. If all classifiers are evaluated in parallel the time aspect becomes less significant.

5 RELATED WORK

Dempster-Shafer evidence theory has been applied to classifier fusion in numerous applications for pattern recognition.

Dempster-Shafer theory was used for multiple classifier fusion in [11]. This approach uses prototype-based classifiers and calculates belief functions from distance measures of different classifiers which are then combined utilising Dempster-Shafer evidence theory. As distance measures the inter-class-distances and intra-class-distances were used. The approach was evaluated in the field of online script recognition.

In [20] classification rates, misclassification rates and rejection rates were used to derive basic probability assignments. Dempster's combination rule is applied to combine the evidences. This approach considers an extra class representing unknown classes or ignorance and it assigns belief to singleton hypotheses, their complement and to the universal proposition Ω. The classifiers used only have class labels as output and do not produce information that can be interpreted as class memberships or other measurements. The approach was applied to the problem of recognising handwritten numerals and scored well compared to other approaches.

A technique closely related to decision templates [10] is used to calculate degrees of belief in [14]. The distances between the classifier outputs for the sample to be classified and the mean classifier outputs calculated on the training samples are transformed into basic probability assignments. The so calculated evidences are then combined using the orthogonal sum. Several experiments in the field of digits and character recognition have been conducted to test this method and it was also part of the experimental comparison of the decision templates approach for classifier fusion to other well-established methods in [10] where it compared favourably well.

In [1] this approach has been varied by using reference outputs adapted to the training data so that the overall mean square error is minimised instead of simply using the mean classifier outputs.

Demspter-Shafer evidence theory is used to combine the normalised outputs of multiple classifiers and to reject samples in case of highly conflicting information in [19].

If at all, these approaches only exploit the possibility to allocate evidence to non-atomic hypotheses by assigning masses to atomic hypotheses θ_i and to their not necessarily atomic complement $\overline{\theta_i}$ or to the frame of discernment Ω. The approach presented in this paper utilises this possibility as the classifier hierarchy naturally provides classification results for sets of hypotheses.

In [12] expert knowledge about the domain of application, namely the detection of anti-personnel mines, is used to calculate basic probability assignments not only for atomic hypotheses but also for composite hypotheses. Hence this approach is rather specific and less general than the proposed approach.

6 CONCLUSIONS

The proposed method of evidence based reasoning utilising Dempster-Shafer evidence theory has proven functional for the combination of expert knowledge extracted from classifier hierarchies and shows encouraging results. When applied to hierarchies that have been created automatically the evidence theoretic method to evaluate the hierarchy yields significantly better classification results than the simple decision-tree-like approach. If applied to manually generated hierarchies the classification results are not significantly better but the Dempster-Shafer approach yields better average classification rates. The already good classification results that are achieved with a simple decision-tree-like evaluation method can be further improved using a more complex and in case of parallel evaluation only slightly more time-consuming evidence based evaluation strategy. The hierarchical class grouping inherent to the classifier hierarchy is apparently suitable for being utilised within the framework of the Dempster-Shafer evidence theory and the natural combination of symbolic and sub-symbolic information seems promising.

ACKNOWLEDGEMENTS

This research has been partially supported by the European Union grant #IST-2001-35282 of the MirrorBot project.

REFERENCES

[1] Ahmed Al-Ani, 'A new technique for combining multiple classifiers using the dempster-shafer theory of evidence', *Journal of Artificial Intelligence Research*, **17**, 333–361, (2002).

[2] Remco R. Bouckaert and Frank Eibe, 'Evaluating the replicability of significance tests for comparing learning algorithms.', in *Advances in Knowledge Discovery and Data Mining, Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2004, Sydney, Australia, May 26-28, 2004*, eds., Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, volume 3056 of *Lecture Notes in Artificial Intelligence LNAI*, pp. 3–12. Springer, (2004).

[3] David M. Coppola, Harriett R. Purves, Allison N. McCoy, and Dale Purves, 'The distribution of oriented contours in the real world', *Proceedings of the National Academy of Sciences USA*, **95**(7), 4002–4006, (1998).

[4] Arthur P. Dempster, 'Upper and lower probablities induced by a multivalued mapping', *Annals of Mathematical Statistics AMS*, **38**, 325–339, (1967).

[5] Arthur P. Dempster, 'A generalization of bayesian inference', *Journal of the Royal Statistical Society*, **B**(30), 205–247, (1968).

[6] Rebecca Fay, Ulrich Kaufmann, Andreas Knoblauch, Heiner MArkert, and Gnther Palm, 'Combining Visual Attention, Object Recognition and Associative Information Processing in a NeuroBotic System.', in *Biomimetic Neural Learning for Intelligent Robots. Intelligent Systems, Cognitive Robotics, and Neuroscience.*, eds., Stefan Wermter, Günther Palm, and Mark Elshaw, volume 3575 of *Lecture Notes in Computer Science LNAI*, 118–143, Springer, Berlin, Heidelberg, (2005).

[7] Wiliam T. Freeman and Michael Roth, 'Orientation histograms for hand gesture recognition', in *IEEE International Workshop on Automatic Face- and Gesture-Recognition*, pp. 296–301, Zürich, Switzerland, (1995).

[8] Peter W. Frey and David J. Slate, 'Letter recognition using holland-style adaptive classifiers', *Machine Learning*, **6**(2), 161–182, (1991).

[9] Ulrich H.-G. Kressel, 'The impact of the learning-set size in handwritten-digit recognition.', in *Proceedings of the International Confernece on Artificial Neural Networks, ICANN 1991, Helsinki, Finland*, eds., T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, pp. 1685–1689, Amsterdam, North-Holland, (1991). Elsevier Science Publishers B.V.

[10] Ludmila I. Kuncheva, James C. Bezdek, and Robert P. W. Duin, 'Decision templates for multiple classifier fusion: An experimental comparison.', *Pattern Recognition*, **34**(2), 299–314, (2001).

[11] Eberhard Mandler and Jürgen Schürmann, 'Combining the classification results of independent classifiers based on the dempaster/shafer theory of evidence.', in *Pattern Recognition and Artificial Intelligence PRAI*, pp. 381–393, (1988).

[12] Nada Milisavljevic and Isabelle Bloch, 'Sensor fusion in anti-personnel mine detection using a two-level belief function model.', *IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews*, **33**(2), 269–283, (2003).

[13] Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase, 'Columbia Object Image Library (COIL-20)', Technical Report Technical Report CUCS-005-96, Department of Computer Science, Columbia University, (February 1996).

[14] Galina Rogova, 'Combining the results of several neural network classifiers', *Neural Networks*, **7**(5), 777–781, (1994).

[15] Friedhelm Schwenker, Hans A. Kestler, and Günther Palm, 'Three learning phases for radial-basis-function networks.', *Neural Networks*, **14**, 439–458, (2001).

[16] Glenn Shafer, *A Mathematical Theory of Evidence.*, University Press, Princeton, 1976.

[17] Philippe Smets, 'The combination of evidence in the transferable belief model.', *IEEE Transactions on Pattern Analysis and Machine Learning*, **12**(5), 447–458, (1990).

[18] Philippe Smets and Robert Kennes, 'The transferable belief model', *Artificial Intelligence*, **66**(2), 191–234, (1994).

[19] Christian Thiel, Friedhelm Schwenker, and Günther Palm, 'Using dempster-shafer theory in mcf systems to reject samples.', in *Proceedings of the 6th International Workshop on Multiple Classifier Systems, MCS 2005*, volume 3541 of *LNCS*, pp. 118–127. Springer, (2005).

[20] Lei Xu, Adam Krzyzak, and Ching Y. Suen, 'Methods of combining multiple classifiers and their application to handwriting recognition.', *IEEE Transaction on Systems, Man and Cybernetics*, **22**(3), 418–435, (1992).

Using activation spreading for ontology merging

Miłosław L. Frey [1]

Abstract. An ontology in informational sciences is an explicit representation of a knowledge for a specific thematic domain. Almost each ontology has a taxonomical structure as a backbone, as it usually also describes relations between classes of objects. The paper at hand presents preliminary investigations of an automatic, activation spreading based, procedure that expands and also joins two taxonomies. Thus, it contributes also to ontology merging. The method to merge two taxonomies, which exploits a hybrid transfer architecture, is described and illustrated by an example.

1 INTRODUCTION

Knowledge representation by an ontology is one of the most informative ways to illustrate dependencies and relations among objects. One of the relations, actually one of the most important ones, is the subsumption relation, called *ISA*. The structure which this relation describes is a taxonomy. It usually underlies the more sophisticated ontological complex.

In philosophy, the Ontology is the systematic account of the Being as such. However, in computational sciences, it is not possible to represent all objects and every relation between them due to limited storage and computational capacities. Thus, these ontologies represent knowledge in declarative formalism, and are limited to so-called thematic domains.

The small domain-oriented ontologies are usually not enough to describe phenomena spanning over many knowledge domains. One of the possibilities to overcome this disability is to merge more ontologies within one construction. Although there exist some successful tools and methods for ontology merging like PROMPT (11), ONION (10) or MOMIS (1), none of them operates in a "network manner". In contrast, the here proposed method of taxonomy merging is in the connectionist tradition and uses connectionist methods. These methods allow to exploit the most important feature of connectionist systems, namely generalization. What's more, generalization even enhances the represented knowledge during the merging process.

2 FROM ONTOLOGY TO NETWORK

An ontology usually contains many different types of relations among stored items. The key relation, however, is the subsumption relation, *ISA*. It constitutes a hierarchical build-up of an ontology, and sets a taxonomical structure as its backbone. Indeed, most ontologies are based on such a hierarchy of concepts, and as such, can be regarded also as a taxonomy.

Coming out from this assumption, in the following, I will focus on the subsumption hierarchy and the *ISA* relationship, which constitutes class–superclass structure. The procedure described below bases on the inheritance property in the taxonomy.

In the following, the term "taxonomy" will not be used *exactly* in a formal way, but in order to describe a network of nodes, characterized by sets of features and organized into hierarchical structure. This structure corresponds to the subsumption relation.

According to the "definition" above, a taxonomy is a connectionist network. It is a network of nodes connected with weighted links. Within this network, a spreading activation mechanism is implemented which serves as a mean to transport information from one node to another. The network used here is a spreading activation localist connectionist network in the tradition of McClelland & Rumelhart (8) and Dell (2). Each node has its own label and its own independent meaning. Moreover, the network itself has the shape of the taxonomy in question.

Typically, however, localist connectionism is used for modeling behaviour, whereas for modeling learning, distributed connectionism is preferred (cf. 5). The method presented here modifies the structure of the net and thus makes the system learnable.

From the data representation point of view, the network described in this paper is anchored in the class of *hybrid transfer architectures* (16). The network is created with use of symbolic data, then the processing is performed within a connectionist architecture, and – in the end – new symbolic rules, stored in connection weights, can be extracted from the final network.

3 NETWORK'S SET-UP AND OPERATION

The unique internal architecture of a node used to build the network used here as well as the types and nature of connections between them are presented in this section. The operational idea of this network is slightly similar to the idea of KBANN (15). However, unlike in KBANN, where two independent algorithms (rules-to-network translation and refinement) exist, the here presented method bases on a single algorithm. This algorithm first constructs a network from symbolic definitions of items and then builds a hierarchical structure, complemented by knowledge discovered by generalization from the data available in the symbolic definitions. The analysis is, however, based on connectionist paradigm.

[1] FGAN - FKIE - ITF, Neuenahrer Straße 20, D-53343 Wachtberg, Germany, e-mail: m.frey@fgan.de

3.1 Nodes

The working principle for a node used in the presented network is inspired by the fact, that a biological neuron can perform virtually any operation on the input signal (6; 7; 9). (It must be noted, however, that this biological inspiration does not imply biological plausibility!) Due to a complex internal structure, a node processes the input signal differently with respect to its source. The internal structure of a node is sketched in the figure 1. In the following, the signal processing within a node is outlined.

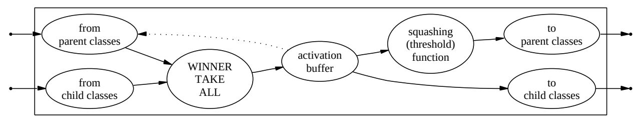


Figure 1. Internal structure of a node.

3.1.1 Signals from parent nodes

Signals coming from parent nodes, that is nodes placed higher in the hierarchy of concepts, are processed in a way similar to calculating a distance in the multi-dimensional space. This multi-dimensional space is defined by incoming connections: their number sets the number of dimensions. Additionally, the weights of those connections set up a point in this phase space. The node calculates the Euclidean distance between the point representing the incoming signal (defined by activations of its parent nodes) and the point set up by weight values of connections coming into the node in question. The final activation function calculates the activation coming from parent nodes basing on current input signal but also takes into account the node's previous activation (this is represented by the reciprocal dotted link from activation buffer in figure 1.)

Thus, a difference between the incoming signal and the signal to which a node is most sensitive to results. It is modified by the node's previous activation.

3.1.2 Signals from child nodes

Signals from child nodes, that is nodes placed lower in the hierarchy of concepts, are calculated simply as a weighted mean of child nodes' activations. Connections strengths are used as weights.

3.1.3 Final activation function.

The final value of activation is a result of a winner-take-all process in which the activation parts coming from parent and child nodes compete. By analogy to the incoming signals, further activation flow is different for signals flowing to feature nodes or class nodes. For connections going in the direction of class nodes a squashing function is used to keep the final value smaller than 1.0. On the other hand, for connections going in the direction of feature nodes, a threshold function applies.

3.2 Connections

Nodes are connected by symmetric, weighted links. They come in two flavors: as excitatory or inhibitory ones. Excitatory connections form the structure of the network (taxonomy). The task of inhibitory connections is twofold (cf. 13). On the one hand, they prevent so called "overheating", the uncontrolled raise of activation values in the network's nodes. On the other hand, they enhance the contrast between nodes that do not belong to the same hierarchy branch.

3.3 Learning

The presented network is a dynamic, constructivist (12) system, designed to evolve along with new data coming from the environment. Such a system must have the ability to learn. In order to accomplish this challenge, it makes use of three standard types of learning (14).

- *Rote learning* is used to simply store input data in the network's structure. This method is comparable to the long-term memory.
- *Connection weight changes:* Changes of connection weights are one of the aspects of the network's dynamics, they shape the working structure of the network: the taxonomy itself.
- *Restructuring* (by creating nodes and connections) is another dynamical process in the development of a taxonomy. The newly added nodes denote taxonomy classes.

3.4 Storing the data

In the first step the presented data is memorized only (stored). This is done by rote learning. Because of the local characteristics of the used data representation, the network must be expanded to store new knowledge. Thus, for each dataset, feature nodes are created when necessary. Additionally, class nodes are created denoting the respective set of all co-occurring features. Between class nodes and feature nodes excitatory connections are created. Their weights correspond to the values of respective features.

3.5 Creating hierarchy

Based on the data stored in already created feature and class nodes, the hierarchy is created. The hierarchy developed in this phase of learning reflects the relations among items only as far as provided explicitly by the input data. One can assume that in most cases the structure of the network gained after this step is not the final hierarchy.

During the hierarchy build up, the network undergoes the following procedure. For each pair of class nodes, both nodes are subsequentially activated. The activation is spread to the feature node layer, and the activation patterns are compared. If one of the nodes generates an activation pattern comprised in the other one's pattern, it is assumed to be its superclass. This principle bases on the simple assumption that a subclass contains all features of its superclass and at least one more, a distinctive one.

3.5.1 Network pruning

The by now described steps of creating a taxonomy lead to a network which usually contains superfluous excitatory connections that do not represent *direct* class – superclass relations. Because the hierarchy creation algorithm discovers only the inclusiveness relations of sets of features, it is the case that all subcategories are linked to the main category even if there are other levels of specifications between.

The superfluous connections are removed by an introspective process. This process analyzes the activation flow between two nodes and compares the activation values in all node pairs. The comparison drives the decision made whether two nodes remain in direct class – superclass relation or not. Roughly speaking, nodes are assumed to lie on the adjacent taxonomical levels when the activation in the subordinate node comes only from the node representing a superclass. Subsequently inhibitory connections are introduced to enhance differences between exemplars presented to the system.

3.5.2 Discovery

The network constitutes so far the representation of raw facts known from input data only. This representation is structured as far as it is provided by this data. That means that relations between classes are known only if they result directly from the definitions.

However, usually more information can be drawn from the underlying data. The discovery procedure is another introspective process which aims at the improvement of the existing network. The process attempts to discover parts of the hierarchy which were not provided explicitly by analyzing pairs of class nodes. If the features for two or more classes overlap (with respect to their presence and value) they form a description of another class which is assumed to be a superclass for those currently being analyzed.

The very final step in the procedure of creating a taxonomy is to clean the network by removing superfluous connections which could have emerged during the discovery phase. The resulting connectionist system reflects the taxonomical structure of the data as far as it could be discovered on the base of the delivered data. Learning is done and the network can be used for "production" purposes.

The so created network is able to generalize, to categorize and to model categories learning. In addition, it displays the cognitive properties like fuzzy categorization, asymmetric category learning, and priming (see 4).

4 TAXONOMY MERGING AND COMPLETION

According to the ontology merging definition (3), one can define a taxonomy merging as a procedure of blending two or more taxonomies into a single one. In this paper, merging of only two taxonomies will be regarded for simplicity. This procedure can be, however, expanded for a case of unlimited taxonomies number by consecutive adding new taxonomies to the result of previous merging process.

There are two common ways of merging a taxonomy: union and intersection. The approach presented here is a union method, that means the resulting taxonomy is the union of all entities in both taxonomies. There is a main problem with merging taxonomies (and ontologies). Objects may be represented differently. In the following section it will be described how this problem can be tackled at least partially within the paradigm at hand.

Taxonomy merging can be regarded as taxonomy completion. In this case, there exists some "main" taxonomy, which is being complemented by the information from the other one. The most important assumption here is that the complement taxonomies have no node, which stands higher in the hierarchy of concepts than a root node of the "main" taxonomy. In other words, the root of "main" taxonomy must remain a root node after merging procedure.

4.1 Knowledge representation

Each entity in a taxonomy is, in our case, represented by a single node in a network. Each node in a hierarchy is characterized by a set of features, which in turn, are represented by nodes in the feature layer. The feature set originates from symbolic input data, namely from symbolic "definitions". An artificial example for two definition sets is given in table 1. It will be used to illustrate the merging mechanism.

item	features
A	root
B	root f11
D	root f12 f21
H	root f12 f22 f32

item	features
C	root f12
G	root f12 f22 f31
F	root f12 f23

Table 1. The example sets of definitions

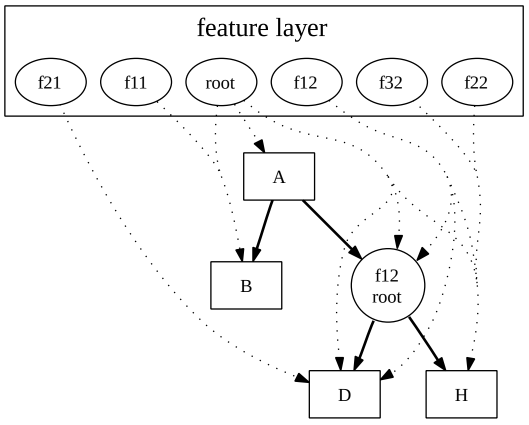
Relations between two entities as well as between entities and features are defined by weighted links. There are two types of links: excitatory and inhibitory ones. The excitatory connections form a taxonomical structure of the network and connect class nodes to feature nodes, while the inhibitory edges serve for enhancing the differences among both single nodes and branches of the represented hierarchy.

4.2 Representation assumption

The problem of different representations of the same concepts in different taxonomies is simplified as follows: While all concepts are described by a label and a set of features, we assume that:

- in case of nodes with the same label, the concepts represented by those nodes are identical, and
- in case of nodes with different labels, the similarity and relation between two concepts in two different hierarchies will be derived from the analysis of features which describe those concepts. In particular, nodes with the same set of features are regarded to represent the same concept even if they have different labels.

a) taxonomy T1



b) taxonomy T2

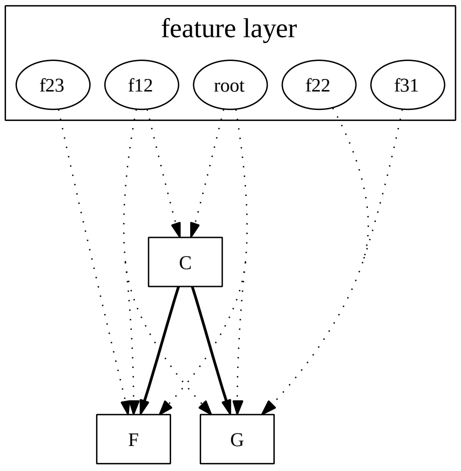


Figure 2. Taxonomies used in the merge example: figure a) corresponds to T1 and b) to T2.

4.3 Method of merging

We start from the situation where there exist two taxonomies, which are regarded to be parts of a bigger hierarchy. This is expressed by either at least one common node within the two taxonomies (a node with the same label) or by common features in the nodes' description. Additionally (as mentioned above), it is known, which one of those taxonomies contains a root node for the hierarchy which has to result from merging.

Merging runs in three main steps: a) searching for features and completing the feature set, b) joining taxonomies, and c) restructuring and pruning. For a descriptional purposes, let us illustrate the merging two taxonomies by an artificial example. The example shows the process of including a taxonomy called T2 into a taxonomy T1 (figures 2a and 2b respectively). (The structure of both taxonomies is derived from the data presented in the table 1.) In the figures, the feature layer is separated from the nodes which constitute the "working" part of a network. Among the latter nodes, the rectangle ones represent nodes which had been defined explicitly in the input data, and the ellipsoid ones those, which had been discovered during creating a network.

4.3.1 Searching for features and completing feature set

The aim of this step is to find and complete the set of features that describe nodes in both taxonomies. It may be the case, that some features are present only in one taxonomy, although they refer in fact to entities in both hierarchies. This is motivated by the fact that the network, which here represents a taxonomy, represents a class of definitional networks, that is networks which are based on definitions (here a definition is identified with a set of features connected to a node). Since definitions are true by definition, the information in the network is assumed to be necessarily true. Thus, if nodes representing the same concepts in two taxonomies have different sets of features, they must be unified.

Of course, the procedure of completing features must be performed with caution: it is assumed that the feature of a given node found in the other taxonomy are included only for nodes from the taxonomy in question which are subnodes of the currently processed one.

The procedure starts with finding two most similar nodes. There are two main cases: taxonomies have nodes with the same label or they have not. In the first case, the task is clear, and a node with corresponding label is chosen. In the second case, the method is as follows. The features corresponding to a currently processed node in the taxonomy T2 are being extracted, and corresponding features in the taxonomy T1 are being activated. Consequently, the activation is being spread over the whole network. Finally, the winner-take-all procedure is performed, which chooses the most similar node.

In the following, the winning node is activated and the activation is spread into the features level. The activated features are being connected to the currently processed node from T2 along with features which have already been present in T2.

The above procedure is repeated for each node from the taxonomy T2. In all consecutive steps, nodes from the taxonomy T2 which already have connection to the feature nodes in the taxonomy T1 are not taken into account.

4.3.2 Joining

After the previous procedure is repeated for all nodes from the taxonomy T2, the networks are joined through the feature level. An example is illustrated in figure 3a), where grayed nodes denote nodes incorporated from the T2 taxonomy, and the black node is a node which is already present in the T1 taxonomy, but which got its label from the T2 taxonomy.

On this stage, the "new" nodes are not yet integrated into the taxonomy T1 itself. To regain the enhanced hierarchical structure, the restructuring process is used.

4.3.3 Restructuring and pruning

The restructuring process serves as a mean to create the taxonomical structure out from the set of nodes connected to the features layer.

During the restructuring process at first a "raw" taxonomy is created, which contains many connections between not adjacent layers (cf. figure 3b). Moreover, it is also possible, that again new nodes are discovered, which were not present in

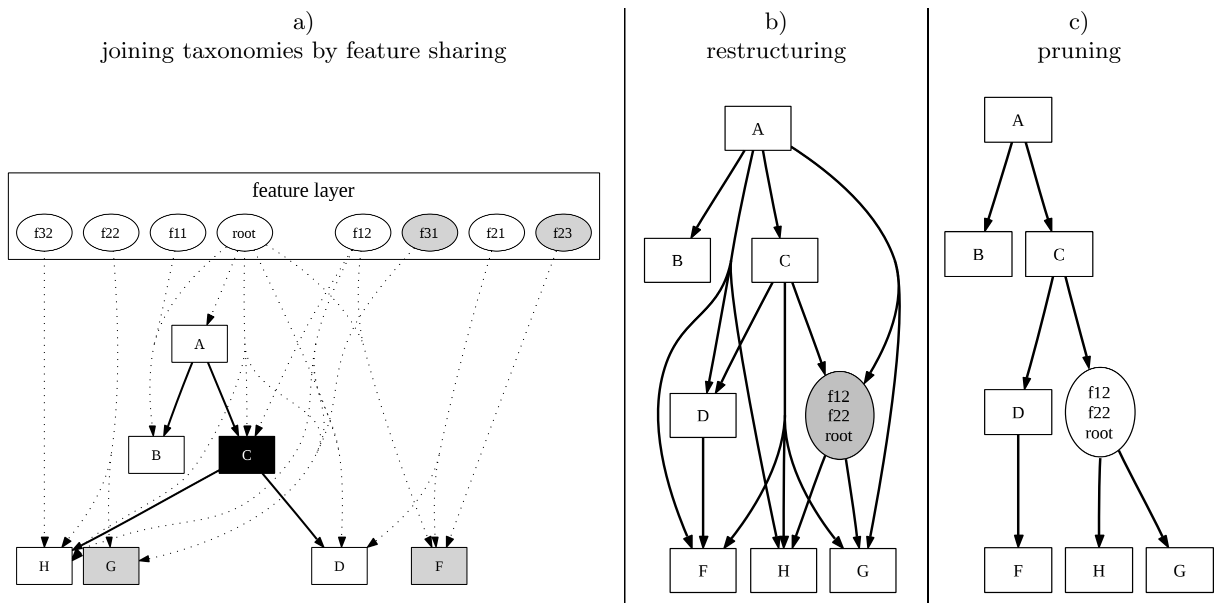


Figure 3. Consecutive steps in the connectionist completion of a taxonomy. See sections 4.3.1 to 4.3.3 for description. (In figures b) and c) the feature level is hidden for readability.)

either from merged taxonomies. Such a node can be seen in figure 3b) and is emphasized with the gray filling.

Not all connections do reflect desired hierarchical dependencies among nodes. The superfluous ones need to be removed. The removal is performed by the introspective process (see section 3.5.1) and leads to the clean taxonomical structure (figure 3c).

5 SUMMARY

The paper at hand presents preliminary investigations on a connectionist method for taxonomy merging. The method for completing one taxonomy with information contained in another was presented and illustrated with an example. The method uses the activation spreading mechanism to first join taxonomies by features sharing, and then to perform an introspective process of restructuring in order to discover even more hierarchical information brought with the new taxonomy.

The most important limitation of the presented method is that it must be known which of two joined taxonomies contains the root node for the taxonomy which results from merging. However, even on this stage of development it can be used for completing taxonomies with new information about hierarchical dependencies in a given thematic domain.

The further development of the connectionist method for taxonomy merging must include discovering the way to make the method symmetrical, that is user should not need to know which from starting taxonomies contains the root of resulting one. This will allow connectionist systems to fully contribute into taxonomy and ontology merging.

REFERENCES

[1] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Beneventano, 'Semantic integration of heterogeneous information sources', *Data Knowl. Eng.*, **36**(3), 215–249, (2001).

[2] Gary S. Dell, 'A spreading-activation theory of retrieval in sentence production', *Psychological Review*, **93**(3), 283–321, (1986).

[3] Marc Ehrig, Jos de Bruijn, Dimitar Manov, and Francisco Martn-Recuerda, 'State-of-the-art survey on ontology merging and aligning v1', SEKT Deliverable 4.2.1, DERI Innsbruck, (JUL 2004).

[4] Miloslaw L. Frey, 'Connectionist contribution to building real-world ontologies', in *Mechanisms, Symbols, and Models Underlying Cognition: First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005, Las Palmas, Canary Islands, Spain, June 15-18, 2005, Proceedings, Part I*, eds., José Mira and José R. Álvarez, volume 3561 of *Lecture Notes in Computer Science*, 489–497, Springer, (2005).

[5] Jonathen Grainger and Arthur M. Jacobs, 'On localist connectionism and psychological science.', in *Localist connectionist approaches to human cognition.*, eds., Jonathan Grainger and Arthur M. Jacobs, Lawrence Erlbaum Associates, Mahwah, New Jersey, (1998).

[6] Christof Koch, Tomaso Poggio, and Vincent Torre, 'Retinal ganglion cells: a functional interpretation of dendritic morphology.', *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, **298**, 227–263, (1982).

[7] Christof Koch, Tomaso Poggio, and Vincent Torre, 'Nonlinear interactions in a dendritic tree: localization, timing and role in information processing.', in *Proceedings*

of the National Academy of Sciences of the United States of America, volume 80, pp. 2799–2802, (1983).

[8] James L. McClelland and David E. Rumelhart, ‘An interactive activation model of context effects in letter perception: Part 1. An account of basic findings’, *Psychological Review*, **88**, 375–407, (1981).

[9] Bartlett W. Mel, ‘Information processing in dendritic trees’, *Neural Computation*, **6**, 1031–1085, (1994).

[10] Prasenjit Mitra, Gio Wiederhold, and Martin L. Kersten, ‘A graph-oriented model for articulation of ontology interdependencies’, in *EDBT ’00: Proceedings of the 7th International Conference on Extending Database Technology*, pp. 86–100, London, UK, (2000). Springer-Verlag.

[11] Natalya F. Noy and Mark A. Musen, ‘The prompt suite: interactive tools for ontology merging and mapping’, *Int. J. Hum.-Comput. Stud.*, **59**(6), 983–1024, (2003).

[12] Steven R. Quartz and Terrence J. Sejnowski, ‘The neural basis of cognitive development: A constructivist manifesto’, *Behavioral & Brain Sciences*, **20**(4), 537–596, (1997).

[13] Ulrich Schade and Thomas Berg, ‘The role of inhibition in a spreading-activation model of language production. ii. the simulational perspective’, *Journal of Psycholinguistic Research*, **21**, 435–462, (1992).

[14] John F. Sowa, ‘Semantic networks’, in *Encyclopedia of Artificial Intelligence*, ed., S. C. Shapiro, John Wiley & Sons, New York, (1992).

[15] Geoffrey G. Towell and Jude W. Shavlik, ‘Knowledge-based artificial neural networks’, *Artificial Intelligence*, **70**, 119–165, (1994).

[16] Stefan Wermter, ‘The hybrid approach to artificial neural network-based language processing’, in *Handbook of Natural Language Processing*, eds., H. Moisl R. Dale R and H. Somers, 823–846, Marcel Dekker, (2000).

On the capacity of unsupervised recursive neural networks for symbol processing

Barbara Hammer [1] and **Nicolas Neubauer** [2]

Abstract. A variety of unsupervised neural models for time series processing has recently been proposed, whereby a large number of models can be derived from a common dynamic equation which also generalizes standard supervised recursive networks. The key point concerns the choice of representation of the temporal context. Interestingly, different choices of the context lead to different model capacities which can be characterized in terms of classical symbol processing systems. In this contribution, we give a systematic overview about existing results and we prove several new equivalences.

1 INTRODUCTION

Time constitutes an ubiquiteous characteristic of natural signals such as sensor streams or language. When processing such signals, humans possess remarkable capabilities with respect to accuracy, speed, noise tolerance, adaptivity and generalization ability for new stimuli. Self-organization plays a major role to achieve this capacity. As demonstrated in numerous applications [13, 16], self-organizing principles allow the development of faithful topographic representations leading to clusters of given data, based on which an extraction of relevant information and supervised or unsupervised postprocessing is easily possible. However, popular self-organizing systems are restricted to standard vectorial data [16], and the capacity of extensions of the basic models to time signals is often limited [3, 15].

In the last years, several complex unsupervised models for time-series processing have been proposed [1]. In principle, one can distinguish the following possibilities to deal with time signals:

1. fixed length time windows as used e.g. in [18, 27];
2. specific sequence metrics, e.g. operators or the edit distance [5, 16, 17, 28]; thereby, adaptation might be batch or online;
3. statistical modeling incorporating appropriate generative models for sequences such as proposed in [2, 32];
4. mapping of temporal dependencies to spatial correlation, e.g. as traveling wave signals or potentially trained temporally activated lateral interactions [4, 23, 35];
5. recurrent processing of the time signals and recurrent winner computation based on the current signal and previous activation [3, 6, 11, 12, 15, 29, 30, 33, 34]

These models have been tested in different application areas. Thereby, the success depends heavily on the model used in the respective application, and the principled suitability of the approaches is not yet understood. Therefore, an exact theoretical characterization of the capability of the choices would be highly desirable.

The Chomsky-hierarchy constitutes a well established specification of symbolic time-processing models. Interestingly, there exists an exact characterization of classical supervised recursive neural networks in terms of the Chomsky hierarchy which relates connectionist models to classical symbol-processing models: unrestricted supervised recurrent neural networks can simulate non-uniform Boolean circuits (but in exponential time) [24]. Weights restricted to rational numbers lead to Turing machines (with polynomial simulation delay) [25]. These results have been proved for the semilinear activation function. For the standard sigmoidal, Turing universality (in exponential time) has been shown in [14]. If the simulation is affected by noise, recurrent neural networks are equivalent to finite automata [19, 22], or even definite memory machines, if the support of the noise is unlimited (e.g. Gaussian noise) [20]. The same restriction applies to recursive neural networks with small weights [10].

The aim of this contribution is to give a systematic overview about the capacity of different unsupervised recursive neural networks in terms of classical symbolic models. Thereby, we restrict to the case where the processing of time dependent signals is realized by means of a recursive dynamics and a specific choice of the internal representation of temporal context. This setting includes the dynamics of standard supervised recursive networks and, in addition, a variety of popular unsupervised models including the temporal Kohonen map, recursive SOM, merge SOM, SOM for structured data, and, in a slight variation, also the recurrent SOM, SARDNET, and feedback SOM [3, 6, 11, 12, 15, 29, 30, 33, 34]. It turns out that, depending on the context choice, different capacities arise ranging from definite memory machines up to pushdown automata.

2 UNSUPERVISED RECURSIVE MODELS

Here, we introduce the general dynamic equation of unsupervised recursive models. The principled idea is to process a given time series step by step, starting from an initialization of the network, whereby in each step the output of the last computation step is also taken into account. Thereby, the models differ in the exact representation of the important information achieved in the last computation step. The respective relevant part is stored in a specific temporal context which differs according to the chosen model. A general mathematical formulation of the dynamics has been introduced in [9] as follows: Assume time series with entries $\mathbf{s}^t \in \mathbb{R}^n$ are considered. An unsupervised neural network is given by N neurons with weights $\mathbf{w}^i \in \mathbb{R}^n$ and context vectors $\mathbf{c}^i \in \mathbb{R}^r$ for each neuron i. In each step, a distance of the current entry of the time series from each neuron is computed, whereby the context is taken into account. This yields a vec-

[1] Clausthal University of Technology, Germany, email: hammer@in.tu-clausthal.de
[2] Institute for Neural Information Processing, Technical University Berlin, email: neubauer@cs.tu-berlin.de

tor of N activations $d_i(t)$ of neuron i at time step t. To formally define how this activation is computed, we need to fix a function d which computes the distance of time series entries and weights w^i, a function d_r which computes the distance of temporal contexts and context vectors of the neurons c^i, and a representation function $\mathrm{rep} : \mathbb{R}^N \to \mathbb{R}^r$ which extracts the temporal context of a computation ¿from the activation of all neurons. Then, the activation of a neuron i at time step t is defined as

$$\tilde{d}_i(t) = \alpha \cdot d(\mathbf{w}^i, \mathbf{s}^t) + \beta \cdot d_r(\mathbf{c}^i, C^t)$$

where

$$C^t = \mathrm{rep}\left(\tilde{d}_1(t-1), \ldots, \tilde{d}_N(t-1)\right)$$

extracts the relevant information from the activation of the previous time step. This formulation emphasizes the importance of an appropriate internal representation of complex signals by means of a context $\mathbf{c}^i$. Given a sequence s of length t, we denote by $d_i(s) = d_i(t)$ the result obtained after processing the whole sequence s.

Often, d and d_r are given by the euclidean norm, the L_1, or the maximum norm. This general formulation covers a variety of concrete neural models proposed in the literature. In particular, it includes supervised recursive networks, by taking d_r as dot product, and rep as sigmoidal function, as shown in [8]. The following concrete context choices constitute a representative coverage:

TEMPORAL KOHONEN MAP (TKM)

The temporal Kohonen map (TKM) [3] performs leaky integration of the distances of each neuron. The dynamics can be obtained by setting $r = N$, $\mathrm{rep} = \mathrm{id}$, d_r as the standard dot product, and $\mathbf{c}^i$ as the i'th unit vector, which realizes a 'focus' of neuron i on its own activation. The recurrent SOM [15] is similar in spirit, but it integrates vectors instead of distances and requires a vectorial quantity $\tilde{d}_i(t)$. In both cases, the internal context focuses on the neuron itself and it neglects the activation of all other neurons. This realization is very fast and it does not require additional memory for the context storage but, as we will see, it is quite restricted.

RECURSIVE SOM (RecSOM)

The recursive SOM (RecSOM) [34] chooses $r = N$. $\mathrm{rep}(x_1, \ldots, x_N) = (\exp(-x_1), \ldots, \exp(-x_N))$ is one-one, i.e. all information is kept. The feedback SOM is similar to RecSOM with respect to the context, however, the context integrates an additional leaky loop onto itself [11]. In both cases, the full information available in each step is taken as context, no compression or information extraction takes place. As a consequence, the model is quite demanding with respect to additional memory, which is proportional to N^2, N being the number of neurons.

SOM FOR STRUCTURED DATA (SOMSD)

The SOM for structured data (SOMSD) [6] is restricted to regular lattice structures. Denote by $L(i)$ the location of neuron i in a d-dimensional lattice. Then $r = d$ and $\mathrm{rep}(x_1, \ldots, x_n) = L(i_0)$ where i_0 is the index of the winner $\mathrm{argmin}_i\{x_i\}$. This context representation is only applicable to priorly fixed, though not necessarily euclidean lattices. SOMSD for a (fixed) hyperbolic lattice has been proposed in [30]. Compared to RecSOM, the memory requirements are much smaller.

MERGE SOM (MSOM)

MSOM is obtained for $r = n$, $d_r = d$, and rep as the merged content of context and weight of the winner in the previous step, i.e. $\mathrm{rep}(x_1, \ldots, x_N) = \alpha \cdot \mathbf{w}^i + \beta \cdot \mathbf{c}^i$ where $i = \mathrm{argmin}_i\{x_i \mid i = 1, \ldots, N\}$. It encodes the temporal context in the weight space by taking an appropriate average of the content of the winner represented by its weight $\mathbf{w}^i$ and its context vector $\mathbf{c}^i$. MSOM can be combined with arbitrary lattice structures.

All models are trained by Hebbian learning of the weights and (except for TKM) the context, whereby a lattice structure can be taken into account. This need not be a prior lattice, but data-optimum lattices such as the neural gas learning rule can be taken as well for all choices except SOMSD [18]. It has been shown in [8] that these learning rules can be interpreted as approximative truncated stochastic gradient descent of a cost function similar to the standard vector case. The training mode, however, is not relevant for this article where we are interested in the principled capacity of the models.

First mathematical results concerning the capacity of the models have already been shown in [7, 8].

3 SYMBOL PROCESSING CAPABILITIES

We will consider three symbol processing models of increasing capacity: definite memory machines, finite automata, and pushdown automata.

Definition 3.1 *Assume $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ is a finite alphabet of input symbols. Denote by Σ^* the set of finite sequences over Σ.*

A definite memory machine (DMM) *accepts a language in Σ^* if and only if there exists a finite memory length l such that for all sequences $s \in \Sigma^*$ holds: s is accepted if and only if $s|l$ is accepted where $s|l$ is the truncation of s to the last l symbols of the sequence.*

A finite state automaton (FSA) *consists of a triple*

$$(\Sigma, S, \delta)$$

whereby $S = \{\mathrm{sta}_1, \ldots, \mathrm{sta}_{|S|}\}$ is a set of states and $\delta : S \times \Sigma \to S$ is the transition function. The automaton accepts *a sequence $s = (\sigma_{i_1}, \ldots, \sigma_{i_t}) \in \Sigma^*$ iff $\delta^{\mathtt{it}}(s) = \mathrm{sta}_{|S|}$, whereby $\delta^{\mathtt{it}}$ is defined as the recursive application of the transition function δ to elements of S, starting from the initial state sta_1:*

$$\delta^{\mathtt{it}}(s) = \begin{cases} \mathrm{sta}_1 & \text{if } t = 0, \\ \delta(\delta^{\mathtt{it}}(\sigma_{i_1}, \ldots, \sigma_{i_{t-1}}), \sigma_{i_t}) & \text{if } t > 0\,. \end{cases}$$

A pushdown automaton (PDA) *is given by the 7-tuple*

$$(S, \Sigma, \Gamma, \delta, \mathrm{sta}, a, F)$$

where S is the set of states, Σ is a finite alphabet of input characters, Γ is a finite alphabet of stack characters,

$$\delta \subset (S \times (\Sigma \cup \epsilon) \times \Gamma) \times (S \times \Gamma^*)$$

is the transition relation mapping state, input and top stack element triples to successor state and stack replacement pairs, $\mathrm{sta} \in S$ *is the start state, $a \in \Gamma^*$ is the initial stack content, and $F \in \mathscr{P}(S)$ is the set of accepted final states. If δ is a function, the automaton is a deterministic push-down automaton (DPDA).*

We are interested in the capability of unsupervised models to simulate symbol processing models as defined above. For this purpose, we specify the following notation:

Definition 3.2 *Assume a language in Σ^* is accepted by a symbol processing model A. Then a recursive self-organizing map with input sequences over $\mathbb{R}^{|\Sigma|}$* simulates *A with constant delay D if the following holds: there exists a mapping $\mathsf{enc} : \Sigma \to (\mathbb{R}^{\mathsf{dimenc}})^D$ and a specified set of neurons with indices I such that for every sequence s in Σ^* holds:*

$$s \text{ is accepted by } A \iff \mathrm{argmin}_j\{d_j(\mathsf{enc}(s))\} \in I$$

whereby $\mathsf{enc}(s)$ denotes the component-wise application of enc to s.

This definition captures the intuition that a recursive unsupervised model which simulates a symbol processing dynamic distinguishes between accepted and not accepted words by the dynamics of a specified set of neuron. Since networks cannot directly deal with symbols, these must be encoded in a real-vector space. In addition, a linear delay in the computation time is allowed, which corresponds to an embedding of the symbols into a vector space of higher dimensionality, whereby D recursive steps are necessary to process an embedded symbol.

The context models of TKM, SOMSD, MSOM, and RecSOM differ considerably, the latter requiring more space and time for context representation. As we will see, this corresponds to different model capacities. It has been shown in [29] that the TKM cannot represent all finite automata in the above sense. However, we have the following result:

Theorem 3.3 *The TKM can simulate all definite memory machines.*

enc is chosen as unary encoding, i.e. $\mathrm{dimenc} = |\Sigma|$. Optimum weights for TKM, given a time series s, are $\mathbf{w}^{\mathrm{opt}(t)} = \sum_{i=0}^{t-1} \beta^i \mathbf{s}^{t-i} / \sum_{i=0}^{t-1} \beta^i$ [33] for $\beta = (1-\alpha)$. Assume the length of a definite memory machine is l. We can choose α close to 1 such that only the last l symbols of a series determine the winner. Furthermore, since the time series of length l are pairwise different, we can find α such that the optimum weights are pairwise different. A corresponding TKM simulates the given definite memory machine because it uniquely recognizes every sequence of length l. $\square$

Thus, the TKM is on the level of definite memory machines, i.e. it deals with only a finite time window. However, compared to an explicit time window, it uses less memory. It has been shown in [7, 29] that SOMSD and MSOM are strictly more powerful than TKM because they can simulate every finite automaton with fixed delay, whereby the L_1-norm is chosen as d and d_r. Obviously, since the internal states of SOMSD and MSOM constitute a finite set, both mechanisms can simulate at most finite automata. Thus, we find the following result:

Theorem 3.4 *The dynamics of SOMSD and MSOM are equivalent to finite automata.*

For RecSOM, the situation is difficult because of the quite complex context computation. On the one hand, an infinite reservoir is available because of real-valued context activations; on the other hand, however, information is very easily blurred because no focus in form of a winner computation takes place. The technical situation can be compared to the difficulties to investigate the capacity of supervised sigmoidal recurrent networks [14]. It has been shown in [31] that RecSOM with small weights implement at most definite memory machines i.e. RecSOM focuses on a finite time window in this case, similar to supervised recursive networks with small weights. The reverse direction (the capacity to simulate DMMs with small weights) is not obvious. Assume we drop the exponential function and take the identity as context representation (which is of the same quality with respect to its information content), then one can easily see that RecSOM with small weights can simulate definite memory machines: we can choose the representation of elements in Σ as small values, the context of the neurons according to the (unique) representations of all sequences of length l, and $(1-\alpha)$ small enough such that entries in previous steps do not change the winner. Similarly, for general weights and the original RecSOM context, the situation is not yet clear. Here we derive results for several simplified (though still reasonable) context models of RecSOM. Thereby, we only sketch the (rather lengthy) proofs, which can be found in [21].

3.1 FSA with RecSOM

Theorem 3.5 *Assume the context model of RecSOM has the form*

$$\mathrm{rep}(x_1,\dots,x_N) = \left(\frac{\exp(-x_1)}{\sum_i \exp(-x_i)},\dots,\frac{\exp(-x_N)}{\sum_i \exp(-x_i)}\right),$$

i.e. it is a normalized version of the original RecSOM context. Assume that the L_2^2 norm is chosen as d and d_r. Then, the model can simulate all finite automata.

Proof The general idea is taken from the FSA simulation with SOMSD[7]. There, two sets of neurons are introduced: The first containing one neuron for each state/input combination, the second containing one neuron for each state. The encoding function is then designed such that in a first processing step, one of the neurons from the first set wins, depending on the previous state sta_i (as indicated by the activity of the neurons from the second set) and the current input σ_j. In a second step, one of the neurons from the second set is chosen. The weights of these neurons can easily be adapted such that the neuron representing state sta_k wins iff $\mathrm{sta}_k = \delta(\mathrm{sta}_i, \sigma_j)$.

In the current case however, we have to deal with continuous context representations without explicit winner computation. This requires additional intermediate computations. The following defitions will help us address these issues more precisely:

Definition 3.6 *Let $f : \mathbb{R}^N \times \mathbb{R}^{\mathsf{dimenc}} \to \mathbb{R}^N$ be the function describing the map's behaviour at a single step, as defined above by the update rule for $\tilde{d}_i(t)$:*

$$f(\tilde{d}(t-1), \mathbf{s}^t) = \tilde{d}(t)$$

For a simulation with constant delay D, the result *of presenting σ to a map in context $\tilde{d}(t)$ is given by* $\mathrm{trans} : \mathbb{R}^N \times \Sigma \to \mathbb{R}^N$*:*

$$\mathrm{trans}(\tilde{d}(t), \sigma) = f(...f(f(\tilde{d}(t), \mathsf{enc}(\sigma)_1), \mathsf{enc}(\sigma)_2), \dots \mathsf{enc}(\sigma)_D)$$

This function describes the repeated application of f to itself and to the D components of $\mathsf{enc}(\sigma)$, and equals $\tilde{d}(t+D)$. One application of trans *corresponds to a single* transition cycle.

Definition 3.7 *Let enc be an encoding function as introduced above, dimenc $= 1$ and $\varepsilon > |\Sigma|$. enc is a* layering encoding function *iff*

$$\mathsf{enc}(\sigma_j) = (j, 1\cdot\varepsilon, \dots, (D-1)\cdot\varepsilon).$$

Note that only the first component of the encoding depends on the input σ_j. For a simulation associated with a layering encoding function, sensible input weights $\mathbf{w}^i$ are either $\{1,\dots,|\Sigma|\}$, the possible values of the first component, or $\varepsilon, \dots, (D-1)\varepsilon$, i.e., values of the following components.

Definition 3.8 *The* j-th layer *of a map associated with a layering encoding function* enc *is the set L_j of neurons such that $\mathbf{w}^i$ is a possible value of the j-th component of* enc *for all $i \in L_j$.*

Since $\varepsilon > |\Sigma|$, each neuron belongs to only one layer. Like this, a temporal structure is induced on the neurons: Each neuron is only activated during one particular step of each transition cycle [3].

So the example vaguely described above can now be seen as a two-layered construction. If we tried to simply adopt this strategy for RecSOM, we would see that the differences between the intended winner neuron's and other neurons' activations would soon become so similar that correct computation is no longer possible.

Definition 3.9 *Assume $\tilde{d} \in \mathbb{R}^n$. We call $\tilde{d}_k$ the* l-maximum in $\tilde{d}$ *iff*

$$\tilde{d}_k = \max\{\tilde{d}_1, \ldots, \tilde{d}_n\} \text{ and } \tilde{d}_k \geq l$$

For short, $\tilde{d}_k = \max_l(\tilde{d}),\ k = \operatorname{argmax}_l(\tilde{d})$

If $\tilde{d}$ is understood as the activation vector of a RecSOM and the kth component refers to the neuron representing a state sta*, we call $\tilde{d}$ a* l-representation of sta*, or short, $\tilde{d} \in \operatorname{staterep}_l(\mathrm{sta})$* .

Finally, we can formalize the need to maintain a certain quality of the state representation. We need to set up our neurons to ensure that

$$\exists l\ \forall \tilde{d}_i \in \operatorname{staterep}_l(\mathrm{sta}_i),\ \sigma_j \in \Sigma,\ \mathrm{sta}_k = \delta(\mathrm{sta}_i, \sigma_j) : \\ \mathrm{trans}(\tilde{d}_i, \sigma_j) \in \operatorname{staterep}_l(\mathrm{sta}_k)$$

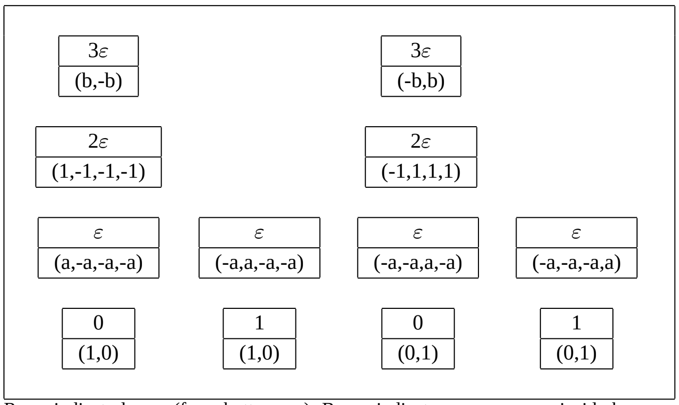
In order to create a sensible output of trans, the neurons in the last layer need a clear enough representation of the current input/state-tuple. The neurons representing these tuples, however, need a clear enough state representation from the last layer to begin with. It turns out that the required quality can be achieved by introducing a 'boosting' layer after each of the actual computation layers. These layers contain a boosting copy for each neuron from the previous layer, each tuned to the context component representing its predecessor with a boosting factor a and to the ones of the competing neurons of the previous layer with $-a$. The combination of a quadratic distance function and normalization causes this to amplify arbitrarily small differences in activation – for $a \to \inf$, this becomes a binary function. For sufficiently large boosting factors, each layer satisfies the entrance requirements of the following one – in particular, the output of the last layer satisfies the requirements of the first layer, for the next step, which means that the above constraints on trans are satisfied. □

Thus, the capacity of (a slightly modified) RecSOM includes at least finite automata. A further modification allows to prove even more:

3.2 Deterministic PDA with RecSOM

Theorem 3.10 *Assume the context model of RecSOM is modified to a winner-takes-almost-all context in the following way: $\mathrm{rep}(x_1, \ldots, x_n) = \mathtt{wta}(\mathtt{lin}(-x_1), \ldots, \mathtt{lin}(-x_N))$ where* wta *sets all but the maximum components to* 0*, the maximum components are copied identically, and* $\mathtt{lin}(x)$ *is the semilinear activation function which is* 1 *for $x = 0$, which is* 0 *for $x \geq 1$, and which is linear in between. Assume the L_1-norm is chosen. Then RecSOM can simulate all deterministic pushdown automata.*

[3] In fact, all neurons are slightly activated during each step in RecSOM - one of the omitted details of this proof is to give a lower bound for ε such that the activation always remains small enough for neurons from supposedly inactive layers



Rows indicate layers (from bottom up). Boxes indicate neurons; rows inside boxes indicate input weights and context weights. Note that only context weights referring to neurons from previous layer are displayed; all other context weights are 0, as the activation of these neurons is close to 0 as well.

a and b are boosting factors whose lower bounds grow in $O(N \log(N))$. Here, a> 57, b> 11100 for maintaining a .75-representation of the current state in the last layer and of the current state/input tuple in the second layer.

Figure 1. A simulation for a sample FSA 'No Ones' ($\Sigma = \{0, 1\}$)
$\delta(\mathrm{sta}, \sigma) = \mathrm{sta}_0$ if $(\mathrm{sta}, \sigma) = (\mathrm{sta}_0, 0), \delta(\mathrm{sta}, \sigma) = \mathrm{sta}_1$ otherwise

We show that RecSOM can simulate all determinstic pushdown automata with a stack alphabet of size 2 ($\Gamma = \{\gamma_0, \gamma_1\}$) – automata with larger stack automata can then be emulated.

3.2.1 *Encoding the stack*

The simulation, on the most general level, follows an approach similar to the previous one: Neurons are arranged in layers to be activated sequentially, from state × input representations to representations of the next state. However, the linear nature of the representation function here allows us to store and retrieve information in its value - we will use this to encode the current stack. Concretely, active neurons at any time step always have a representation of $1 - \frac{1}{4}\xi$, where ξ is a fractal encoding of the stack, as follows.

We first convert the string of stack symbols into a string of natural numbers (by replacing γ_i by their indices i), and then convert this sequence into a real number, following [26]:

Definition 3.11 $f_4(\epsilon) = 0 \quad f_4(\alpha) = \alpha|_4 = \sum_{i=1}^{|\alpha|} \frac{2\alpha_i + 1}{4^i}$

In the following, let ξ be the interpretation of a stack sequence in under f_4. It follows that $\xi \in [0, 1[$ and

empty(s) $\iff \xi = 0$
top(ξ)=γ_0 $\iff \xi \in [\frac{1}{4}, \frac{1}{2}[$, **top($\xi$)=$\gamma_1$** $\iff \xi \in [\frac{3}{4}, 1[$
push(ξ,γ_0) $= \frac{1}{4}\xi + \frac{1}{4}$, **push($\xi$,$\gamma_1$)** = $\frac{1}{4}\xi + \frac{3}{4}$
pop(ξ,γ_0) $= 4(\xi - \frac{1}{4})$, **pop(ξ,γ_1)** = $4(\xi - \frac{3}{4})$

This implies for the activation $1 - \frac{1}{4}\xi$ of the neurons:

- if the stack is empty, the winner's activation is 1
- if the stack is not empty, the winner's activation is from $]\frac{1}{2}, \frac{7}{8}]$
- after a push(γ_0) operation, the new state should have a representation of $1 - \frac{1}{4}\mathrm{push}(\xi, \gamma_0) = 1 - \frac{1}{4}(\frac{1}{4}\xi + \frac{1}{4}) = \frac{15}{16} - \frac{1}{16}\xi \in]\frac{7}{8}, \frac{15}{16}]$
- after a push(γ_1) operation, it should be $1 - \frac{1}{4}\mathrm{push}(\xi, \gamma_1) = 1 - \frac{1}{4}(\frac{1}{4}\xi + \frac{3}{4}) = \frac{13}{16} - \frac{1}{16}\xi \in]\frac{3}{4}, \frac{13}{16}]$
- after a pop(γ_0) operation, it should be $1 - \frac{1}{4}\mathrm{pop}(\xi, \gamma_0) = 1 - \frac{1}{4}(4(\xi - \frac{1}{4})) = \frac{5}{4} - \xi \in]\frac{3}{4}, 1]$
- after a pop(γ_1) operation, it should be $1 - \frac{1}{4}\mathrm{pop}(\xi, \gamma_1) = 1 - \frac{1}{4}(4(\xi - \frac{3}{4})) = \frac{7}{4} - \xi \in]\frac{3}{4}, 1]$

3.2.2 *Global structure*

So far, we have fixed that the information *which* neuron is activated represents the current state and/or a certain point in the computation, while *how much* this neuron is activated encodes the stack. For the proof, we hence need to ensure that for any state $\times$ stack $\times$ input combination, the neuron representing the correct successor state is activated with a value representing the correct, potentially modified stack.

For expressing this more formally, let us turn the classic definition of PDA into an equivalent more imperative one:

$$\delta : (S \times (\Sigma \cup \epsilon) \times \Gamma) \rightarrow (S \times U)$$

where U designates the possible stack updates:

$$U = \{\text{do nothing}, \text{push}(\gamma_0), \text{push}(\gamma_1), \text{pop}(\gamma_0), \text{pop}(\gamma_1)\},$$

or $U = \{\cdot, +_0, +_1, -_0, -_1\}$, for short. Note that success and failure are typically defined implicitly; here, we make them explicit as states and add the corresponding transitions – turning δ into a total function.

If we define $\text{staterep}(\text{sta}_\text{i}, \xi)$ as the context representation where the neuron representing sta_i is active, with an activation representing ξ, we can now express the condition for correct simulation as

$$\tilde{d}_i = \text{staterep}(\text{sta}_i, \xi) \rightarrow \text{trans}(\tilde{d}_i, \sigma_j) = \text{staterep}(\text{sta}_k, f(\xi, u)),$$

where $(\text{sta}_k, u) = \delta(\text{sta}_i, \sigma_j, \xi)$ and $f(\xi, u)$ the function applying stack update u to stack ξ. This is ensured in four different phases, i.e., functional assemblies of layers:

Separate makes the top stack element, so far implicit in the activation-encoded stack, explicit by exposing 'input/state/top element' neurons. That is, starting with the first layer of neurons encoding $S \times \Sigma$ – just like in the previous construction –, this phase creates a layer encoding $S \times \Sigma \times \Gamma$ in its neurons - the input space of the transition function.

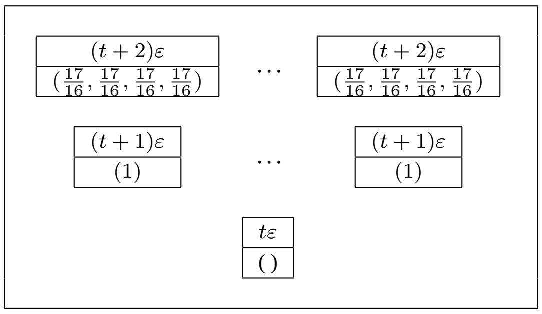
Merge groups these neurons by the output values of the input triples they represent, i.e., by the intended successor state and the stack update. So, starting from $S \times \Sigma \times \Gamma$, this phase ends with a layer encoding a subset of $S \times U$ – the image of δ.

Execute applies the stack update. This means it starts with the output layer of Merge, representing tuples from $S \times U$, and it ends with a layer representing these tuples, too – the activation, however, changed as to reflect the stack update.

Finalize makes sure there is only one neuron finally representing each possible successor state. If there are several δ output tuples (sta, u) with different u but same sta, the last phase has several neurons representing the new state sta. Here, all neurons representing sta are merged in a way that in the output layer, there is only one definite representation for each state sta – so this phase starts from $S \times U$ and ends in S.

3.2.3 *Operators*

So far, it may have remained rather vague how the introduced concepts relate to an actual map. We are now descending to the final layer of abstraction before getting to the level of actual neurons: Operators. As opposed to layers and phases (groups of layers), operators group neurons not only 'vertically' (over time, or processing steps), but also 'horizontally' (by function). They always consist of three layers with the third ('output') layer being the first ('input') layer of the next operator.



$(t+2)\varepsilon$
$(\frac{17}{16}, \frac{17}{16}, \frac{17}{16}, \frac{17}{16})$
$\dots$
$(t+2)\varepsilon$
$(\frac{17}{16}, \frac{17}{16}, \frac{17}{16}, \frac{17}{16})$
$(t+1)\varepsilon$
(1)
$\dots$
$(t+1)\varepsilon$
(1)
$t\varepsilon$
$()$

Note that the context weights of the input neuron are not relevant for the operator and hence left out. '…' indicates two additional identical neurons in that layer.

Figure 2. A Push$_{\gamma_0}$ operator

Top is an operator applied to each neuron of the first layer; it has three output neurons representing an empty, γ_0 or γ_1 top element.

Or is applied to two neurons – it ends in one neuron which is active if any of the two input neurons were active. This is used during the Merge phase, repeatedly applied to different neurons representing input triples equivalent under δ, and during the Finalize phase, applied to all neurons representing the same successor state.

Copy simply copies neurons' activities from layer to layer. If one set of input triples, e.g., is larger than another one during Merge, fewer operations are required for the smaller set – in this case, a number of Copy operators are inserted.

Push$_\gamma$ yields an output neuron with activation $\frac{15}{16} - \frac{1}{16}\xi | \frac{13}{16} - \frac{1}{16}\xi$ for $\gamma = \gamma_0 | \gamma_1$ if its input neuron was active with activation $1 - \frac{1}{4}\xi$.

Pop$_\gamma$ yields an output neuron with activation $\frac{5}{4} - \xi | \frac{7}{4} - \xi$ for $\gamma = \gamma_0 | \gamma_1$ if its input neuron was active with activation $1 - \frac{1}{4}\xi$. Push and Pop are applied during the Execute phase exclusively.

3.2.4 *Actual computation*

At this point, we have to describe how the functionality described above can be achieved, on the level of actual neurons.

The key question is how to divide and multiply the stack representation by four as needed for the push and pop operations – it is not clear how this should be achieved using the semilinear activation function.

Up to now, talking about 'the active neuron', we have, for ease of formulation, neglected the fact that `wta` actually allows for several active neurons, if they have the same, maximal activation.

Here lies the solution: We can copy the activation from a neuron i in one layer to another neuron o in the next layer by tuning o's context weight concerning i to 1 (assuming $\beta = 1$). If we have two identical neurons i_1 and i_2, the distance is doubled if o is tuned to both. Accordingly, if we divide the number of representing neurons i, distance is divided. In order to be able to divide, we set the standard number of identical representing neurons to 4, by setting $\beta = \frac{1}{4}$. So having four identical i creates an exact copy in o, while a single i results in a division by four, and two layers of eight i neurons result in multiplication by four. See figure 2 for a drafted Push operator.

It then has to be shown that

1. for each operator, the desired output values are always created, and
2. of different operators in the same layer, it is always the correct one that wins, i.e. the unique one having active input neurons.

	TKM	MSOM	SOMSD	RecSOM
DMM	yes	yes	yes	yes (small weights)
FSA	no	yes	yes	yes (normalized context)
PDA	no	no	no	yes (winner-takes-almost-all)

Figure 3. Summary of the capacity of the different context models.

Once these conditions are satisfied – as they are, `lin` contributing to the first, `wta` to the second one –, the global structure can be shown to work as intended and the simulation turns out to be correct. □

Thus, the overall picture as shown in Fig. 3 arises. Thereby, the picture concerns the principled representational capability. It is not clear whether standard learning schemes such as Hebbian learning lead to these categories, given sample data. First steps into the investigation of the attractor of concrete learning algorithms can be found in [7].

4 CONCLUSION

Unsupervised recursive networks constitute a promising self-organizing learning scheme which can account for the topographic organization of temporal signals in neural maps. Whereas supervised recursive networks are quite well understood, a variety of different plausible unsupervised recursive models exists.

In this contribution, a systematic link of unsupervised recursive neural models and classical symbol-processing mechanisms has been investigated. Interestingly, the different context choices lead to quite different capacities ranging from definite memory machines up to pushdown automata. The latter are of particular interest when exploring language learning since finite state mechanisms combined with embedded constructions (e.g. embedded sentences) are covered this way.

REFERENCES

[1] Barreto, G., Araújo, A., and Kremer, S.C. (2003). A Taxonomy for Spatiotemporal Connectionist Networks Revisited: The Unsupervised Case. *Neural Computation*, 15(6):1255-1320.

[2] Bishop, C.M., Hinton, G.E., and Strachan, I.G.D. (1997a). *Proceedings IEE Fifth International Conference on Artificial Neural Networks*, Cambridge, U.K., 111-116.

[3] Chappell, G. and Taylor, J. (1993). The temporal Kohonen map. *Neural Networks* 6:441-445.

[4] Euliano, N.R. and Principe, J.C. (1999). A spatiotemporal memory based on SOMs with activity diffusion. In E.Oja and S.Kaski (eds.), *Kohonen Maps*, Elsevier.

[5] Günter, S. and Bunke, H. (2002). Self-organizing map for clustering in the graph domain. *Pattern Recognition Letters*, 23:401–417.

[6] Hagenbuchner, M., Sperduti, A., and Tsoi, A.C. (2003). A Self-Organizing Map for Adaptive Processing of Structured Data. *IEEE Transactions on Neural Networks* 14:191-505.

[7] Hammer, B., Micheli, A., Sperduti, A., Strickert, M. (2004). Recursive self-organizing network models. *Neural Networks* 17(8-9), 1061-1086.

[8] Hammer, B., Micheli, A., Sperduti, A., and Strickert, M. (2004). A general framework for unsupervised processing of structured data. *Neurocomputing* 57, 3-35.

[9] Hammer, B., Micheli, A., and Sperduti, A. (2002). A general framework for unsupervised processing of structured data. In M.Verleysen (ed.), *European Symposium on Artificial Neural Networks*, pages 389-394, D-side publications.

[10] Hammer, B. and Tino, P. (2003). Recurrent neural networks with small weights implement definite memory machines. Neural Computation 15(8), 1897-1929.

[11] Horio, K. and Yamakawa, T. (2001). Feedback self-organizing map and its application to spatio-temporal pattern classification. *International Journal of Computational Intelligence and Applications* 1(1):1-18.

[12] James, D.L. and Miikkulainen, R. (1995). SARDNET: a self-organizing feature map for sequences. In G.Tesauro, D.Touretzky, and T.Leen (eds.), *Advances in Neural Information Processing Systems 7*, pages 577-584, MIT Press.

[13] Kaski, S., Kangas, J., and Kohonen, T. (1998). Bibliography of self-organizing maps, papers: 1981-1997. *Neural Computing Surveys*. 1, 102-350.

[14] Kilian, J. and Siegelmann, H.T. (1996). The dynamic universality of sigmoidal neural networks. *Information and Computation*, 128.

[15] Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998). Time Series Prediction using Recurrent SOM with Local Linear Models. *Int. J. of Knowledge-Based Intelligent Engineering Systems* 2(1): 60-68.

[16] Kohonen, T. (1997). *Self-organizing maps*. Springer.

[17] T. Kohonen and P. Somervuo (2002), How to make large self-organizing maps for nonvectorial data, *Neural Networks* **15:**945-952.

[18] Martinetz, T., Berkovich, S., and Schulten, K. (1993). "Neural-gas" network for vector quantization and its application to time-series prediction. *IEEE-Transactions on Neural Networks* 4(4): 558-569.

[19] W. Maass and P. Orponen. On the effect of analog noise in discrete-time analog computation. *Neural Computation*, 10(5), 1998.

[20] W. Maass and E. D. Sontag. Analog neural nets with Gaussian or other common noise distributions cannot recognize arbitrary regular languages. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, Volume 9. The MIT Press, 1998.

[21] Neubauer, N. (2005). Recursive SOMs and Automata. M.Sc. Thesis, Cognitive Science, University of Osnabrück.

[22] Omlin, C.W., and Giles, C.L. (1996). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 43(6):937-972.

[23] Schulz, R. and Reggia, J.A. (2004). Temporally asymmetric learning supports sequence processing in multi-winner self-organizing maps. *Neural Computation* 16(3): 535–561.

[24] H. T. Siegelmann and E. D. Sontag. Analog computation, neural networks, and circuits. *Theoretical Computer Science*, 131, 1994.

[25] H. T. Siegelmann and E. D. Sontag. On the computational power of neural networks. *Journal of Computer and System Sciences*, 50, 1995.

[26] H. T. Siegelmann *Neural Networks and Computation: Beyond the Turing Limit* Progress in Theoretical Computer Science. Birkhäuser, Boston, 1999.

[27] Simon, G., Lendasse, A., Cottrell, M., Fort, J.-C., and Verleysen, M. (2003). Double SOM for Long-term Time Series Prediction *WSOM 2003, Workshop on Self-Organizing Maps*, Hibikino (Japan), 11-14 September 2003, pp. 35-40.

[28] Somervuo, P.J. (2004). Online algorithm for the self-organizing map of symbol strings. *Neural Networks*, 17(8-9):1231-1240.

[29] Strickert, M. and Hammer, B. (2005). Merge SOM for temporal data. *Neurocomputing* 64:39-72, 2005

[30] Strickert, M., Hammer, B., and Blohm, S. (2005). Unsupervised recursive sequences processing. *Neurocomputing* 63, 69-98, 2005.

[31] Tino, P., Farkas, I., and van Mourik, J. (2005). Topographic Organization of Receptive Fields in Recursive Self-Organizing Map. Technical Report CSRP-05-06, School of Computer Science, University of Birmingham, UK.

[32] Tino, T., Kaban, A., and Sun, Y. (2004). A Generative Probabilistic Approach to Visualizing Sets of Symbolic Sequences. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - (KDD-2004)*, R. Kohavi, J. Gehrke, W. DuMouchel, J. Ghosh (eds). pp. 701-706, ACM Press.

[33] Varsta, M., Heikkonen, J., Lampinen, J., and Milán, J.del R. (2001). Temporal Kohonen map and recurrent self-organizing map: analytical and experimental comparison. *Neural Processing Letters*, 13(3):237-251.

[34] Voegtlin, T. (2002). Recursive self-organizing maps. *Neural Networks* 15(8-9):979-992.

[35] Wiemer, J.C. (2003). The time-organized map algorithm: extending the self-organizing map to spatiotemporal signals. *Neural Computation* 15(5): 1143–1171.

Position Paper: Towards a dynamic assessment of formal language complexity

André Grüning[1]

Abstract. We review two concepts that have been suggested to assess the complexity of a formal language from the point of view of a recurrent neural network. While well-defined concepts of complexity exist for symbolic computations, it is not clear how they relate to the dynamic complexity of a recurrent network that represents a particular formal language in the from of a dynamical recogniser. Thus intrinsic dynamical concepts of symbolic complexity are needed.

1 INTRODUCTION

Artificial neural systems, be it in the guise of simple recurrent networks, synfire chains or associative networks, have in common that they all are essentially dynamical systems. When we speak about neuro-symbolic interaction, we thus mean the interaction between symbolic systems and dynamic systems. Since the basic properties of symbolic and dynamical systems are quite different – symbolic systems for example have i. a discrete state space, ii. a single powerful finite control, and iii. restricted memory resources as the principal restricted resources, while dynamical systems have i. a bounded continuous state space, ii. many simple elementary processors, and iii. restricted numeric precision as the principal restricted resource, it can be difficult to see how a system of the one kind ought to be interpreted in terms of the other.

For symbolic systems there exists a series of measures of complexity, e. g. i. the Chomsky hierarchy [5] that classifies formal languages according to the type of rewrite rules; ii. computational complexity theory that takes into account resource limits [10]; and iii. finally Kolmogorov or algorithmic complexity that formalises the concept of a shortest programme [15].

In many cases input and output of dynamical systems need to be interpreted symbolically and it is not clear how the symbolic complexity of this input and output relates to the dynamic complexity of the dynamical system in between. For example, it has been noted that the – in symbolic computation – big step between context-free and context-sensitive processes is only a tiny one for a dynamical system [14, 20].

It has also been noted that even though recurrent networks are in principle Turing-equivalent [9, 19] by explicit construction, a trained dynamical representation of a formal language differs from hand-coded symbol-inspired implementations [24], the latter seemingly 'unnatural' for a network.

Thus we are interested in the possibilities of an intrinsic dynamic approach towards the complexity of formal languages based on *dynamical recognisers* [18, 17]. A dynamical recogniser (DR) (in the abstract sense) consists of the following (leaving out technical details): i. an input alphabet A, ii. a bounded continuous state space X, iii. a system of iterated functions, such that for each $i \in A$ there is one $F_i : X \mapsto X$, iv. an *accept region* $S \subset X$, and v. a start state $x_0 \in X$. A DR processes a language L over the alphabet A as follows: It starts in the start state x_0, then a string over A is input symbol by symbol applying the corresponding maps F_i on the current state x. The string is accepted when x lies in S after the last input.

The concrete example of a DR we have in mind is of course a recurrent network, for which each input symbol causes an update of the current state, and whose accept region is defined by those states that will lead to an output activation above a certain threshold.

[1] Cognitive Neuroscience Sector, International School of Advanced Studies (ISAS) / Scuola Internazionale Superiore di Studi Avanzati (SISSA), Via Beirut 2–4, I-34014 Trieste, Italy, email: gruening@sissa.it

2 THE CONSTRUCTION BASED APPROACH

Moore [17] considers DRs that stem from neural networks for which only certain function classes are allowed as the neurons' activation functions, i. e. linear, piece-wise linear, (piece-wise) polynomial or transcendent functions etc. DR so constructed from different function classes accept different formal languages and thus define a classification of formal languages. These classes do not agree with the Chomsky hierarchy.

Moore balances the power of the state space dynamics and the shape of the accept region in that the functions defining them stem from the same class. Else one could have fairly trivial dynamics and put the computation completely in the map that defines the accept region.

3 THE WEIGHT BASED APPROACH

Tabor [20] introduces a concept of similarity of formal languages in the following way: given a network that processes a formal language, it is essentially defined by its weight matrix (given the functional classes of the activation functions are fixed). Slight variations of the weights will lead to different DRs processing different languages. A measure for the similarity of two languages can then be based on the (Euclidian) distance of the corresponding weight matrices. Tabor gives an example of a one-parameter family of dynamical recognisers that accepts a context-free language for a rational weight and a non-context-free language for an irrational one. Thus there are non-context-free languages in any neighbourhood of a context-free language. Accordingly, this concept of similarity cuts across the Chomsky hierarchy, too.

4 DRAWBACKS

While Moore's dynamical recogniser language classes are technically and mathematically sound, they are based on *external* properties of the functions that a dynamical system is constructed of, not on dynamical invariants of a dynamical system itself, which would yield a more intrinsic concept of the dynamics and its possible symbolic interpretation.

In Tabor's approach, completely different weight matrices may give rise to DRs with the same language. Thus proximity of weight matrices defines proximity of languages. But determining whether two language possess similar representations in DRs seems to be difficult. It would be useful to have a standard or canonical dynamical implementation of a formal language in a DR or neural network. Furthermore Tabor overlooks the possibility that slight weight changes can lead to completely different dynamics (bifurcations) at critical points and thus also to completely different languages.

5 NEW DIRECTIONS

We do not have complete solutions to the challenges put forward by the above approaches, but we want to lay out what is desirable and thus what future roads of research could be. We would like to base a dynamical classification of formal language on intrinsic invariants of dynamic systems.

Entropies have been used to classify dynamical systems [7, 6]. However these are much too coarse-grained notions of similarity and do not agree with an intuitive notion of similarity of dynamical systems [11]. However recently there have been developments in dynamical systems theory that make use of fixed points, periodic points, attractors etc. (in short: invariant sets) and the limiting trajectories between them in a systematic fashion in order to categorise dynamical systems [12, 13, 16]. This approach avoids the use of external properties of dynamical systems unlike Moore's, and it also abstracts away from concrete implementations: dynamical systems are similar when they have similar attractor structures. To our knowledge the theory has so far been developed for autonomous (non-input driven) dynamical systems (there is generally little systematic literature on iterated function *systems* that are externally driven [1, 21, 22]). What it does tell us though, is that we should focus on fixed and periodic points or more complicated invariant sets as the important building blocks of a dynamical system that are entangled with a system's computational capabilities.

In the case of regular languages, it is granted that a closed loop in the finite state automaton (FSA) corresponds to an invariant set of the corresponding combinations of iterated DR maps [4]. Since for non-regular languages a processing system has to keep track of an infinite number of states, there will be an infinite number of invariant sets for combinations of the iterated maps F_i that define the system. For a DR language $a^n b^n$ it is e. g. necessary that each combination $F_b^m \circ F_a^m$ be the identity map on all states representing a sequence of a inputs. Furthermore all known trained or evolved network implementations of $a^n b^n$ use the distance to a (pair of) fixed points to encode the number of a inputs [2, 3, 25]. For a stack-like language ww^r where w is an arbitrary string and w^r its mirror-reverse, a dynamical system forms much more complicated attractors (of fractal shape) that nevertheless have to be invariant under certain combinations of the input maps F_i [8].

These observations seem promising starting points for an intrinsic theory of dynamical complexity. We hope to have convinced the reader that it is worth to think along these lines.

References

[1] Michael F. Barnsley, *Fractals Everywhere*, Academic Press Professional, Cambridge, MA, second edn., 1993.

[2] Mikael Bodén and Alan Blair, 'Learning the dynamics of embedded clauses', *Applied Intelligence*, **19**(1/2), 51–63, (2003).

[3] Mikael Bodén, Henrik Jacobsson, and Tom Ziemke, 'Evolving context-free language predictors', in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1033–1040. GECCO, (2000).

[4] Mike Casey, 'The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction', *Neural Computation*, **8**, 1135–1178, (1996).

[5] Noam Chomsky, 'Three models for the description of language', *IRE Trans. on Information Theory*, **2**(3), 113–124, (1956).

[6] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.

[7] Robert L. Devaney, *An Introduction to Chaotic Dynamical Systems*, Addison-Wesley, Menlo Park, Ca., 1989.

[8] André Grüning, 'Stack- and queue-like dynamics in recurrent neural networks', *Connection Science*, **18**(1), (2006).

[9] Barbara Hammer, 'On the generalization of Elman networks', in *International Conference on Artificial Neural Networks '97*, eds., W. Gerstner, A. Germond, and M. Hasler, pp. 404–414, Berlin, (1997). Springer.

[10] John E. Hopcroft and Jerrey D. Ullmann, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Mass., 1979.

[11] Jürgen Jost, 'On the notion of complexity', *Theory Biosci.*, **117**, 161–171, (1998).

[12] Jürgen Jost, *Dynamical Systems – Examples of complex Behavior*, Springer, 2005.

[13] T. Kaczynski, K. Mischaikow, and M. Mrozek, *Computational Homology*, Applied Mathematical Sciences 157, Springer-Verlag, 2004.

[14] John F. Kolen, 'Fool's gold: Extracting finite state machines from recurrent network dynamics', in *Proceedings of NIPS*, (1993).

[15] Ming Li and Paul Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer, Berlin, 1997.

[16] K. Mischaikow and M. Mrozek, 'Conley index theory', in *Handbook of Dynamical Systems II: Towards Applications*, ed., B. Fiedler, North-Holland, (2002).

[17] Cristopher Moore, 'Dynamical recognizers: Real-time language recognition by analog computers', *Theoretical Computer Science*, **201**, 99–136, (1998).

[18] Jordan B. Pollack, 'The induction of dynamical recognizers', *Machine Learning*, **7**, 227–252, (1991).

[19] Hava T. Siegelmann and Eduardo D. Sontag, 'On the computational power of neural nets', *Journal of Computer and System Sciences*, **50**, 132–150, (February 1995).

[20] Whitney Tabor, 'Fractal encoding of context free grammars in connectionist networks', *Expert Systems*, **17**(1), 41–56, (2000).

[21] Peter Tiňo and Barbara Hammer, 'Architectural bias in recurrent neural networks – fractal analysis', *Neural Computation*, **15**, 1931–1957, (August 2003).

[22] Peter Tiňo, Michal Čerňanský, and Ľubica Beňušková, 'Markovian architectural bias of recurrent neural networks', *IEEE Transactions on Neural Networks*, **15**(1), 6–15, (2004).

[23] Ingmar Visser, *Rules and Associations*, Ph.D. dissertation, Afdeling Psychologie, Faculteit Maatschappij- en Gedragswetenschappen, Universiteit van Amsterdam, 2002.

[24] Ingmar Visser, Maartje E. J. Raijmakers, and Peter C. M. Molenaar, 'On the computational capabilities of symbolic and subsymbolic models', *Submitted*, (2004). See also chapter 2 in [23].

[25] Janet Wiles and Jeff Elman, 'Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks', in *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, pp. 482–487, Cambrigde, Mass., (1995). MIT Press.

An Approach to Language Understanding and Contextual Disambiguation in Human-Robot Interaction

Heiner Markert and **Günther Palm** [1]

Abstract. An approach to language understanding should be able to handle ambiguities to a certain degree at all levels of processing. We present a system based on several interacting associative memories that is able to understand simple statements or instructions and to represent and resolve ambiguities at different levels. For example, ambiguous input on the phoneme level leading to an ambiguous word representation might be resolved later using contextual information from the whole sentence or even from the whole sensory-motor situation. The system is implemented on a robot that is able to react to simple command sentences like "bot show plum". This requires the robot to relate auditory sensory information to internal representations of the corresponding words, to associate the given sequence of words with the complete command sentence and to relate the sentence representation to objects sensed by the camera and to motor actions required to point to the object. The system uses basic neural mechanisms in a plausible global network architecture that is formulated essentially in terms of cortical modules and their intracortical and corticocortical interconnections. The modules represent and translate between different aspects of the same entities, e.g. auditory, syntactical and semantical aspects of words or visual, auditory and grasping related aspects of objects to achieve the required functionality. Presently, the system can handle a few types of simple sentences and a small vocabulary, but grammar and vocabulary can be extended easily.

1 INTRODUCTION

We created a neurobiologically plausible neural network architecture for understanding simple language and performing corresponding actions on a robot. The network consists of a large number of interconnected modules, each containing a network of spiking neurons. The modules represent different aspects of objects, e.g. sensory, visual, auditory, motor, syntactical or semantical aspects, while the connections between the modules translate between the different aspects. The network architecture is motivated by the idea of distributed cell assemblies and associative memories [6, 3, 14].

The network is able to understand and to react to simple command sentences. In order to demonstrate the functionality, we embedded it into a simple robot scenario (see figure 1): A robot stands in front of a white table. It receives spoken commands like "bot show plum" and has to react accordingly. This involves understanding of the sentence and analysing it with respect to a given grammar, extracting the meaning of the command, relating the visual input from the camera to object words in the sentence and finally triggering the right motor commands to perform the requested action (see e.g. [5] for details).

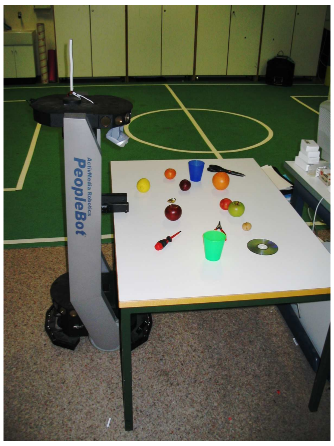


Figure 1. The robot standing in front of a white table. Different objects are laying on the table. The robot has to grasp or point to objects specified in simple command sentences.

In this paper, we focus on the language processing part of the system. The current implementation does not yet feature a complete speech recognition system. Instead, the network gets input in form of written phonetic pair representations, which could be generated by standard Hidden Markov Model based speech recognition software. The model is able to extract words from an input stream of phonetic pairs and to grammatically analyse the resulting stream of words with respect to a given regular grammar. The language model is able to represent ambiguities on the single word level and to resolve them even some time after the ambiguous input arrived in the system: If it is not possible to interpret a word or a given sequence of phonetic pairs in a unique way, several alternatives can be kept until enough context information from a broader context arrives to resolve the ambiguity. For example, the sentence "bot lift bwall" with an ambiguity between "ball" and "wall" can be resolved to "bot lift ball" because a wall is not liftable. Here, the ambiguous input could have been caused by an unclear pronunciation of the word "ball" that confused the speech recognition unit.

2 NETWORK ARCHITECTURE

Figure 2 gives an overview of the architecture of the network used for language understanding. Each box in the figure corresponds to one cortical module and contains an auto-associative memory (see section 3). The modules are interconnected to each other with Hebbian learned binary hetero-associative connections.

[1] University of Ulm, Germany, email: {hmarkert,palm}@neuro.informatik.uni-ulm.de

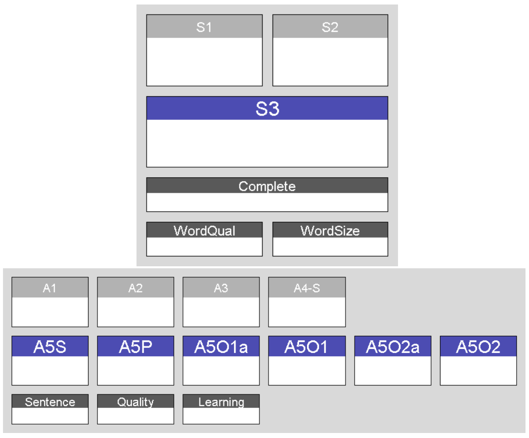


Figure 2. The network architecture of the language model. Each box corresponds to a module of the network consisting of a spiking auto-associative network. The top of the figure shows the word recognition part consisting of three cortical modules (S1-S3) and three control modules that give additional status information. The bottom part shows the language understanding system which analyses the stream of words from the word recognition network with respect to a regular grammar. It consists of 10 cortical modules (A1-A4, A5X) and two control modules. There are additional thalamic activation modules in the whole network where only one of them, the field "Learning" in the language understanding part, is shown.

The network consists of two parts, the word recognition network (top of figure 2) and the language understanding network (bottom of figure 2). The word recognition part, consisting of modules S1-S3 and several status modules, translates a sequence of phonetic pairs into corresponding words. Area S1 receives single phonetic pairs e.g. from a speech recognition software. Area S2 again represents phonetic pairs, but in contrast to S1 it is able to activate a superposition of approximately 10 pairs. S2 is used to remember all phonetic pairs that the system heard in the near past. S2 projects into S3, which stores all words known by the system. The word that matches best to the given phonetic pairs is activated (or a superposition, if more than one word matches closely). There are additional control modules that inhibit S2 and S3 if silence is heard (a new word starts) and measure the quality of the word representation.

The language part, consisting of modules A1-A4, A5X and several status modules, then analyses the output sequence of words from the word recognition part with respect to a regular grammar. The modules A1-A3 serve as auditory input areas and represent auditory, syntactical and semantical aspects of the input word.

The modules A4 and A5X mainly serve grammatical functions, where A4 works as a sequence memory and represents the grammar the system is able to understand. Areas A5X store the words with respect to their grammatical role, i.e. they classify into subject, predicate, attributes and objects of the sentence. Here, A5S holds the subject, A5P the predicate, A5O1a an attribute, A5O1 the first object, A5O2a an attribute to the second object and A5O2 the second object of the sentence.

There are additional control modules in the language model that give miscellaneous status information on the model. For a more detailed description of the model see e.g. [8, 11].

The system is able to understand regular grammars. The current implementation allows only for very simple sentence types, namely "subject predicate object" sentences (SPO) and "subject predicate adjective" sentences (SPA) and different versions of them. The following sentences are all valid and can be correctly understood by the system:

"bot show plum", "bot show green apple", "where is plum?","this is plum", "this is green","wall is red","bot put apple plum","bot put red plum yellow lemon".

3 NEURAL ASSOCIATIVE MEMORY

Each module in our network is modelled with a variant of the so called spike counter model of associative memory (see [9, 7]) which is based on Willshaw's binary associative memory [21]. The spike counter model extends Willshaw's model by a more sophisticated retrieval algorithm that allows in particular for much better pattern separation if the memory is addressed with a superposition of patterns. Further extensions allow for automatic activation of a superposition of patterns if the input is not uniquely addressing one of the stored patterns and the memory is configured to allow ambiguous representations (see section 4 for details). The spike counter model uses spiking neurons and allows to measure spike time coincidences. We have chosen Willshaw's model as a basic system because it is a biologically plausible while still simple implementation of the idea of cell assemblies.

In this paper, we use a rather technical implementation of the model which still allows for fine measurement of spike timing and especially of the temporal order of the spikes. The neurons are of a simple integrate-and-fire type with reset. We further simplify calculations by introducing global time steps that roughly correspond to one time step within the binary Willshaw model. The global steps are subdivided by a finer relative time scale that allows for exact representation of spike times. In one global time step, each module calculates one complete pattern retrieval with respect to the relative time scale. After the global step ended, the output activity of all modules is propagated through the hetero-associative connections between the modules, before the next global time step starts.

In one retrieval step, each neuron counts to how many addressing neurons it is connected (amount of input it receives) and this influences the speed of growth of the neuron's membrane potential. During the retrieval, if a neuron spikes, the spike is fed back through auto-associative connections and the membrane potential of the auto-associatively connected neurons start growing faster, supporting the pattern that has already started to become active. In addition to that, neurons that are not auto-associatively connected to the neurons that have already spiked are inhibited, which is realised by additional spike counter variables. For more details, see [10, 7]. Each neuron is allowed to spike at most twice per global step and the retrieval ends as soon as no more neuron is able to spike. The algorithm necessarily terminates, because at some point, either all neurons have already emitted two spikes or the remaining ones are inhibited so strongly that they are not going to spike anymore (i.e. they receive no more input in the current global step).

For demonstration purposes we use a special way of displaying neural activity in the system (see figure 3). Instead of showing neural activation directly, we display names of the patterns that match best. In our architecture, all patterns that are stored in the associative memories have names. After each global time step, the result of the pattern retrieval is determined as shown in figure 3: In a first step, a histogram that measures the overlap of the current neural activation with each stored pattern is calculated. Then, the names of all patterns that have a large enough overlap are displayed in the box representing the cortical module. There is a maximum number of patterns displayed so that displaying superpositions of many patterns does not

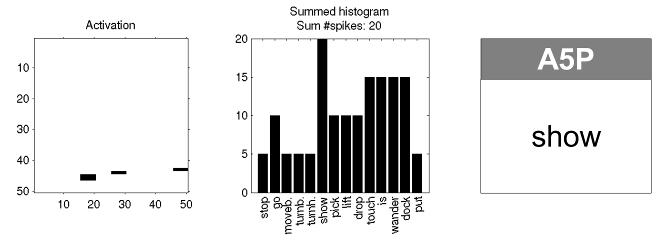


Figure 3. From neural activation to pattern name display: In the left part, an example neural activation pattern of one cortical module is shown. In the middle, the overlap histogram over all stored patterns is calculated. Note that each pattern has a name in our architecture. From the histogram, the name(s) of the best matching patterns are determined and used for displaying the neural activation in the module (right).

require too much processing time and space on the display.

4 DISAMBIGUATION

The model is able to represent and resolve ambiguous inputs, e.g. the sentence "bot lift bwall" with an ambiguity between "ball" and "wall" can be correctly understood as "bot lift ball", because a wall is not a liftable object. The model is also capable of representing ambiguities until contextual information arriving later can be used to resolve it. For example, the sentence "bot show/lift green wall" with an artificial ambiguity between "show" and "lift" is correctly understood as "bot show green wall" as soon as "wall" comes in, because a wall is not liftable. Before "wall" is heard, a superposition of "lift" and "show" is kept in the corresponding module in the language model (A5P).

In the following we will explain shortly how representing and resolving ambiguities works. For more details on the neuron model see also [10].

Each cortical module has a special parameter α, the so called separation strength. When this parameter gets lower, the module allows for superposition of several patterns, the lower the parameter is, the more patterns can become active at the same time. If α is in a medium range, only patterns which overlap can be active concurrently, while even lower separation strength allows for arbitrary superpositions of patterns. High separation strength forces the module to activate one pattern at most and leads to a hard decision if the input is not unique (it is a random decision if more than one pattern are addressed equally strong and α is really high).

Furthermore, each module can measure the quality of the address pattern, where the quality becomes high when exactly one pattern is addressed strongly, while it becomes lower if several patterns are addressed equally strong. The exact mechanisms for determining the quality measure is quite complex. Roughly speaking, the module performs a control retrieval with high separation strength to find out which pattern is addressed strongest and compares the results with an additional retrieval with the actual value of the separation strength. In this two retrievals, the neurons are allowed to spike up to two times each, and the second retrieval is cancelled shortly after the first neuron emitted its second spike. The idea is the following: When exactly one pattern is addressed strongly, the whole pattern will spike twice before any other neuron, that is only driven by noise, can emit a second spike. If however a superposition of patterns is addressed, the overlap spikes twice before the other neurons can do so because it gets the strongest input activation. If two disjoint patterns are addressed almost equally strong and the maximum value of the separation strength is not too high, a few neurons from both patterns will spike in the second retrieval, leading to a high activation, which in turn decreases the quality measure.

With a feedback connection from the quality measure to the separation strength parameter, the modules become able to automatically decide whether they have to activate a superposition of several patterns or if they can switch to a higher separation strength: If the address pattern does not match a stored pattern uniquely, quality becomes lower, separation strength is decreased and a superposition becomes active. If on the other hand the separation strength is already low, but the address matches precisely one of the stored patterns, the activity in the second retrieval will be decreased, leading to a higher quality measure which in turn increases separation strength. This in turn will lead to even lower activity in the next retrieval, until separation strength becomes high enough to decide for the addressed pattern and suppress the noise.

5 RESULTS

We have implemented the language understanding system on a ActivMedia PeopleBot robot platform. While the robotics part of the software (object detection, motor control) runs on the PeopleBot's onboard-PC, the neural network for language understanding runs on a separate PC and communicates with the robot via wireless LAN. The current network involves about 26000 neurons in total and has a

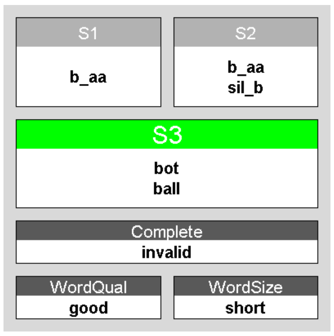


Figure 4. The word recognition system after half of the word "bot" is processed. The already processed phonetic pairs are activated as a superposition in module S2. S3 already has some suggestions for matching words active.

vocabulary of about 50 words. On a standard laptop machine (Pentium 4M, 1.5GHz), the network needs 5 seconds to process the sentence "bot show apple", missing real time constraints by a factor of about 5. Note that the language understanding part meets real time constraints and can grammatically analyse a sentence faster than one is able to speak it. In the word recognition part, however, each phonetic pair currently requires three global processing steps which slows down the system. Note that the word recognition part is work in progress and we believe that a more efficient implementation on a faster PC would suffice to meet real time constraints for the whole architecture.

In the following, we will show in detail how the system processes the sentence "bot lift bwall", where the ambiguity between "ball" and "wall" is already present in the phonetic inputs, i.e. we assume

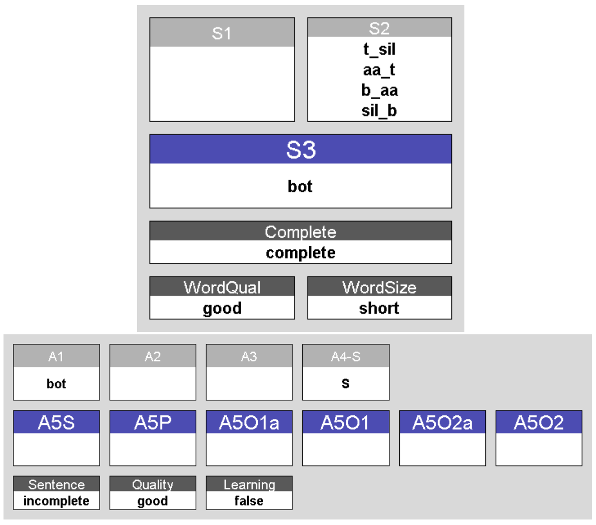


Figure 5. **Top:** The word recognition system after the word "bot" is completely processed. S3 has a unique assembly activated, the "Complete"-field has recognised that a complete word is understood and the word quality is good, meaning that it was possible to uniquely decide for a word that matches well with the list of phonetic pairs in S2. **Bottom:** The language understanding system at the same time. The "bot" pattern has already been transfered to module A1, the first input module in the sentence layer.

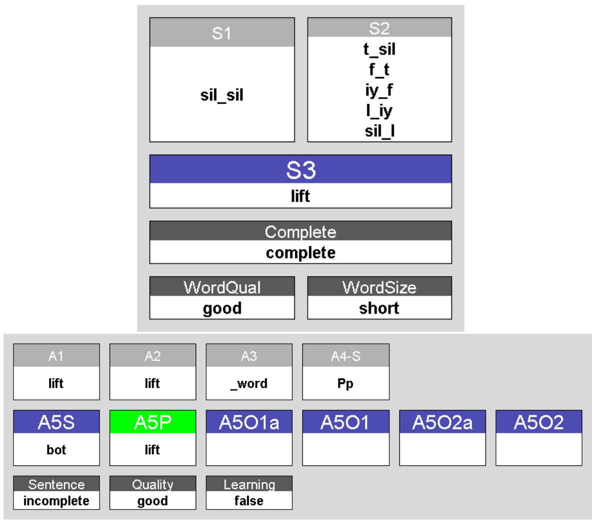


Figure 6. The system after the word "lift" has completely been processed. In the language understanding system (bottom), A5S represents the subject of the sentence and A5P the predicate. In the word recognition system (top), S2 still shows the superposition of all phonetic pairs representing the word "lift", while S3 has uniquely decided for the word "lift". S1 is about to receive a pair of two silence markers ("sil_sil") which causes the S2 and S3 areas to be inhibited as a new word is going to begin.

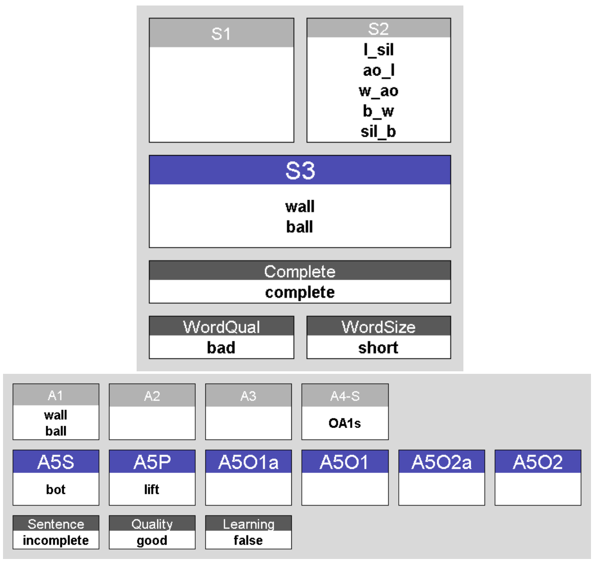


Figure 7. The system after the ambiguous input "bwall" has been processed. S2 shows a superposition of the phonetic pairs that lead to the ambiguous activation in S3. In S2, the more recent input pairs are at the top, so the first recognised phonetic pair for "bwall" was "sil_b" and the second was "w_ao". The two pairs are not matching, which causes the ambiguity. The pattern "bwall" already arrived in module A1 of the language understanding system. The memory has activated a superposition of the "ball" and "wall" assembly.

that the speech recogniser mixed up the "b" and "w" phonemes in the beginning of the word "ball". In figure 4, the state of the word recognition system is shown after the first two phonetic pairs of the word "bot" are processed. S2 shows a superposition of the two pairs, where the older one is display at the bottom because it spikes later. In S3, possible candidates for matching words become active. With only two pairs in S2, all words beginning with "b_aa" are possible, hence, "bot" and "ball" show up in S3 (due to our limited vocabulary, there are no other matching words). Figure 5 shows the system after the word "bot" has been processed completely. The word recognition part shows a unique decision for "bot" in module S3, while the language understanding system receives the input word "bot" in its first input area A1. The additional status fields have recognised that a complete word is understood (which is currently caused by detecting a "xx_sil" pair which means that a silence period is starting) and that the quality of the recognised word is good, i.e. that the system was able to uniquely decide for the word "bot". The field "WordSize" only switches between short and long words, which helps to avoid confusion between long words and shorter words contained in them, e.g. between "bot" and "bottom". Whenever enough phonetic pairs (more than 5) are detected in module S2, a long word is assumed and all short words are inhibited, if a small word is expected, all long words are inhibited. Otherwise, a superposition of "bot" and "bottom" would always become active if "bot" is entered.

A few global steps later, the word "lift" has been processed completely (see figure 6). In the language understanding modules A5X, the subject ("bot") and predicate ("lift") of the sentence are activated, while in the word recognition part, the superposition of phonetic pairs leading to "lift" is still visible. Figure 7 shows the system after the ambiguous input "bwall" has been processed. S2 shows all phonetic pairs that are responsible for the ambiguous state. The pairs the system processed first (the ones displayed at the bottom of S2)

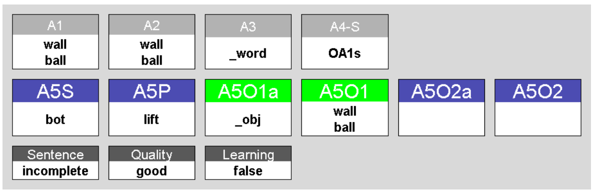


Figure 8. The language understanding system after the "bwall"-pattern is activated in A5O1.

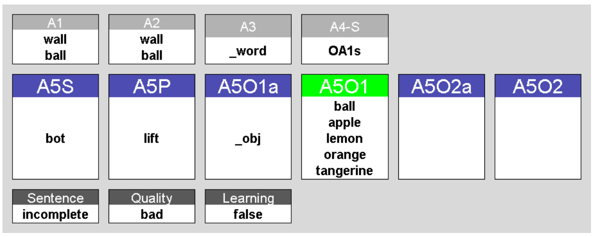


Figure 9. Disambiguation starts to work in module A5O1. There is a weak bidirectional connection between A5P and A5O1 relating verbs with matching objects. When this connection starts to resolve the ambiguity, in a first step, the "wall" assembly is eliminated and "ball" becomes the most likely pattern. The other patterns showing up are other objects that can be lifted, they become active due to the disambiguation connection from A5P to A5O1, but they spike much later than the "ball" assembly. They will be inhibited in a later time step when disambiguation is finished.

are "sil_b", "b_w" and "w_ao", showing that the phonetic input is already ambiguous. The system realises that a unique decision is not possible and activates a superposition of "ball" and "wall" in S3. The ambiguous representation is then forwarded to the language understanding module, which shows the ambiguous pattern in its A1 module. Several processing steps later, the superposition of ball and wall becomes active in module A5O1 (see figure 8). Here, disambiguation starts to take place via a weak bidirectional hetero-associative connection from A5P to A5O1. The connection relates the verbs stored in A5P with objects in A5O1 that the corresponding actions can be performed with. One time step later, the connection starts to work and the "wall" assembly is already inactive in A5O1 (see figure 9). There are many other patterns showing up in A5O1. They are activated by the disambiguation connection from A5P and are all objects that are liftable. Due to the ambiguity that has already been present in A5O1, the module is currently allowing the activation of superpositions of patterns (low separation strength). The liftable objects get input from area A5P, and although the input is very weak, in the current state of A5O1 it is enough to get the patterns activated. A closer look at the spike timings shows however, that the "ball" pattern spikes much earlier than all the other assemblies, meaning that "ball" is the most likely interpretation of the superposition. A few time steps later, the module A5O1 managed to completely resolve the ambiguity (see figure 10). The liftable objects are similar to each other, while wall and ball are completely different patterns. If similar patterns (with larger overlap) are active, the quality of the retrieval is better than in the case of two completely disjoint active patterns. Thus, the quality increases as soon as the wall assembly is not active anymore, leading to a slow increase of the separation strength. This in turn deactivates the very weakly activated additional pattern and resolves the ambiguity to the correct "ball" assembly.

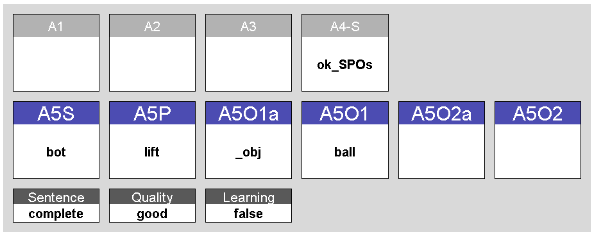


Figure 10. A few time steps after the state shown in figure 9, the ambiguity in the language model is completely resolved due to the context information from the whole sentence.

6 DISCUSSION

We have presented a neurobiologically plausible neural network architecture for understanding language, which is implemented in a robotics context. The model is able to detect and represent ambiguities on different levels of processing and to resolve them as soon as enough context information becomes available.

Our architecture aims at large-scale cortical modelling which uses the interaction of several cortical areas to achieve the understanding of language and the organisation of appropriate responses to these sentences. There are related approaches to brain modelling (see e.g. the work of Arbib and others (e.g. [1, 2]) or the NOMAD project (e.g [12])), but our system can better be understood in the context of a larger model [5] that covers many cortical areas and integrates many different tasks (e.g. language understanding, visual object recognition, visual attention and action planning). Most of the other systems deal only with one or two of these aspects at a time but implement them with a biologically more realistic neural network or focus on learning from a naive initial state. Other approaches deal more specifically with the interpretation or parsing of sentences (e.g. [4, 19, 13]), but often without considering the possibility of embodiment.

A language understanding system like the one presented in this paper generally has at least two difficult problems to solve: The first one is to find out the correct sequence of spoken words from an input stream of phonemes (or other primitive entities a speech recognition software handles), the second one is to analyse the resulting stream of words with respect to a given grammar. In both levels it is necessary to be able to represent and resolve ambiguities: the input from speech recognition software is error prone and even if there are no errors in the input, unique decisions on the single word level without contextual information from the whole sentence might not be possible (e.g.if two words can be concatenated to one longer word it might not be possible to distinguish whether two single words or one long word was spoken). These ambiguities might only be resolved in the context of the whole sentence or might require an even broader context. Thus the ambiguous state must not only be kept over time in one of the processing levels, it must also be forwarded to the next level in some cases.

In earlier work (see e.g. [5, 8, 10]) we have already shown that resolving ambiguities with the help of contextual information and keeping ambiguous states over time is possible with our architecture. This paper now demonstrates that forwarding ambiguous states between several levels of processing is possible with our model. Our previous work was operating on whole-word input, while the new network presented here now implements two stages of sequence detection: On the first level, sequences of phonemes are translated into corresponding words. On this stage of processing, the restrictions with respect to "grammar" in the sequence are not very strong, so we decided to chose a robust architecture that does not rely on grammatical information and is very fault tolerant. On the second level, a sequence of words is detected and interpreted as a sentence. On this level, grammatical restrictions become important which is reflected by the more complex architecture in the language understanding part, which is motivated by the theory of deterministic finite automata and their representation with neural networks (see e.g. [11]).

Obviously, the system should be extended to cover a larger vocabulary and more types of sentences. This will be done in future extensions of our network. Extending the model only requires storing more items in the various associative memories. The current system could handle a larger syntax and vocabulary (at least a factor of three or four times). Further increase would require more neurons. According to the theory of associative memory (see [15, 16, 17, 18]), the storage space needed for the program will increase linearly with the size of the task, whereas the number of neurons and even more the computation time will increase less than linearly.

REFERENCES

[1] M.A. Arbib, A. Billard, M. Iacoboni, and E. Oztop, 'Synthetic brain imaging: Grasping, mirror neurons and imitation.', *Neural Networks*, **13(8-9)**, 975–997, (2000).

[2] M.A. Arbib, A.H. Fagg, and S.T. Grafton, 'Synthetic pet imaging for grasping: From primate neurophysiology to human behavior.', in *Exploratory Analysis and Data Modeling in Functional Neuroimaging.*, eds., F.T. Sommer and A. Wichert, 231–250, MIT Press, Boston, MA, (2002).

[3] V. Braitenberg, 'Cell assemblies in the cerebral cortex.', in *Lecture notes in biomathematics (21). Theoretical approaches to complex systems.*, eds., R. Heim and G. Palm, 171–188, Springer-Verlag, Berlin Heidelberg New York, (1978).

[4] J.L. Elman, 'Finding structure in time.', *Cognitive Science*, **14**, 179–211, (1990).

[5] R. Fay, U. Kaufmann, A. Knoblauch, H. Markert, and G. Palm, 'Combining visual attention, object recognition and associative information processing in a neurobotic system', In Wermter et al. [20].

[6] D.O. Hebb, *The organization of behavior. A neuropsychological theory.*, Wiley, New York, 1949.

[7] A. Knoblauch, 'Synchronization and pattern separation in spiking associative memory and visual cortical areas.', *PhD thesis, Department of Neural Information Processing, University of Ulm, Germany*, (2003).

[8] A. Knoblauch, H. Markert, and G. Palm, 'An associative cortical model of language understanding and action planning', in *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, Lecture Notes in Computer Science 3562, 405–414, Springer, Berlin, (2005).

[9] A. Knoblauch and G. Palm, 'Pattern separation and synchronization in spiking associative memories and visual areas.', *Neural Networks*, **14**, 763–780, (2001).

[10] H. Markert, A. Knoblauch, and G. Palm, 'Associative language processing in cortical areas', in *Proceedings of the IEEE SMC UK-RI Chapter Conference 2005 on Applied Cybernetics*. 4th Chapter Conference on Applied Cybernetics 2005, City University, London, (2005).

[11] H. Markert, A. Knoblauch, and G. Palm, 'Detecting sequences and understanding language with neural associative memories and cell assemblies', In Wermter et al. [20], 106–116.

[12] J.L. McKinstry, G.M. Edelman, and J.L. Kirchmar, 'A cerebellar model for predictive motor control tested in a brain-based device', *Proc Natl Acad Sci USA*, **103**, 3387–3392, (2006).

[13] R. Miikkulainen and D. Bijwaard, 'Parsing Embedded Clauses with Distributed Neural Networks', in *Proceedings of the twelfth National Conference on AI (AAAI 94)*, Menlo Park, Cal, (1994). AAAI Press, MIT Press.

[14] G. Palm, 'Cell assemblies as a guideline for brain research.', *Concepts in Neuroscience*, **1**, 133–148, (1990).

[15] G. Palm, 'Local learning rules and sparse coding in neural networks', in *Advanced Neural Computers*, ed., R. Eckmiller, North-Holland, Amsterdam, (1990).

[16] G. Palm, 'Memory capacities of local rules for synaptic modification. A comparative review.', *Concepts in Neuroscience*, **2**, 97–128, (1991).

[17] G. Palm, 'On the information storage capacity of local learning', *Neural Computation*, **4**, 703–711, (1992).

[18] G. Palm and F.T. Sommer, 'Information capacity in recurrent mcculloch-pitts networks with sparsely coded memory states', *Network*, **3**, 177–186, (1992).

[19] S. Wermter, *Hybrid Connectionist Natural Language Processing*, Chapman and Hall, London, 1995.

[20] *Biomimetic Neural Learning for Intelligent Robots*, eds., S. Wermter, G. Palm, and M. Elshaw, Springer, Heidelberg, New York, 2005.

[21] D.J. Willshaw, O.P. Buneman, and H.C. Longuet-Higgins, 'Non-holographic associative memory.', *Nature*, **222**, 960–962, (1969).

Active Sonar Target Identification Using Evolutionary Neural Logic Networks

Athanasios Tsakonas[1] and **Georgios Dounias**[2] and **Nikitas Nikitakos**[3]

Abstract. A real-world problem is addressed in this work using a novel approach belonging to the area of neural-symbolic systems. Specifically, we apply evolutionary techniques for the development of neural logic networks of arbitrary length and topology. The evolutionary algorithm consists of grammar guided genetic programming using cellular encoding for the representation of neural logic networks into population individuals. The application area is related to the classification of active sonar signals. Our aim is to demonstrate the capability of the system to produce competitive to feedforward neural networks results, yet potentially interpretable. Our experiments show that the overall system is capable to generate arbitrarily connected and competitive evolved solutions for the active sonar target identification, leading potentially to knowledge extraction.

1 INTRODUCTION

Sonar has been used for submarine and mine detection, depth detection, commercial fishing, diving safety and communication at sea. It is a system that uses transmitted and reflected underwater sound waves to detect and locate submerged objects or measure the distances underwater, qualities that fully explicate its name (i.e. an acronym for SOund, NAvigation and Ranging). There are two major kinds of sonar, active and passive. Active sonar creates a pulse of sound, often called a "ping", and then listens for reflections of the pulse. The pulse may be at constant frequency or a chirp of changing frequency. If a chirp, the receiver correlates the frequency of the reflections to the known chirp. The resultant processing gain allows the receiver to derive the same information as if a much shorter pulse of the same total power were emitted. Various intelligent techniques have been developed during the past in order to exploit the sonar data [7,24,25], and computational intelligence (CI) is among them. Although CI has nowadays substituted traditional artificial intelligence in major applications, for a number of high-level decision tasks common expert systems remain still applicable. The reason can be found into the need for symbolic representation of the knowledge into these systems, which is a feature in which many CI systems fail to succeed. In other words, it is considered that symbolic representation can be of significant value in these systems for humans, by making clear the inference process to users. Among CI methodologies, neural networks are powerful connectionist systems that still lack the element of complete and accurate interpretation into human-understandable form of knowledge and remain a black box for experts. To deal with this situation, a number of alternative approaches have been proposed. Neural logic networks [19] belong to this category, and by their definition can be interpreted into a number of logical or Prolog rules that consist an expert system. Virtually every logic rule can be represented into these networks and then transformed into Prolog commands. Although this model offers excellent results when used within the AI framework (i.e. building a system in a top-down process), the application of neural logic networks in CI's data mining tasks – considered a bottom-up procedure- has undergone limited success. The reason lies in that proposed systems suffered at least one of the following limitations: (a) The extracted neural logic network cannot be interpreted into expert rules [19]-[18]. (b)The proposed methodology cannot express neural logic networks in their generic graph form [5]. (c) The user has to select topology and network connection model [19]-[18]. The application of neural logic networks into adaptive tasks seems promising: the extracted model will preserve its interpretability into a number of expert rules and there is not needed any knowledge-acquiring step. Moreover, a solution obtained this way, leads to potential knowledge extraction. Recently, a new system, namely the evolutionary neural logic networks (ENLN), has been proposed [20] that fulfils those requirements. The new approach uses grammar-guided genetic programming to produce neural logic networks. The evolved solutions can be arbitrarily large and connected networks, since an indirect encoding is adopted. Also, neural logic networks produced by this methodology can always be interpreted into human-understandable expert rules, thus leading to potential knowledge extraction. Our aim in this paper is to demonstrate the effectiveness of the methodology of evolutionary neural networks into real-world problems. The paper is organized as follows. Next section describes the theoretical background, presenting the neural logic networks concept and the grammar guided genetic programming. Following this section, we deal with the design and the implementation of the ENLN system. Next, the results and a following discussion are presented. The paper ends with our conclusion and a description of future work.

2 BACKGROUND

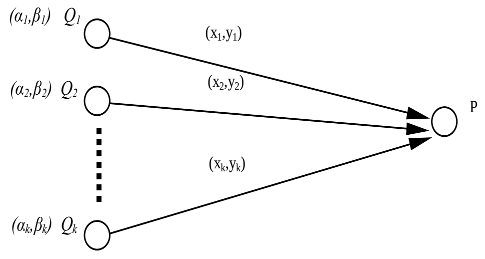
2.1 Active Sonar

In Sea-Water environments, propagation to the target, reflection off

[1] University of the Aegean, Dept. of Finance and Management Eng., Greece, email: tsakonas@stt.aegean.gr

[2] University of the Aegean, Dept. of Finance and Management Eng., Greece, email: g.dounias@aegean.gr

[3] University of the Aegean, Dept. of Shipping, Trade and Transport, Greece, email: nnik@aegean.gr



$$(a,b)=\begin{cases}(1,0) & if \sum_{j=1}^{k} a_j x_j - \sum_{j=1}^{k} b_j y_j \geq 1\\ (0,1) & if \sum_{j=1}^{k} a_j x_j - \sum_{j=1}^{k} b_j y_j \leq -1\\ (0,0) & otherwise\end{cases}$$

Figure 1. Example of a neural logic network and its output.

the target, and propagation to the receiver spread active sonar transmit signal in time and frequency. Traditionally, detection and subsequent range estimation has been performed by thresholding a normalized matched filter output for each of several beams pointing in directions of interest. This is only justifiable as a generalized likelihood ratio test when the received echo is simply a time-shifted scaled version of the transmitted waveform plus white noise, obviously not a realistic scenario in the shallow-water active problem. The primary objective of the detector is to determine if there is a target echo present in the received time series. Subsidiary to detection is the estimation of the starting and stopping times of the echo to be used for subsequent signal processing such as more accurate range and bearing estimation, depth estimation, or classification. Traditionally, signal segmentation is performed by clustering matched filter threshold crossings. Without exact knowledge of the environment and a priori information on the target location and reflection properties, the starting time, duration, and shape of the received echo are unknown, thus hindering design of an optimal receiver. It is, however, desirable to exploit available environmental information to the extent that it can feasibly improve detection performance. Were the optimal detector implementable, it would coherently combine the standard matched filter output according to the multipath structure and the target reflection properties. Few literature papers exist in applications with active sonar. In [25], a procedure of mapping unknown obstacles using active sonar is presented. An active sonar imaging and classification system is described in [2] were three neural network architectures were used as classifiers. In [6], the Probabilistic Multi-Hypothesis Tracking (PMHT) algorithm proposed by Streit and Luginbuhl in 1995 is adapted for use in active sonar applications. In [3], a remote, aerial, laser-based sonar method for detecting and locating underwater targets from the air is discussed.

2.2 Neural logic networks

The neural logic network is a finite directed graph. It is usually consisted by 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 assigned also an ordered pair weight (x,y) where x and y are real numbers. An example neural logic and its output value (a,b) of node P is shown in Fig. 1. Different sets of weights enable the representation of different logical operations. It is actually possible to map any rule of conventional knowledge into a neural logic network. Neural logic networks can be expanded into fuzzy neural logic networks, enabling this way the handling of real valued attributes [19]. The interpretation of the network into Prolog rules is straightforward. Even though powerful in their definition, neural logic networks are not widely applied. The main reason can be located in the fact that for the known training methodologies [19]-[18], the refinement of the edge weights reduces significantly the interpretability of these networks to expert rules, thus depriving these networks from their valuable feature. Some steps for the preservation of the interpretability have been performed by [1], without however the ability to express arbitrarily large and connected neural logic networks. In this direction, the definition and use of the *neulonet* is demonstrated in [4], still however producing tree-like rule programs.

2.3 Grammar Guided Genetic Programming

The ability to construct functional trees of variable length is a major advantage of genetic programming over genetic algorithms. 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 symbolic regression. Later paradigms extended this concept to calculate any boolean or programming expression. Consequently, complex intelligent structures, such as fuzzy rule-based systems or decision trees have already been used as the desirable target solution in genetic programming approaches [1], [22], [23], [24]. The main qualification of this solving procedure is that the feature selection, and the system configuration, derive in the searching process and do not require any human involvement. Moreover, genetic programming, by inheriting the genetic algorithms' stochastic search properties, does not use local search -rather uses the hyper plane search-, and so avoids driving the solution to any local minimum. The potential gain of an automated feature selection and system configuration is obvious; no prior knowledge is required and, furthermore, not any human expertise is needed to construct an intelligent system. Nevertheless, 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 GP 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. Thus, a genotype - a point in the search space- corresponds always to a phenotype - a point in the solution space. This approach -known as legal search space handling method [28]- is applied in this work using context-free grammars. As we will discuss in the next paragraph, the implementation of constraints using a grammar can be the most natural way to express a family of allowable architectures. While each intelligent system -such as a neural logic network- has a functional equivalent -by means of being composed by smaller, elementary functions-, what defines and distinguishes this system is its grammar.

2.4 Context-free grammars

The genetic programming procedure may be proved greedy in computational and time resources. Consequently, when the syntax

```
<PROG>   : =  PROG <PLACE1><SYNAPSE>
<PLACE1> : =  S1 <PLACE1><SYNAPSE><PLACE2>
             | P1 <PLACE1><PLACE1>
             | IN
IN       : =  Data attribute (system input)
<PLACE2> : =  S2 <PLACE2><SYNAPSE><PLACE2>
             | P2 <PLACE2><SYNAPSE><PLACE2>
             | E
E        : =  ∅
<SYNAPSE>     : =  LNK <NUM><CUT><SYNAPSE>
             | 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]
```

Figure 2. Context free grammar for the production of neural logic networks within the genetic programming framework.

form of the desired solution is already known, it is useful to restrain the genetic programming from searching solutions with different syntax forms [8]-[14]. The most advantageous method to implement such restrictions among other approaches [15], is to apply syntax constraints to genetic programming trees, usually with the help of a context-free grammar declared in the Backus-Naur-Form (BNF) [14]. The BNF-grammar consists of terminal nodes and non-terminal nodes and is represented by the set {N,T,P,S} where N is the set of non-terminals, T is the set of terminals, P is the set of production rules and S is a member of N corresponding to the starting symbol. The use of the terms terminal and non-terminal in a BNF-grammar, does not correspond to what is usually referred in genetic programming as terminal and function [14]. Rather, a function -a non-terminal node in terms of the GP tree architecture- is expressed as terminal in a BNF grammar.

2.5 Cellular Encoding

Although mapping decision trees or fuzzy rule-based systems to specific grammars can be relatively easy to implement, the execution of massively parallel processing intelligent systems -such as the neural logic networks- is not forthright. In order to explore variable sized solutions, we applied indirect encoding. The most common one is the cellular encoding [9], in which a genotype can be realized as a descriptive phenotype for the desired solution. More specifically, within such a function set, there are elementary functions that modify the system architecture together with functions that calculate tuning variables. Current implementations include encoding for feed forward and Kohonen neural networks [16], [21] and fuzzy Petri-nets [27], [21]. In his original work, Gruau also used a context-free grammar - a BNF grammar- to encode indirectly the neural networks. On the other hand, in [27] a logic grammar - a context-sensitive one- is adapted to encode fuzzy Petri-nets. In our work, we show that as long as the depth-first execution of the program nodes of a GP tree is ensured -which is the default-, a context-free grammar such as a BNF grammar is adequate for expressing neural networks. Gruau's original work has been facing some scepticism [11] on the ability to express arbitrarily connected networks. Later developments [8] seem to offer less restrictive grammar, though the cut function in those implementations still maintained bounded effect. A similar technology, called edge encoding, developed by [12] is also today used with human competitive results in a wide area of applications.

Table I. Operations for Function CNR

Parameter	Calculation
0	Conjunction
1	Disjunction
2	Priority
3	At least k-true
4	At least k-false
5	Majority influence
6	Majority influence of k
7	2/3 Majority
8	Unanimity
9	If-Then operation, Kleene’s model
10	Difference

3 DESIGN AND IMPLEMENTATION

The data is normalized to the system’s acceptable data range and the procedure creates the evolutionary neural logic network, which is then tested on unknown data. The resulted network is stored and the rules extracted can be used without the need of a computer.

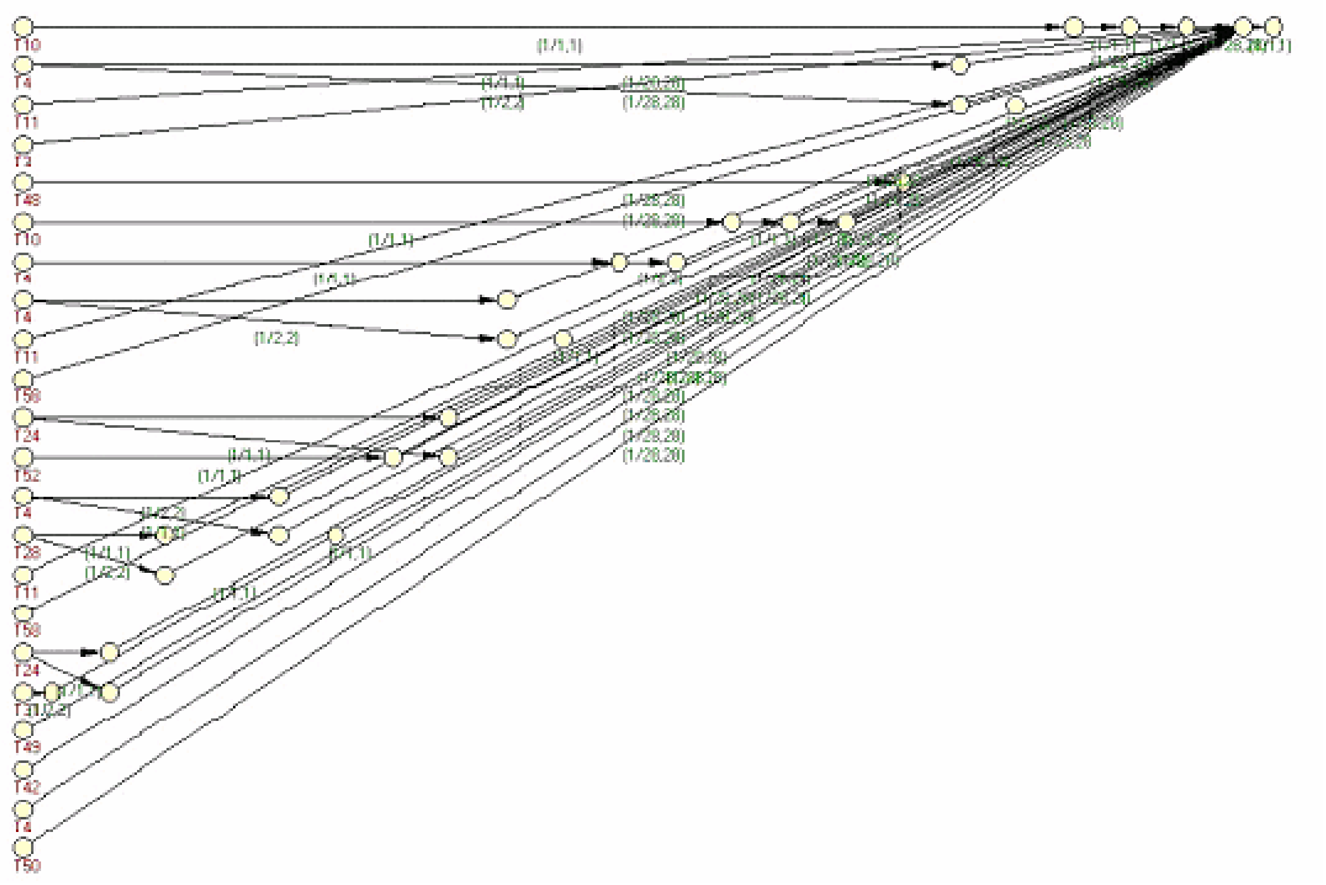
3.1 Data Pre-processing and Genetic Programming Setup

The Sonar data set is consisted of sixty real-valued attributes between 0.0 and 1.0 used to define 208 mines and rocks. Attributes are obtained by bouncing sonar signals of a metal cylinder (or rock) at various angles and rises in frequency. The value of the attribute represents the amount of energy within a particular frequency band, integrated over a certain period of time. It is accepted that the genetic programming procedure may suffer size problems during initialisation [16]. Although the fine-tuning of our algorithm was not the main concern of this paper, we investigated various initialisation approaches. Without claiming optimality, the GP parameters are adapted by [20]. This setup, together with function selection probability optimisation, offered for the presented grammar stable and effective runs throughout experiments. Although the initialisation of the population is random, using this probability bias the algorithm is forced to generate individuals of acceptable size. This optimisation was decided after experimentation, since it is not possible to obtain a general principle regarding the most proper probability values for every case. As it can be observed in [20], the setup denotes our preference for significantly high mutation rates, especially shrink mutation [17] that slows down the code bloat caused by crossover operations.

3.2 System Grammar and Operating Functions

The proposed system grammar is shown in Figure 2. Initial symbol (root) of a genetic programming tree can be a node of type <PROG>. The function set is as follows:

- Function PROG: The function PROG creates the embryonic network that is used later by the functions S1, S2, P1 and P2 to be expanded. An alternative name for this function, which is used throughout this paper, is the term “CNLN”.
- Function S1: The function S1 enters a node in serial to the node that is applied, and is applied to input nodes.
- Function P1:The function P1 enters a node in parallel to



(CNLN (P1 (P1 (P1 (S1 (S1 (In T10) (Rule 0 0) E) (Rule 0 0) (S2 E (Rule 0 0) E)) (P1 (S1 (In T4) (Rule 0 0) (P2 E (Rule 10 3) (S2 E (Rule 10 3) E))) (In T11))) (P1 (P1 (In T3) (S1 (In T48) (Rule 12 8) E)) (P1 (P1 (P1 (S1 (S1 (In T10) (Rule 0 0) E) (Rule 0 0) (S2 E (Rule 0 0) E)) (P1 (S1 (S1 (In T4) (Link 50 0 (Rule 0 0)) E) (Rule 10 3) E) (P1 (S1 (In T4) (Rule 12 8) (P2 E (Rule 10 3) (S2 E (Rule 0 0) E))) (In T11)))) (P1 (P1 (In T58) (S1 (In T24) (Rule 12 8) (P2 E (Link 133 0 (Rule 0 0)) E))) (P1 (P1 (S1 (In T52) (Rule 0 0) E) (P1 (S1 (In T4) (Link 133 0 (Rule 10 3)) (P2 E (Rule 0 0) (S2 E (Rule 0 0) E))) (P1 (S1 (In T28) (Rule 0 0) (P2 E (Rule 10 3) (S2 E (Rule 0 0) E))) (In T11)))) (P1 (P1 (In T58) (S1 (In T24) (Rule 12 8) (P2 E (Link 133 0 (Rule 0 0)) E))) (P1 (P1 (S1 (In T31) (Rule 10 3) E) (In T49)) (In T42)))))) (In T4)))) (In T50)) (Rule 2 8))

Figure 3. Extracted solution and the corresponding neural logic network for the Active Sonar classification problem.

the node that is applied, and is also applied to input nodes.

- Function S2: The function S2 enters a node in serial to the node that is applied, and is used for hidden layer nodes.
- Function P2: The function P2 enters a node in parallel to the node that is applied, and is also used for hidden layer nodes. This mechanism, consisting of two different sets of expanding functions (P1 and S1 vs. P2 and S2), is used to ensure that population individuals will include at least one input node.
- Function IN: The operation of function IN is to assign a variable to the input node that it is applied.
- Function E: The operation of function E is to mark the end of the expansion of the network.
- Function LNK: This function provides the framework for the application of cut function. It actually enables the non-full connectivity of the network, a feature that offers larger solution search space.
- Function CNR: This function performs the node inference. Based on the first parameter, the corresponding calculation is performed. The second parameter assists the calculation for the at-least-k and majority-of-k operators. Possible computations are shown in Table I. An alternative name for this function, which is used throughout this paper, is the term “Rule”. In order to be able to process values other than true, false and don’t know, we applied the fuzzy propagation algorithm [19], which allows us to process any real valued variables (using proper normalization).
- Function NUM: The function NUM returns an integer in the interval [1,256] to be used by the calling LNK function.
- Function CUT: The function CUT returns an integer in the interval [0,1] to be used by the calling LNK function. If the returned value is 1, then the link will be ignored in the calculations (considered "cut").
- Function CNRSEL: The function CNRSEL returns an integer in the interval [0,8] to be used by the calling CNR function as its first parameter.
- Function K: The function K returns an integer in the interval [1,256] to be used by the calling CNR function if the returned value of the CNRSEL is 3,4 or 6 (corresponding to the calculation of the at least k-true, at least k-false and majority of k functions).

Having discussed the system design, in the following session we shall apply the methodology in the active sonar target identification domain.

4 RESULTS AND DISCUSSION

The remote detection of undersea mines in shallow waters using active sonar is a crucial capability required to maintain the security of important harbours ands coastline areas. It is often very difficult to distinguish active sonar returns from mines and return from clutter on the sea floor. There is currently no reliable signal classification scheme for automatically interpreting such sonar returns. Instead highly trained sonar operators must be relied upon to identify the presence of a mine. The present study was conducted to explore the use of neural networks as a means of automating mine-hunting operations. More specifically, the system develops a decision whether the signal corresponds to a cylinder (mine) or to a rock [7]. The database is consisted of two parts. The first part contains 111 records which are acquired by returning sonar signals from a metal cylinder in various positions. The second part is comprised by 97 records which correspond to sonar signals returned by rocks in similar situations. Training and testing data records were randomly selected by these sets. Also, in order to avoid overfitting during the training phase, we used a validation set. According to the literature, the target is to develop a system with high accuracy and potential knowledge interpretation. The evolved neural logic network and its graphical representation are depicted in Figure 3. This solution achieves a 86.27% (44/51) accuracy in unknown data. The accuracy in the training data was 88.24% (90/102) and in the validation data set it was 80.39% (41/51). In the literature, the derived accuracy for the various systems applied in the same data set ranges from 73.1% to 89.2%. Other experiments in [3] using neural logic networks by means of genetic programming offer for the same data set an equivalent classification score (86%) when *neulonet* association rules are applied, and a lower score (78.3%) when conjunctive association rules are used, a direct score comparison not being applicable however, since different training and test data sets have been used. The extracted neural logic network, due to the nature of the problem, maintains significant complexity, yet it achieves competitive to the literature results. It is worth to note also, that our solution can still be interpreted into a number of logical or Prolog rules, although solution interpretation was not among our primary targets for the specific problem.

4 CONCLUSION AND FURTHER RESEARCH

The aim of this paper was to demonstrate the effectiveness of the evolutionary neural logic networks paradigm into real-world problems, such as the active sonar target identification. In general, neural networks are powerful connectionist systems that have been introduced in areas where symbolic processing systems of traditional artificial intelligence used to be applied. As a tool of computational intelligence, the adaptation of the neural network to the problem domain using an inductive method, offers advantage over expert systems where the knowledge must be acquired first, before the system development. Ever since their first application, interpretation of the obtained knowledge was a research target for neural networks. In the scope of this area, the neural logic networks have been proposed as a class of networks that by their definition preserve their interpretability into symbolic knowledge.

Until recently however, the application of an effective training / production method within the CI framework has not been successful. A novel system that uses genetic programming with indirect encoding that has been proposed recently [20], overcomes these problems, producing automatically designed and tuned neural logic networks, which always preserve their interpretability. In this work we applied the system into a real-world problem, the Active Sonar classification problem. The system has been proved capable of producing competitive to the literature results. The acquired solution although being in a complicated form, it still maintains its interpretability. The complexity of the solution is rather straightforwardly related to the nature of the problem, i.e. physical measurements. However, as obviously seen, the solution interpretation could not be among the primary targets of this research, rather than the high classification rate. Hence, according to the experts, the application to a sonar classification problem shows that under particular circumstances the system can be implemented to some degree into this real situation problem.

Future work involves the application of the system in other sonar data sets, as well as in other areas, and the incorporation of recursive structures into the neural logic network architecture. Moreover, the *minimum description length* principle will be developed to be included as an anti-overfitting measure into the active sonar target identification problem. Finally, we believe that parameter-tuning optimisation of the underlying genetic programming algorithm will offer better efficiency; hence this will be of primary importance among our future work.

REFERENCES

[1] Alba E., Cotta C. and Troya J.M., "Evolutionary Design of Fuzzy Logic Controllers Using Strongly-Typed GP", Proc. 1996 IEEE Int'l Symposium on Intelligent Control, pp. 127-132. New York, NY., 1996.

[2] L. L. Burton and H. Lai. Active sonar target imaging and classication system. In Proceeedings of the SPIE International Symposium on Aerospace/Defence Sensing and Control, pages 19-33, Orlando, FL, April 1997.

[3] Chia H W-K and C-L Tan, Confidence and support classification using genetically programmed neural logic networks, Genetic and Evolutionary Computation Conference, GECCO 2004, 26-30 June 2004, Seattle, Washington, USA

[4] Chia H W-K and C-L Tan, Association-based evolution of comprehensive neural logic networks, GECCO 2004, 26-30 June 2004, Seattle, Washington, USA.

[5] Chia H.W-K. and Tan C-L.,"Neural logic network learning using genetic programming", Intl. Journal of Comp. Intelligence and Applications, 1:4, 2001, pp 357-368.

[6] Christian G. Hempel and Sheri L. Doran, A PMHT algorithm for active sonar, Proc. SPIE 5430, 132 (2004)

[7] Gorman, R. P., and Sejnowski, T. J. (1988). "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets" in Neural Networks, Vol. 1, pp. 75-89.

[8] Gruau F., Whitley D. and Pyeatt L., "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, 1996.

[9] Gruau F., "Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm", Ph.D. Thesis, Ecole Normale Superieure de Lyon, anonymous ftp:lip.ens-lyon.fr (140.77.1.11) pub/Rapports/ PhD PhD94-01-E.ps.Z.

[10] Gruau F., "On Using Syntactic Constraints with Genetic Programming", in P.J.Angeline, K.E.Jinnear,Jr., Eds., Advances in Genetic Programming, MIT,1996.

[11] Hussain T., "Cellular Encoding: Review and Critique", Technical Report, Queen's University, 1997, http://www.qucis.queensuca/ home/hussain/web/ 1997_cellular_encoding_review.ps.gz

[12] J.Koza, F. Bennett, D. Andre and M. Keane, Genetic Programming III: Automatic Programming and Automatic Circuit Synthesis, Morgan Kaufmann, 2003.

[13] Montana D.J., "Strongly Typed Genetic Programming", Evolutionary Computation, vol. .3, no. 2, 1995.

[14] Naur P., "Revised report on the algorithmic language ALGOL 60", Commun. ACM, Vol 6, No 1, pp 1-17, Jan 1963.

[15] N.Paterson and M.Livesey,"Evolving Caching Algorithms in C by

GP", Genetic Programming 1997, pp 262-267, MIT Press, 1997.

[16] Ratle A. and Sebag M., "Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery", Schienauer et al., Eds., Proc. of the 6th Conf. on Parallel Problems Solving from Nature, LNCS, Springer, Berlin, 2000, pp 211-220

[17] Singleton A., "Genetic Programming with C++", BYTE Magazine, Feb 1994.

[18] Tan A-H. and Teow L-N., “Inductive neural logic network and the SCM algorithm”, Neurocomputing, Vol. 14, 2 : 5, pp.157-176, 1997.

[19] Teh H.H., Neural Logic Networks: A New Class of Neural Networks, World Scientific Pub Co, 1995.

[20] Tsakonas A., Aggelis V., Karkazis I. and Dounias G., “An Evolutionary System for Neural Logic Networks using Genetic Programming and Indirect Encoding”, Journal of Applied Logic, Special Issue on Neural-Symbolic Systems, Vol.2 (3), September 2004, pp.349-379, Elsevier.

[21] Tsakonas A. and Dounias G., Decision Making in the Medical Domain: Comparing the Effectiveness of GP-Generated Fuzzy Intelligent Structures, Proc. of Eunite-03, Oulou, Finland, 2003.

[22] Tsakonas A., Dounias G., "Hierarchical Classification Trees Using Type-Constrained Genetic Programming", Proc. of 1st Intl. IEEE Symposium in Intelligent Systems, Varna, Bulgaria, 2002.

[23] Tsakonas A., Dounias G., Axer H., and von Keyserlingk D.G., "Data Classification using Fuzzy Rule-Based Systems represented as Genetic Programming Type-Constrained Trees", Proc. of the UKCI-01, Edinbourgh, UK, pp 162-168, 2001.

[24] Tsakonas A. and Dounias G., "A Scheme for the Evolution of Feedforward Neural Networks using BNF-Grammar Driven Genetic Programming", Proc. of Eunite-02, Algarve, Portugal, 2002.

[25] F. Wallner, R. Graf, and R. Dillmann. Real-time map refinement by fusing sonar and active stereo-vision. In IEEE International Conference on Robotics and Automation, Nagoya, Japan, 1995.

[26] Whigham P., "Search Bias, Language Bias and Genetic Programming", Genetic Programming 1996, pp 230-237, MIT Press, 1996

[27] Wong M.L., "A flexible knowledge discovery system using genetic programming and logic grammars", Decision Support Systems, 31, 2001, pp 405-428.

[28] Yu T. and Bentley P., "Methods to Evolve Legal Phenotypes", Lecture Notes in Comp. Science 1498, Proc. of. Parallel Problem Solving from Nature V, pp 280-291,1998.

Construction of Neurules from Training Examples: A Thorough Investigation

Jim Prentzas[1] and Ioannis Hatzilygeroudis[2]

Abstract. Neurules are a type of hybrid rules combining a symbolic and a connectionist representation. A neurule base consists of a number of autonomous adaline units (neurules), in contrast to existing neuro-symbolic knowledge bases. A neurule base is constructed from training examples. To overcome the inability of the adaline unit to classify non-separable training examples, the notion of 'closeness' between training examples has been used to split the initial training set into subsets that can be successfully trained. In this paper, we investigate previously unexplored aspects regarding the construction of neurules from training examples. First, we compare different splitting policies, i.e. policies using different criteria for splitting the training set. We also introduce two alternative approaches to splitting not solely relying on closeness and compare them with our initial approach, which is solely based on closeness. The comparison demonstrates the effectiveness of the notion of 'closeness' in splitting the initial non-separable training set. Finally, we evaluate the generalization capability of neurules.

1 INTRODUCTION

Recently there has been extensive research activity at combining (or integrating) the symbolic and the connectionist approaches for problem solving in intelligent systems [3, 4, 5, 12, 13, 14, 15, 19, 21]. Especially, there are a number of efforts at combining symbolic rules and neural networks for knowledge representation [6, 20]. What they do is a kind of mapping from symbolic rules to a neural network. Also, connectionist expert systems are a type of integrated systems that represent relationships between concepts, considered as nodes of a neural network [7, 8]. The strong point of those approaches is that knowledge elicitation from experts is reduced to a minimum. A weak point of them is that the resulted systems lack the naturalness and modularity of symbolic rules. This is mainly due to the fact that those approaches give pre-eminence to connectionism. So, explanations are often provided in the form of if-then rules by rule extraction methods [1, 2].

Neurules constitute a hybrid rule-based representation scheme achieving a uniform and tight integration of a symbolic component (production rules) and a connectionist one (the adaline unit) [8, 9]. In contrast to other integrated approaches, neurules give pre-eminence to the symbolic component. Each neurule is considered as an adaline unit. Thus, neurules give a more natural way of representing knowledge since the constructed knowledge base retains the modularity and (to some degree) the naturalness of symbolic rules. Also, the corresponding inference mechanism, which is a tightly integrated process, results in more efficient inference than those of symbolic rules, and explanations, in the form of if-then rules, can be provided [11]. Mechanisms for efficiently updating a neurule base, given changes to its source knowledge, have also been developed [17, 18].

One way of constructing neurules is from empirical data (i.e., training examples) [10]. A difficult point in this approach is the inherent inability of the adaline unit to classify non-separable training examples. To overcome this difficulty of the adaline unit, we introduced the notion of 'closeness', as far as the training examples are concerned. That is, when the LMS algorithm fails to produce weights that classify all the examples, due to non-separability, we split the initial training set of the involved neurule in two subsets, which contain 'close' examples, and train a copy of the neurule for each subset. Failure of training any copy leads to further splitting as far as success is achieved.

In this paper, we investigate previously unexplored aspects regarding the construction of neurules from training examples. First, we compare different splitting policies, i.e. policies using different criteria for splitting the training set. Second, we introduce alternative approaches to constructing neurules from training examples, not solely relying on closeness to perform splitting. We also compare these alternative approaches with our initial approach, which is solely based on closeness. Finally, we present experimental results evaluating the generalization capability of neurules and comparing it with the generalization capability of a back-propagation neural network and a single adaline unit.

The structure of the paper is as follows. Section 2 presents neurules, the mechanism for their construction from training examples and different splitting policies (based on closeness). Section 3 introduces alternative approaches to splitting (not solely relying on closeness). Section 4 presents experimental results and finally Section 5 concludes the paper.

[1] Technological Educational Institute of Lamia, Department of Informatics and Computer Technology, 35100 Lamia, Greece, email: dprentzas@teilam.gr.
[2] University of Patras, Dept of Computer Engineering & Informatics, 26500 Patras, Greece, email: ihatz@ceid.upatras.gr.

2 NEURULES

2.1 Syntax and Semantics

Neurules (: *neu*ral *rules*) are a kind of hybrid rules. Each neurule (Fig. 1a) is considered as an adaline unit (Fig.1b). The *inputs* C_i (i=1,...,n) of the unit are the *conditions* of the rule. Each condition C_i is assigned a number sf_i, called a *significance factor,* corresponding to the weight of the corresponding input of the adaline unit. Moreover, each rule itself is assigned a number sf_0, called the *bias factor,* corresponding to the *bias* of the unit.

Each input takes a value from the following set of discrete values: [1 (true), -1 (false), 0 (unknown)]. The *output D,* which represents the *conclusion* of the rule, is calculated via the formulas:

$$D = f(\mathbf{a}), \quad \mathbf{a} = sf_0 + \sum_{i=1}^{n} sf_i \; C_i \qquad (1)$$

where **a** is the *activation value* and *f(x)* the *activation function,* which is a threshold function:

$$f(\text{a}) = \begin{cases} 1 & \text{if a} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Hence, the output can take one of two values, '-1' and '1', representing failure and success of the rule respectively.

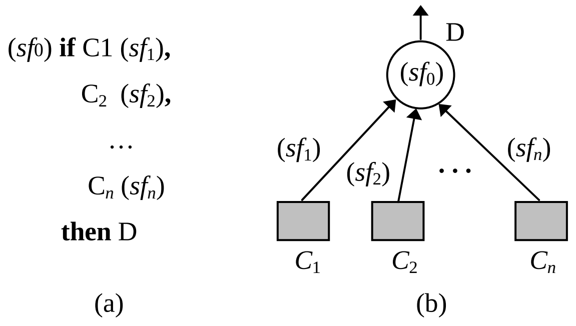


Figure 1. (a) Form of a neurule (b) corresponding adaline unit

The general syntax of a neurule (in a BNF notation, where '{}' denotes zero, one or more occurrences and '<>' denotes non-terminal symbols) is:

<rule>::= (<bias-factor>) **if** <conditions> **then** <conclusions>
<conditions>::= <condition> {**,** <condition>}
<conclusions>::= <conclusion> {**,** <conclusion>}
<condition>::= <variable> <l-predicate> <value>
(<significance-factor>)
<conclusion>::= <variable> <r-predicate> <value> .

In the above definition, <variable> denotes a *variable*, that is a symbol representing a concept in the domain, e.g., 'sex', 'pain' etc, in a medical domain. A variable in a condition can be either an *input variable* or an *intermediate variable,* whereas a variable in a conclusion can be either an *intermediate* or an *output variable*. <l-predicate> denotes a symbolic or a numeric predicate. The symbolic predicates are {is, isnot}, whereas the numeric predicates are {<, >, =}. <r-predicate> can only be a symbolic predicate. <value> denotes a value. It can be a symbol or a number. <bias-factor> and <significance-factor> are (real) numbers. The significance factor of a condition represents the significance (weight) of the condition in drawing the conclusion.

2.2 Constructing Neurules from Training Examples

Each neurule is individually trained via a *training set*, which contains *training examples* in the form [v_1 v_2 … v_n d], where v_i, i= 1, …,n are their *component values,* corresponding to the n inputs of the neurule, and d is the *desired output* ('1' for success, '-1' for failure). We call *success examples* the examples with d=1 and *failure examples* the ones with d=-1. The learning algorithm employed is the standard least mean square (LMS) algorithm.

However, there are cases where the LMS algorithm fails to specify the right significance factors for a number of neurules. That is, the adaline unit of a rule does not correctly classify some of the training examples. This means that the training examples correspond to a non-separable (boolean) function. To overcome this problem, the initial training set is split into two subsets in a way that each subset contains success examples, which are 'close' to each other in some degree. The *closeness* between two examples is defined as the number of common component values. For example, the closeness of [1 0 1 1 1] and [1 1 0 1 1] is '2'. Also, we define as *least closeness pair* (LCP), a pair of success examples with the least closeness in a training set. There may be more than one LCP in a training set.

Initially, a LCP in the training set is found and two subsets are created each containing as its initial element one of the success examples of that pair, called its *pivot*. Each of the remaining success examples is distributed between the two subsets based on its closeness to the pivots. More specifically, each subset contains the success examples, which are closer to its pivot. Then, the failure examples of the initial set are added to both subsets, to avoid neurule misfiring. After that, two copies of the initial neurule, one for each subset, are trained employing the LMS learning algorithm. If the factors of a copy misclassify some of its examples, the corresponding subset is further split into two other subsets, based on one of its LCPs. This continues, until all examples are classified. This means that from an initial neurule more than one final neurule may be produced, called *sibling neurules* (for details see [10]).

To illustrate how splitting is performed, we use as an example the training set presented in Table 1. As it is clear, the majority of the examples in the training set are failure examples, whereas success examples, which are shown in bold, are a minority. The training set has been extracted from empirical data concerning five input (domain) variables and an output variable (disease) that depends on the five domain variables. Given that each input variable can take more than one discrete value, each initial neurule has thirteen conditions (C1-C13). D corresponds to the conclusion. Actually Table 1, for simplicity reasons, shows only a subset of the failure examples.

Table 1. An example training set

C1	*C2*	*C3*	*C4*	*C5*	*C6*	*C7*	*C8*	*C9*	*C10*	*C11*	*C12*	*C13*	*D*
-1	-1	1	-1	-1	1	1	1	-1	1	-1	-1	-1	-1
-1	**-1**	**1**	**-1**	**-1**	**1**	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**	**1**
-1	-1	1	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	1	-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	**-1**	**1**	**1**	**-1**	**1**	**-1**	**-1**	**-1**	**-1**	**-1**	**1**	**1**	**1**
-1	-1	1	1	-1	1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1	1	-1	1	-1	-1
-1	-1	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1
-1	**-1**	**1**	**1**	**-1**	**1**	**-1**	**-1**	**1**	**-1**	**1**	**-1**	**-1**	**1**
-1	1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	1	-1
-1	1	1	-1	-1	1	-1	1	-1	-1	-1	-1	-1	-1
-1	1	1	-1	-1	1	-1	1	-1	-1	-1	-1	1	-1
-1	**1**	**1**	**-1**	**-1**	**1**	**-1**	**1**	**-1**	**1**	**-1**	**-1**	**-1**	**1**
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
1	1	1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
1	1	1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
1	**1**	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**1**

For presentation reasons, names (P1-P5) are assigned to the five success examples/patterns (of Table 1), as presented in Table 2. Also, let F be the set of failure examples in the training set.

Table 2. Success examples

symbol	description
P1	[-1, -1, 1, -1, -1, 1, 1, 1, 1, -1, -1, -1, -1, 1]
P2	[-1, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, 1]
P3	[-1, -1, 1, 1, -1, 1, -1, -1, 1, -1, 1, -1, -1, 1]
P4	[-1, 1, 1, -1, -1, 1, -1, 1, -1, 1, -1, -1, -1, 1]
P5	[1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1]

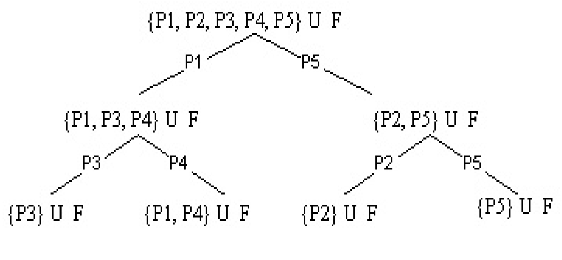


Figure 2. Splittings of the training set of Table 1

Due to inseparability, the initial training set {P1, P2, P3, P4, P5} ∪ F is split in two subsets: {P1, P3, P4} ∪ F and {P2, P5} ∪ F with as least closeness pair (P1, P5). Subset {P1, P3, P4} ∪ F is subsequently split into subsets {P3} ∪ F and {P1, P4} ∪ F. Subset {P3} ∪ F produces a neurule (see Figure 3). Subset {P1, P4} ∪ F produces another neurule. Similarly, from subset {P2, P5} ∪ F two other neurules are produced (corresponding to its two leaves). The performed splittings are illustrated in Figure 2, as a tree.

In creating the training subsets, some requirements were implicitly satisfied. Each training subset contains: (a) all the failure examples of the initial training set to protect from misactivations and (b) at least one success example to guarantee the activation of the corresponding neurule. Furthermore, the two subsets created by splitting a (sub)set do not have common success examples to avoid having different neurules activated by the same success example(s). In the following sections, the approach to splitting based on closeness will be called CLOSENESS-SPLIT.

A point of interest in training a neurule with a non-separable training set is how to choose a least closeness pair (LCP), in the process of producing the two subsets of the initial training set. Not all LCPs result in the same number of final neurules. So, we are looking for the LCP that finally produces the minimum number of sibling neurules. We tried three heuristic methods for that: the *random choice*, the *best distribution* and the *mean closeness method*. The *random choice method* (RC) chooses randomly one of the LCPs and is the simplest and least expensive of the three methods. The *best distribution method* (BD) suggests choosing the LCP that assures distribution of the two elements of all the other (or most of the other) LCPs in different sets. So, examples with least closeness will be included in different sets, which may assure separability. The *mean closeness method* (MC) initially computes the mean closeness of each of the two subsets to be created from each LCP. Then, it calculates the mean closeness of each LCP, which is the mean closeness of the two subsets, and chooses the LCP with the greatest mean closeness. It is obvious that MC is (computationally) the most expensive method.

NR1
(-13.5) **if** venous-conc is slight (12.4),
blood-conc is moderate (11.6),
art-conc is moderate (8.8),
scan-conc is normal (8.4),
cap-conc is moderate (8.4),
blood-conc is slight (8.3),
venous-conc is moderate (8.2),
venous-conc is normal (8.0),
arterial-conc is slight (-5.7),
cap-conc is slight (4.5),
blood-conc is normal (4.4),
blood-conc is high (1.6),
venous-conc is high (1.2)
then disease is inflammation

Figure 3. One of the produced neurules

3 ALTERNATIVE APPROACHES

In this section, we present two alternative approaches to splitting a non-separable training set not solely relying on closeness. The two alternative approaches will be called ALTERN-SPLIT1 and ALTERN-SPLIT2 respectively. Both of these approaches satisfy the implicit requirements mentioned in the previous section. The idea behind both approaches is simple. More specifically, they focus on the examples which are misclassified by the weights calculated by LMS and try to split the training set into two subsets: one containing the correctly classified success examples (along with all failure examples) and one containing the misclassified success examples (along with all failure examples). This process can be followed only if some (not all) success examples (and possibly failure examples) are misclassified. If all success examples are misclassified or if only failure examples are misclassified, there is no alternative but to split based on closeness. Therefore, in this process one should distinguish the following cases: (a) all of the success examples are misclassified, (b) only failure examples are misclassified, (c) only some of the success examples and none of the failure ones are misclassified, (d) failure examples and some of the success examples are misclassified. In cases (a) and (b) splitting is based on closeness. The two approaches differ only in the way of handling case (d).

More formally, approach ALTERN-SPLIT1 is as follows:

1. If all success examples are misclassified by the calculated weights, split the training set based on closeness.
2. Else, if only failure examples are misclassified, split the training set based on closeness.
3. Else, if only some of the success examples (and none of the failure examples) are misclassified, split the training set in two subsets: one containing the correctly classified success examples (along with all failure examples) and one containing the misclassified success examples (along with all failure examples).
4. Else, if failure examples and some of the success examples are misclassified, split the training set in two subsets: one containing the correctly classified success examples (along with all failure examples) and one containing the misclassified success examples (along with all failure examples).

Approach ALTERN-SPLIT2 does the same as ALTERN-SPLIT1 in steps 1, 2, 3 and handles step 4 based on closeness. It can be easily seen that ALTERN-SPLIT2 lies between CLOSENESS-SPLIT and ALTERN-SPLIT1.

4 EXPERIMENTAL RESULTS

In this section, we present various experimental results using datasets from the UCI Machine Learning Repository [15]. The experimental results involve the following aspects: (a) evaluation of the three different splitting policies based on closeness (i.e., RC, BD, MC), (b) comparison of the three approaches to splitting, CLOSENESS-SPLIT, ALTERN-SPLIT1 and ALTERN-SPLIT2 and (c) evaluation of the generalization capability of neurules and comparison with the generalization capabilities of the back propagation neural networks and the adaline unit.

Table 3. Number of neurules produced by the RC, MC and BD policies

Dataset	Condi-tions	Conclu-sions	RC	MC	BD
Monks1_train (124 patterns)	17	2	17	17	**13**
Monks2_train (169 patterns)	17	2	46	47	**38**
Monks3_train (122 patterns)	17	2	14	**11**	12
Tic-Tac-Toe (958 patterns)	27	2	26	26	**24**
Car (1728 patterns)	21	4	**151**	163	153
Nursery (12960 patterns)	27	5	830	839	**823**

Table 3 depicts experimental results for CLOSENESS-SPLIT comparing RC, MC and BD. Comparison is based on the number of neurules produced from each splitting method, shown in columns ‘RC’, ‘MC’ and ‘BD’. Column ‘Conditions’ denotes the number of conditions for each sibling neurule and column ‘Conclusions’ the number of different (final) conclusions. For the ‘monks’ datasets we used the training sets provided in the UCI Repository. Based on the results of Table 3, none of the three methods is clearly better than the others for all datasets. Further on, there is no great difference in the number of neurules produced by the three methods. BD performs better in most of the cases. RC, the simplest of the three methods, performs quite well even in the large datasets compared to the other two more complex methods. On the other hand, MC, which is computationally the most expensive method, does not perform quite well compared to the other methods to justify its use. So, BD or RC can be considered as better alternatives as far as the number of produced neurules is concerned. The number of produced neurules is the basic criterion of the comparisons, because it plays a crucial role in inference efficiency and neurule-base size.

Table 4 presents experimental results regarding ALTERN-SPLIT1 and ALTERN-SPLIT2. RC, MC and BD play a role for subsets in which splitting based on closeness is used.

Table 5 presents summary results comparing the three approaches to splitting, CLOSENESS-SPLIT, ALTERN-SPLIT1 and ALTERN-SPLIT2. Comparison is based on the minimum number of neurules produced from each method. In parentheses, the name of the splitting policy (i.e., RC, BD, MC) used, when producing the minimum number of neurules is shown. CLOSENESS-SPLIT is generally better than the other two methods. This demonstrates the effectiveness of the notion of 'closeness'. This last conclusion is further intensified by the fact that ALTERN-SPLIT2 that lies between ALTERN-SPLIT1 and CLOSENESS-SPLIT generally performs better than ALTERN-SPLIT1. The results also show that it may be worth to employ ALTERN-SPLIT1 and ALTERN-SPLIT2. A further result is that BD generally performs better than RC and MC.

Table 4. Number of neurules produced by ALTERN-SPLIT1 and ALTERN-SPLIT2

Dataset	**ALTERN-SPLIT1**			**ALTERN-SPLIT2**		
	RC	**MC**	**BD**	**RC**	**MC**	**BD**
Monks1_train	**22**	24	24	19	16	**13**
Monks2_train	34	**32**	33	43	49	**39**
Monks3_train	**15**	**15**	**15**	14	**11**	13
Tic-Tac-Toe	44	41	**40**	43	41	**38**
Car	189	171	**169**	**152**	161	154
Nursery	**1330**	1382	1378	837	842	**821**

Table 5. Number of neurules produced by CLOSENESS-SPLIT, ALTERN-SPLIT1 and ALTERN-SPLIT2

Dataset	**CLOSENESS-SPLIT**	**ALTERN-SPLIT1**	**ALTERN-SPLIT2**
Monks1_train	**13 (BD)**	22 (RC)	13 (BD)
Monks2_train	38 (BD)	**32 (MC)**	39 (BD)
Monks3_train	**11 (MC)**	15 (RC, MC, BD)	11 (MC)
Tic-Tac-Toe	**24 (BD)**	40 (BD)	38 (BD)
Car	**151 (RC)**	169 (BD)	152 (RC)
Nursery	823 (BD)	1330 (RC)	**821 (BD)**

Tables 6 and 7 present results regarding the classification accuracy (generalization) of neurules on unseen test examples. Table 6 compares the classification accuracy of neurules produced from the three splitting policies based on closeness. Table 7 compares the classification accuracy of neurules (i.e., the best result of Table 6) with the ones of the adaline unit and back-propagation neural networks. The results for each dataset (except for the three monks datasets) were produced by using 75% of the examples as training set and 25% of the examples as testing set in four different runs. Needless to say that the training examples in the test sets were not included in the training sets. Different and disjoint test sets were used in each run, so that the union of the four test sets formed the whole dataset. The classification accuracy was computed as the mean value of the accuracies obtained from the four runs. For 'monks1' and 'monks2' datasets this procedure for creating training and test sets was applied to the corresponding test sets of 432 training examples available in the UCI repository. For the 'monks3' dataset, the training and test set available in the UCI repository were used since the training set is reported to contain noise. It should be mentioned that we were not able to construct a back-propagation neural network for the 'Nursery' dataset with competitive generalization capability.

For the training of back-propagation neural networks, the standard back-propagation algorithm was employed using a momentum in adjusting the weights and one layer of hidden nodes. The values of these three back-propagation parameters along with the average error threshold were tuned separately for the training sets of each dataset after a number of experiments (based on error-and-trial). Training stopped when either the number of training epochs reached an upper threshold or the average squared error became less than or equal to the average error threshold. Furthermore, no cross-validation was used when training the adaline unit, the neurules or the back-propagation neural network (perhaps with cross-validation the results of Table 7 for all approaches would have been slightly better). Also, if the activations of multiple output nodes exceeded 0.5 (when a test example was given as input), then the example took the category of the most active output node (i.e., the one with the greatest activation) [20].

Table 6. Generalization of neurules produced from RC, MC, BD policies

Dataset	**RC**	**MC**	**BD**
Monks1	100%	100%	100%
Monks2	96.30%	96.99%	**97.92%**
Monks3	92.36%	93.52%	**96.06%**
Tic-Tac-Toe	**98.85%**	97.50%	98.12%
Car	94.44%	**94.56%**	94.50%
Nursery	**99.63%**	99.53%	99.52%

Table 7. Generalization of adaline unit, neurules and back-propagation neural network

Dataset	**Adaline Unit**	**Neurules**	**BPNN**
Monks1	67.82%	100%	100%
Monks2	43.75%	97.92%	100%
Monks3	92.13%	96.06%	97.22%
Tic-Tac-Toe	61.90%	98.85%	98.23%
Car	78.93%	94.56%	95.72%
Nursery	82.26%	99.63%	

The results in Tables 6 and 7 show that neurules generalize quite well. Table 6 shows that none of the three splitting policies performs better than the others in all datasets. Comparing the results of Table 5 and Table 6, it can be said that it is not unlikely that a splitting policy may generalize better than the other policies although it produced a greater number of neurules. Table 7 shows that neurules outperform the adaline unit and are worse than back-propagation neural networks. These results are very promising. It was expected that the generalization capability of neurules would be somewhere between the adaline unit and the back-propagation neural network. This is due to the nature of the three approaches: the adaline unit is a single unit for performing classification, a neurule base consists of a number of autonomous adaline units (neurules) and a back-propagation neural network is a multi-layer network containing hidden nodes useful for the computation of non-separable functions.

A parameter not shown in Table 7 involves the total effort in constructing the corresponding knowledge base. The construction of a neurule base is easier than the construction of a back-propagation neural network. When constructing neurules, one should only try out the different splitting approaches. So, construction of neurules is straightforward. On the other hand, in the case of a back-propagation neural network, one should simultaneously adjust three different parameters (based on error-and-trial): the number of hidden nodes (assuming one hidden layer), the learning rate and the momentum. The number of hidden nodes is an integer, whereas the learning rate and the momentum are real numbers lying between 0.0 and 1.0. Simultaneously adjusting those three parameters can be a non-trivial and time-consuming task. However, the adjustment of those parameters plays an important role in the classification accuracy of the neural network regarding the training and test sets.

It should be also mentioned that when we developed a method for producing neurules from training examples [10], we did not have generalization as our primary intention. Our effort was to develop an alternative method to the one producing neurules through conversion from existing symbolic rule bases [9]. In this way, the knowledge acquisition process is facilitated since neurules can be constructed from two alternative sources, existing symbolic rule bases and training examples. However, according to the results of this paper, regarding generalization capability of neurules, neurules could be a choice in applications with available training examples and in which naturalness, modularity of the knowledge base and provision of interactive inference and explanation mechanisms are desirable factors besides generalization. Obviously in applications in which generalization is the only concern, one should choose back-propagation neural networks.

5 CONCLUSIONS

In this paper, we investigate previously unexplored aspects regarding the construction of neurules from training examples. Results validate our initial choice, demonstrating the effectiveness of solely using the notion of ‘closeness’ to handle non-separable training sets. Alternative splitting approaches performed worse. Furthermore, experimental results show that neurules generalize quite well even compared to back-propagation neural networks. Our future research will involve investigation of possible improvements to the construction and generalization capability of neurules.

REFERENCES

[1] R. Andrews, J. Diederich and A. Tickle, ‘A survey and critique for extracting rules from trained ANN’, *Knowledge-Based Systems*, **8**, 373-389, (1995).

[2] A.S. d’Avila Garcez, K. Broda, D.M. Gabbay, ‘Symbolic knowledge extraction from trained neural networks: A sound approach’, Artificial Intelligence 125, 155-207, 2001.

[3] S. Bader and P. Hitzler, ‘Dimensions of neural-symbolic integration – a structured survey’, In S. Artemov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb, J. Woods, *We Will Show Them: Essays in Honour of Dov Gabbay*, International Federation for Computational Logic, College Publications, volume 1, 167-194, 2005.

[4] I. Cloete, J. M. Zurada (eds.), *Knowledge-Based Neurocomputing*, MIT Press, 2000.

[5] A.S. d’Avila Garcez, K. Broda, D.M. Gabbay, *Neural-symbolic Learning Systems: Foundations and Applications, Perspectives in Neural Computing*, Springer-Verlag, Heidelberg, 2002.

[6] L-M Fu, *Neural Networks in Computer Intelligence*, McGraw-Hill, 1994.

[7] S.I. Gallant, *Neural Network Learning and Expert Systems*, MIT Press, 1993.

[8] A.Z. Ghalwash, ‘A Recency Inference Engine for Connectionist Knowledge Bases’, *Applied Intelligence*, **9**, 201-215, (1998).

[9] I. Hatzilygeroudis and J. Prentzas, ‘Neurules: Improving the Performance of Symbolic Rules’, *International Journal on AI Tools*, **9**, 113-130, (2000).

[10] I. Hatzilygeroudis and J. Prentzas, ‘Constructing Modular Hybrid Knowledge Bases for Expert Systems’, *International Journal on AI Tools*, **10**, 87-105, (2001).

[11] I. Hatzilygeroudis and J. Prentzas, ‘An Efficient Hybrid Rule Based Inference Engine with Explanation Capability’, Proceedings of the 14th International FLAIRS Conference, Key West, FL, 227-231, (2001).

[12] I. Hatzilygeroudis and J. Prentzas, ‘Neuro-Symbolic Approaches for Knowledge Representation in Expert Systems’, *International Journal of Hybrid Intelligent Systems*, **1**, 111-126, (2004).

[13] M. Hilario, ‘An Overview of Strategies for Neurosymbolic Integration’, in [16].

[14] K. McGarry, S. Wertmer, and J. MacIntyre, ‘Hybrid neural systems: from simple coupling to fully integrated neural networks’, *Neural Computing Surveys*, **2**, 62-93, (1999).

[15] L.R. Medsker, *Hybrid Neural Networks and Expert Systems*, Kluwer Academic Publishers, Boston, 1994.

[16] D.J. Newman, S. Hettich, C.L. Blake, C.J. Merz, ‘UCI Repository of machine learning databases’ [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA, University of California, Department of Information and Computer Science (1998).

[17] J. Prentzas, I. Hatzilygeroudis and A. Tsakalidis, ‘Updating a Hybrid Rule Base with New Empirical Source Knowledge’, Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence, Washington, DC, USA, 9-15, (2002).

[18] J. Prentzas and I. Hatzilygeroudis, ‘Rule-based Update Methods for a Hybrid Rule Base’, *Data and Knowledge Engineering*, **55**, 103-128, (2005).

[19] R. Sun and E. Alexandre (eds), *Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, Lawrence Erlbaum, 1997.

[20] G. Towell and J. Shavlik, ‘Knowledge-Based Artificial Neural Networks’, *Artificial Intelligence* **70(1-2)**, 119-165, (1994).

[21] S. Wermter and R. Sun (eds), *Hybrid Neural Systems*, Springer-Verlag, Heidelberg, 2000.

Towards the integration of abduction and induction in artificial neural networks

Oliver Ray[1] and **Artur d'Avila Garcez**[2]

Abstract. This paper presents a method for realising abduction in artificial neural networks (ANNs) by generalising existing neuro-symbolic approaches from normal logic programs to abductive logic programs (ALPs) in order to provide a more expressive formalism for representing and reasoning about partial knowledge and integrity constraints. The aim is to develop a massively-parallel technique for abduction that can also be integrated with standard connectionist learning approaches to offer more control over which assumptions can and cannot be made in learning. Existing methods for abduction in neural networks are not well suited to this task as they only apply to a restricted a class of abduction problems or they do not adequately address the problem of computing multiple solutions. By contrast, this paper proposes an approach for translating ALPs into ANNs whereby no restrictions are imposed on the underlying programs and, if required, the network can be made to systematically compute all abductive explanations or provide a guarantee when none exist. Moreover, since the topology of the network mirrors the structure of the program, it can be acquired and revised by standard neuro-symbolic training techniques and can also be exploited to impose a preference on the order in which the solutions are found.

1 Introduction

Neurosymbolic integration [9, 6] aims to combine the advantages of artificial neural networks (ANNs) and logic programs by providing practical methods of learning that use declarative representations of knowledge. This is done by translating logic programs into neural networks: either to yield an initial network which can be trained on further data with techniques such as back-propagation as in [20]; or to compute the consequences of the program under the stable model semantics by means of massively parallel deduction as in [8]. But, normal logic programs are not especially suited for representing and reasoning about partial knowledge that is inherent in learning; and this limitation motivates the study of more expressive formalisms for dealing with uncertainty.

Abductive logic programs (ALPs) [10] are an extension of normal logic programs that are more suitable for handling incomplete knowledge. In particular, they allow the truth or falsity of some ground literals, known as *abducibles*, to be left unspecified subject to given integrity constraints. Unlike normal logic programming, abductive proof procedures are free to assume any consistent set of abducibles when solving a goal. Thus, abduction does not merely determine whether a given goal follows from a program, but computes a set of assumptions which, when added to the program, ensure that the goal succeeds. Each set of abducibles is called an *abductive explanation* and represents an extension of the program that is referred to as a *generalised stable model* [11]. By extending the program in this way, abduction can extrapolate potentially useful assumptions from partially complete theories.

The incompleteness of knowledge inherent in learning suggests inductive techniques may benefit from a facility for abduction. This claim is supported by logic-based machine learning systems which have recently shown that abduction and induction can be combined to achieve superior reasoning capabilities, as shown in [15, 12, 4]. The benefits offered by neural networks over logical approaches in terms of noise-tolerance and massive-parallelism provide an even greater incentive to investigate the integration of abduction and induction at the subsymbolic level. But, existing methods for abduction in neural networks are not well suited to this task as they only apply to a very restricted a class of abduction problems whose expressivity is limited to definite acyclic programs [7, 18, 2, 22] or they do not adequately address the problem of computing multiple solutions [13, 21, 14, 1]. In this work we seek to demonstrate the importance of abductive reasoning in the neurosymbolic context and to set the scene for the subsymbolic integration of abduction and induction.

This paper presents a novel methodology for abduction in neural networks by generalising existing neurosymbolic approaches from normal logic programs to abductive logic programs. This provides a formalism for expressing uncertainty and querying programs with more than one stable model. An algorithm is given for translating ALPs into ANNs such that the fixpoints of the network represent the generalised stable models of the program. The translation is introduced in three steps. First, a function θ is defined that maps logic programs into ANNs by adapting existing neurosymbolic encodings. Second, a function ϕ is defined that maps acyclic ALPs into ANNs by extending the program with some additional clauses for abduction. Third, a function ψ is defined that maps arbitrary ALPs into ANNs using a preprocessing transformation which allows positive and negative cycles to be uniformly handled through abduction.

The paper is structured as follows. Section 2 recalls some notation and terminology relating to neural networks and logic programs and it introduces the task of ALP. As this paper does not directly address the problems of learning or extracting of programs from networks, it is sufficient to only consider networks of binary threshold neurons. Section 3 defines the functions θ and ϕ and shows how the networks they produce can compute the generalised stable models of acyclic abductive logic programs. Section 4 then shows how the approach is extended to abductive logic programs with positive and negative cycles. The paper concludes with a summary and directions for future work. All of the examples have been implemented and tested using the *BrainBox* neural network simulator [5] and the configuration files may be downloaded from [16].

[1] Imperial College London, UK, email: or@doc.ic.ac.uk
[2] City University London, UK, email: aag@soi.city.ac.uk

2 Background

(Threshold) Neural Networks: A *neural network*, or just *network* hereafter, is a graph (N, E) whose nodes N are called *neurons* and whose edges $E \subseteq N \times N$ are called *connections*. Each neuron $n \in N$ is labeled with a real number $t(n)$ called its *threshold* and each connection $(n, m) \in E$ is labeled with a real number $w(n, m)$ called its *weight*. The *state* of a network is a function s that assigns to each neuron the value 0 or 1. A neuron is said to be *active* if its state is 1 and it is said to be *inactive* if its state is 0. For each state s there is a unique successor state s' such that a neuron n is active in s' iff its threshold is exceeded by the sum of the weights on the connections coming into n from nodes which are active in s. A network is said to be *relaxed* iff all of its neurons are inactive. A *fixpoint* of the network is any state that is identical to its own successor state. If a fixpoint t is reachable from an initial state s by repeatedly computing successor states, then t is referred to as the *fixpoint of* s.

Normal Logic Programs: A *rule* is an expression of the form $H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m$, where the H, B_i and C_j are all atoms. The atom to the left of the arrow is called *head* of the rule, while the literals to the right comprise the *body*. The head atom H and the positive body atoms B_i are said to occur *positively* in the rule, while the negated body atoms C_j are said to occur *negatively*. A rule with no negative body literals is called a *definite clause* and written $H \leftarrow B_1, \ldots, B_n$. A rule with no body literals at all is called a *fact* and written H. A *normal logic program*, or just *program* hereafter, is a set of rules. If P is a program, then $\mathcal{B}_P$ (the *Herbrand base* of P) is the set of all atoms built from the predicate and function symbols in P; and $\mathcal{G}_P$ (the *ground expansion* of P) is the program comprising all ground instances of the clauses in P. In additon, $\mathcal{A}_p^+$ and $\mathcal{A}_P^-$ denote, respectively, the sets of ground atoms that occur positively and negatively in $\mathcal{G}_P$; and $\mathcal{D}_P$ (the *dependency graph* of P) is the directed graph with signed edges whose nodes are the atoms in $\mathcal{A}_p^+ \cup \mathcal{A}_P^-$ and where there is a positive (resp. negative) edge from a to b iff there is a clause in $\mathcal{G}_P$ with a in the head and b occurring positively (resp. negatively) in the body. A cycle in $\mathcal{D}_P$ is *positive* if it has no negative edges and is *negative* otherwise. A program P is said to be *acyclic* iff $\mathcal{D}_P$ contains no (positive or negative) cycles. A *stable model* of P is a Herbrand interpretation $I \subseteq \mathcal{B}_P$ that coincides with the least Herbrand model of the definite program P^I obtained by removing from $\mathcal{G}_P$ each rule containing a negative literal not satisfied in I, and by deleting all of the negative literals in the remaining rules.

Abductive Logic Programs: An *abductive logic program* [10] is a triple (T, IC, A) where T is a program (the *theory*), IC is a set of rules (*integrity constraints*) with the head atom $\perp$ (denoting logical falsity), and A is a set of ground atoms (*abducibles*). Given a set G of ground atoms (the *goals*), the task of ALP is to compute a set $\Delta \subseteq A$ of abducibles such that G and IC are satisfied in some stable model of $T \cup \Delta$. In the terminology of [11], the goal G is said to be satisfied in a *generalised stable model* of T, and Δ is said to be an *abductive explanation* of G with respect to T, IC and A.

To select between alternative explanations, additional preference criteria are often utilised. The most widely-used criterion is that of *minimality* [10], which intuitively means the none of the atoms in the abductive explanation are redundant (i.e. there is no strict sub-explanation). Formally, an explanation Δ of G with respect to (T, IC, A) is *minimal* iff there is no $\Delta' \subset \Delta$ such that Δ' is an explanation of G, For convenience the four inputs (T, G, IC, A) are collectively called an *abductive context*. An abductive context is said to be definite, acyclic, etc, iff the theory T is definite, acyclic, etc.

Definition 2.1 (Abductive Context). *An abductive context is a four-tuple (T, G, IC, A) where T is set of rules, G and A are sets of ground atoms, and IC is a set of integrity constraints.*

Example 2.1. *Consider the abductive context below describing an old car. The theory states that the car wont start if its battery is flat or its fuel tank is empty; that the battery is flat on wet days; that the car will overheat if its fan is broken; and that the lights of the car are on. The integrity constraint states that the lights cannot be on at the same time the battery is flat. The goal to that must be proved is $wont\_start$. The abducibles which may be assumed are wet_day, fan_broke, $fuel\_empty$.*

$$
\begin{aligned}
T &= \left\{ \begin{array}{l} wont\_start \leftarrow battery\_flat \\ wont\_start \leftarrow fuel\_empty \\ battery\_flat \leftarrow wet\_day \\ overheat \leftarrow fan\_broke \\ lights\_on \end{array} \right\} \\
G &= \{\ wont\_start\ \} \\
IC &= \{\ \perp \leftarrow battery\_flat, lights\_on\ \} \\
A &= \{\ fan\_broke, fuel\_empty, wet\_day\ \}
\end{aligned}
$$

There are two abuctive explanations of this context: $\Delta_1 = \{fuel\_empty\}$ and $\Delta_2 = \{fan\_broke, fuel\_empty\}$. The former is minimal but the latter not (as it is a superset of the former). These are the only correct explanations since all other sets of abducibles fail to satisfy either the goal or the integrity constraints.

3 Neural Network Abuction: Simple Case

This section presents a first methodology for realising abduction in neural networks by defining a translation which maps definite acyclic abductive logic programs into networks whose fixpoints correspond to the generalised stable models of the program. The initial restriction to acyclic programs is merely to simplify the presentation of the key ideas and is immediately lifted in the next section through some simple syntactic preprocessing of the inputs.

The proposed methodology builds upon existing neurosymbolic techniques for transforming logic programs into neural networks and is easily adapted to suit any choice of encoding. In this paper, for ease of exposition, we introduce a translation based on multi-layer threshold networks, which combines the approaches in [20, 8] and is easily generalised to the recurrent sigmoidal networks using the techniques in [6] to allow backpropagation learning.

As formalised in Definition 3.1 below, the neural network $\theta(P)$ corresponding to a normal program P is obtained from the ground expansion $\mathcal{G}_P$ of P by adding the following nodes and edges for each rule r of the form $H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m$ in $\mathcal{G}_P$:

- a node with threshold $n - 1/2$ to represent the rule r
- a node with threshold $1/2$ for each atom H,B_i,C_j in the rule (which has not already been added through an earlier rule)
- an edge with weight 1 from r to the head atom H
- an edge with weight 1 from each unnegated body atom B_i to r
- an edge with weight -1 from each negated body atom C_j to r

Definition 3.1 (θ)**.** *If P is a program, then $\theta(P)$ is the network (N, E) such that*

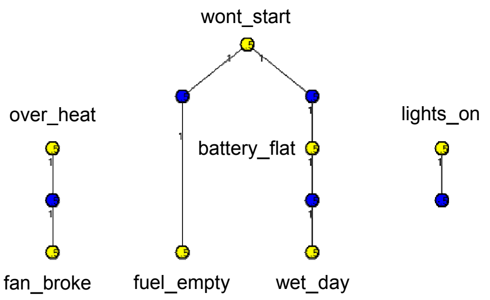
$$N = \bigcup_{r \in \mathcal{G}_P} \left\{ \begin{array}{l} r, H, B_1, \ldots, B_n, C_1, \ldots, C_m \\ \mid r = H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \end{array} \right\}$$

$$E = \bigcup_{r \in \mathcal{G}_P} \left\{ \begin{array}{l} (r, H), (B_1, r), \ldots, (B_n, r), (C_1, r), \ldots, (C_m, r) \\ \mid r = H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \end{array} \right\}$$

and for all $r = H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \in \mathcal{G}_P$

$$\begin{array}{lll} & t(H) = 1/2 & w(r,H) = 1 \\ t(r) = n - 1/2 & t(B_i) = 1/2 & w(B_i, r) = 1 \\ & t(C_j) = 1/2 & w(C_j, r) = -1 \end{array}$$

Example 3.1. *If P is the program T in Example 2.1 above, then $\theta(P)$ is the network below. For convenience, nodes representing atoms are lightly shaded and are annotated with the name of the atom, while nodes corresponding to the rules in the program are darkly shaded.*



The above translation produces a neural network encoding of a given program. In common with other approaches, it can be shown that if the program is acyclic, then the fixpoint of the relaxed network exists and corresponds to the unique stable model of the program. But, in order to perform abduction, this procedure must be supplemented with some way of representing goals, integrity constraints and some means of activating and evaluating different combinations of abducibles. As formalised in Definition 3.2 below, the required abductive machinery can be obtained by transforming an abductive context (T, G, IC, A) into a logic program with one set of clauses $(T' \cup G' \cup IC' \cup A')$ representing the context and another set of clauses $(C \cup K \cup L)$ representing some additional logic to ensure the fixpoints of the network correspond to the generalised stable models of the theory.

Definition 3.2 (ϕ)**.** *Let (T, G, IC, A) be an abductive context. Let N be the number of abducibles in A. Let P be the length of the longest directed path in $\mathcal{D}_T$ with no repeated nodes. Let M be the smallest integer greater than or equal to $\frac{1}{2}(P + 2N + 3)$. Let goal, ic, soln, next, done, sync, nogood, hold, a_i, b_i, c_i, d_i and k_j be propositions not appearing in (T, G, IC, A) for all $0 \leq i \leq N$ and for all $0 \leq j \leq M$. Then $\phi(T, G, IC, A)$ is the network $\theta(T' \cup G' \cup IC' \cup A' \cup C \cup K \cup L)$ where*

$$\begin{array}{lll} T' & = & T \\ G' & = & \{goal \leftarrow B_1, \ldots, B_n \mid \{B_1, \ldots B_n\} = G\} \\ IC' & = & \{ic \leftarrow L_1, \ldots, L_m \mid \bot \leftarrow L_1, \ldots, L_m \in IC\} \\ A' & = & \{A_i \leftarrow a_i \mid A_i \in A\} \end{array}$$

$$C = \bigcup_{i=1}^{N} \left\{ \begin{array}{l} a_i \leftarrow a_i, \neg c_i \\ a_i \leftarrow d_i \\ b_i \leftarrow a_i \\ c_i \leftarrow b_{i-1}, \neg a_{i-1}, a_i \\ d_i \leftarrow b_{i-1}, \neg a_{i-1}, \neg a_i \end{array} \right\} \cup \left\{ \begin{array}{l} b_0 \leftarrow next \\ done \leftarrow b_N, \neg a_N \\ done \leftarrow done \end{array} \right\}$$

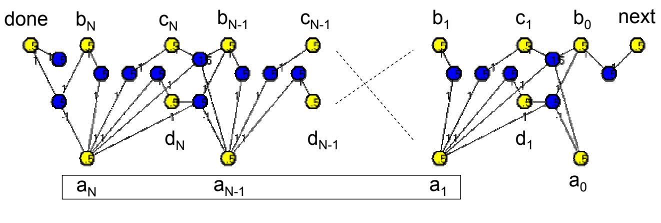
$$K = \bigcup_{i=1}^{M} \{\ k_i \leftarrow k_{i-1}\ \} \cup \left\{ \begin{array}{l} k_0 \leftarrow \neg hold, \neg k_M \\ sync \leftarrow k_0, \neg k_1 \end{array} \right\}$$

$$L = \left\{ \begin{array}{l} nogood \leftarrow ic \\ nogood \leftarrow \neg goal \\ soln \leftarrow sync, \neg nogood \\ soln \leftarrow soln, \neg nogood \\ hold \leftarrow soln \\ hold \leftarrow done \\ next \leftarrow sync, nogood \end{array} \right\}$$

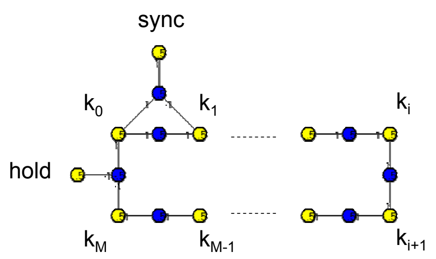
The four theories T',G',IC' and A' are a representation of the abductive context in which $goal$ is true when the goal is satisfied, ic is true when an integrity constraint is violated, and each abducible A_i is true when the corresponding atom a_i is true. More formally, T' is the theory T, G' comprises a single clause with $goal$ in the head and the atoms of G in the body, IC' is obtained by inserting ic into the head of each constraint in IC, and A' contains one clause of the form $A_i \leftarrow a_i$ for each abducible $A_i \in A = \{A_1, \ldots, A_n\}$.

The last three theories C, K and L denote some control logic for activating different combinations of abducibles until an explanation is found or all possibilities are exhausted. When a solution is found, the network will enter a stable state in which $soln$ is activated and the a_i indicate which abducibles are in the explanation. If $next$ is briefly activated (for two consecutive time points), the network will leave this stable state and look for the next solution. Once all possibilities have been tried, the network will enter a stable state in which $done$ is activated.

The theory C represents a binary counter whose outputs $a_N a_{N-1} \ldots a_1$ each drive one abducible. The network encoding of C is shown below. The counter advances each time the node $next$ is briefly activated and it activates the node $done$ when the counter overflows. Each bit of the counter uses four nodes, a_i, b_i, c_i and d_i, to implement a divide by two register that toggles the state of a_i whenever the state of a_{i-1} changes from on to off – with the nodes c_i and d_i signalling a_i to turn off and on, respectively.



The theory K represents a clock whose output $sync$ is used to advance the counter if the current state is not a solution. The network encoding of K is shown below. The nodes k_i form a loop where the state of each one follows that of its predecessor; except for the first, which opposes the last. The period of the clock is proportional to the number of nodes $M + 1$, which is chosen to give the rest of the network sufficient time to stabilise between successive signals. The clock is disabled when $hold$ is active. The output $sync$ is active when k_0 is on but k_1 is not – which is true for 2 consecutive time points out of every 4(M+1).



The theory L represents some simple control logic that uses $sync$ to advance the counter or to suspend the clock according to whether the current abducibles are a valid explanation. The state of $nogood$ indicates when the goal is not satisfied or one of the integrity constraints is violated. When $sync$ becomes active, either $next$ or $soln$ will be activated depending on the state of $nogood$. The first case will advance the network into the next state while the second will force the network to stabilise.

Example 3.2. *If (T, G, IC, A) is the context in Example 2.1 above, then $\phi(T, G, IC, A)$ is the network shown in Figure 1(a). The theories T', G', IC', A' are shown below. There are $N = 3$ abducibles in A and the longest simple path in $\mathcal{G}_T$ is $(wet\_day, fuel\_empty, fan\_broke)$ with length $P = 3$. The least upper bound of $\frac{1}{2}(P + 2N + 3)$ is $M = 6$.*

$$\begin{aligned}
T' &= T \\
G' &= \{goal \leftarrow wont\_start\} \\
IC' &= \{ic \leftarrow battery\_flat, lights\_on\} \\
A' &= \left\{ \begin{array}{l} fan\_broke \leftarrow a_1 \\ fuel\_empty \leftarrow a_2 \\ wet\_day \leftarrow a_3 \end{array} \right\}
\end{aligned}$$

For any acyclic abductive context (T, G, IC, A) it can be shown that the fixpoint of the relaxed network $\phi(T, G, IC, A)$ exists and is computed in a finite time. If $soln$ is active in the fixpoint, then that state represents a generalised stable model of T that satisfies G and IC, where the hypothesis Δ consists of the active abducibles. All other solutions can be computed by briefly activating the neuron $next$ to force the network to search for the next stable state, which also exists and is computed in finite time. If $done$ is active, then no further solutions exist.

In Example 3.2 above, it can be verified[3] that the initially relaxed network rejects the initial hypothesis $H = \{fan\_broke\}$ (which does not satisfy the goal) and converges instead to the solution $\Delta_1 = \{flat\_battery\}$. If a signal is then applied to $next$, the network will converge to the next solution $\Delta_2 = \{flat\_battery, fan\_broke\}$. If another signal is applied to $next$, the network will reject all other hypotheses and converge to the final $done$ state, indicating that no other solutions exist for this context.

4 Neural Network Abduction: General Case

This section shows how the methodology introduced above can be extended to abductive logic programs with cycles using a simple preprocessing transformation. But, before doing so, it is instructive to illustrate why programs with cycles are potentially problematic.

[3] The reader can use the software available from [5] with the data at [16] to run the network in Fig 1(a) by holding down ctrl-F1 to advance the network one time point and double clicking neuron 98 to apply a signal to $next$. Note that the data file contains some redundant neurons which merely serve to ensure that the connections between neurons follow the same easy-to-read layout as shown in the figures above.

First consider positive cycles by supposing that the rule $fan\_broke \leftarrow over\_heat$ is added to T in Example 2.1 and the constraint $\bot \leftarrow over\_heat$ is added to IC. The problem is that the cycle between fan_broke and $over\_heat$ introduces a memory into the network that causes a permanent violation of integrity. Once $over\_heat$ is activated by fan_broke, they both remain high, and so does ic. Hence, the correct solution Δ_1 would be rejected due to the memory of the violation caused by the initial hypothesis H.

One solution to this problem is to relax the sub-networks T', G' IC' and A' after each set of abducibles is tried. This can easily be achieved by adding a special abducible $true$ to the body of each rule that is always connected to the least significant bit a_1 of the counter to ensure that its state is continuously alternating with respect to the other abducibles. In this way, any self-sustaining loops are systematically deactivated before the next set of abducibles is presented to the network.

Next consider negative cycles by supposing that the rules $door\_open \leftarrow \neg door\_closed$ and $door\_closed \leftarrow \neg door\_open$ are added to T in Example 2.1 and the atom $door\_open$ is added to G. The problem is that the cycle between $door\_open$ and $door\_closed$ introduces an instability into the network that prevents any fixpoint being reached from the initially relaxed state. Instead of converging to a stable state in which $door\_open$ is active and $door\_closed$ is inactive, these atoms continually force each other to change state.

Following [3], one answer to this problem involves re-writing negative literals as positive abducibles and implementing negation through abduction. This is achieved by introducing a new abducible predicate p_i^* to denote the negation $\neg p_i$ of each predicate p_i in the context and adding integrity constraints to ensure that for any ground terms $t_1, \ldots, t_n$ exactly one of $p(t_1, \ldots, t_n)$ and $p^*(t_1, \ldots, t_n)$ is true. As shown in [11], there is a 1-1 correspondence between the generalised stable models of the original and transformed contexts.

These solutions are implemented together in Definition 4.1 below, which transforms an arbitrary context (T, G, IC, A) into a definite context (T'', G'', IC'', A'') before using ϕ to generate the network. Since the latter context is definite, there are no potential instabilities in the network caused by negative cycles; and assuming that ϕ maps $true$ to a_1, there will be no residual memory in the network caused by positive cycles. Thus, it can be shown that the stable states of $\phi(T'', G'', IC'', A'')$ reachable from the relaxed state by applying signals to $next$ are the generalised stable models of (T, G, IC, A).

Definition 4.1 (ψ)**.** *Let (T, G, IC, A) be an abductive context not containing the proposition $true$. Let $R = \{p_1, \ldots, p_k\}$ be the set of predicates p_i appearing in (T, G, IC, A) and let $S = \{p_1^*, \ldots, p_k^*\}$ be a set of predicates p_i^* not appearing in (T, G, IC, A). For each atom C of the form $p_i(t_1, \ldots, t_n)$, let C^* denote the atom $p_i^*(t_1, \ldots, t_n)$. Recall that $\mathcal{A}^-_{T \cup IC}$ denotes the set of atoms that appear negated in the ground expansion of the program $T \cup IC$. Then $\psi(T, G, IC, A)$ is the network $\phi(T'', G'', IC'', A'')$ such that*

$$\begin{aligned}
T'' &= \left\{ \begin{array}{l} H \leftarrow true, B_1, \ldots, B_n, C_1^*, \ldots, C_m^* \\ \mid H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \in T \end{array} \right\} \\
G'' &= G \cup \{true\} \\
IC'' &= \left\{ \begin{array}{l} \bot \leftarrow B_1, \ldots, B_n, C_1^*, \ldots, C_m^* \\ \mid \bot \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \in IC \end{array} \right\} \\
&\cup \ \left\{ \ \bot \leftarrow C, C^* \mid C \in \mathcal{A}^-_{T \cup IC} \ \right\} \\
&\cup \ \left\{ \ \bot \leftarrow \neg C, \neg C^* \mid C \in \mathcal{A}^-_{T \cup IC} \ \right\} \\
A'' &= A \cup \{true\} \cup \{C^* \mid C \in \mathcal{A}^-_{T \cup IC}\}
\end{aligned}$$

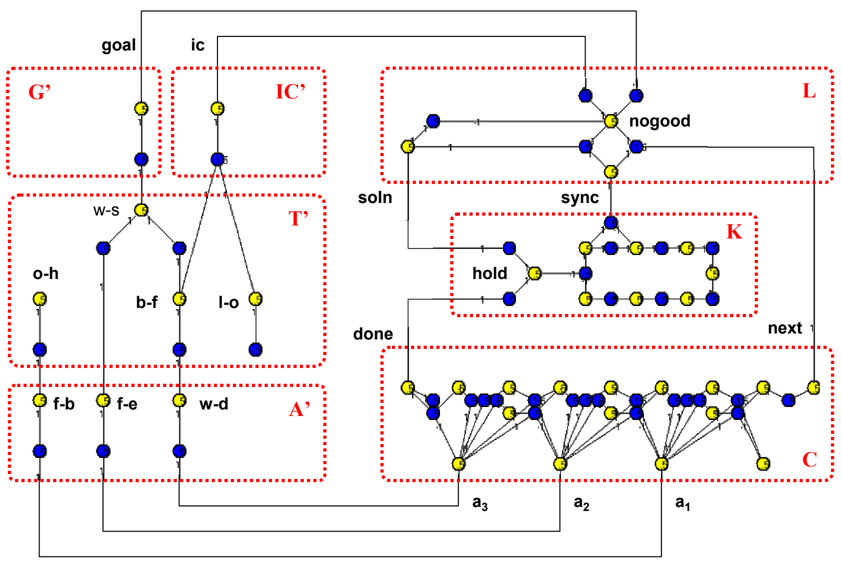


Figure 1(a) Example 3.2

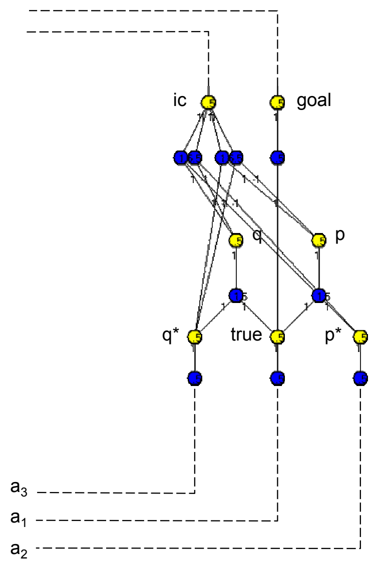


Figure 1(b) Example 4.2

Example 4.1. *Consider the context obtained by extending Example 2.1 as described above: with one clause $fan\_broke \leftarrow over\_heat$ stating that the fan will break if the car overheats; with two clauses $door\_open \leftarrow \neg door\_closed$ and $door\_closed \leftarrow \neg door\_open$ stating that the car door is open if it is not closed and vive versa; with one goal $door\_open$; and with one constraint $\bot \leftarrow over\_heat$. The theories T'', G'', IC'' and A'' obtained by applying Definition 4.1 to this extended context are shown below.*

$$T'' = \left\{ \begin{array}{l} wont\_start \leftarrow true, battery\_flat \\ wont\_start \leftarrow true, fuel\_empty \\ battery\_flat \leftarrow true, wet\_day \\ overheat \leftarrow true, fan\_broke \\ fan\_broke \leftarrow true, over\_heat \\ door\_open \leftarrow true, door\_closed^* \\ door\_closed \leftarrow true, door\_open^* \\ lights\_on \leftarrow true \end{array} \right\}$$

$$G'' = \{ \; wont\_start, door\_open, true \; \}$$

$$IC'' = \left\{ \begin{array}{l} \bot \leftarrow battery\_flat, lights\_on \\ \bot \leftarrow over\_heat \\ \bot \leftarrow door\_open, door\_open^* \\ \bot \leftarrow door\_closed, door\_closed^* \\ \bot \leftarrow \neg door\_open, \neg door\_open^* \\ \bot \leftarrow \neg door\_closed, \neg door\_closed^* \end{array} \right\}$$

$$A'' = \left\{ \begin{array}{l} fan\_broke, fuel\_empty, wet\_day, \\ door\_closed^*, door\_open^*, true \end{array} \right\}$$

For lack of space, the network $\psi(T, G, IC, A)$ is not shown. However, the reader can easily verify that the relaxed network converges to a fixpoint in which exactly three abducibles are activated: $fuel\_empty$, $door\_closed^*$ and $true$. This implies that G and IC are satisfied in a stable model of the program obtained by adding the hypothesis $\Delta = \{fuel\_empty\}$ to the extended theory. Moreover, if a signal is applied to $next$, the network will converge to the done state, indicating that no other solutions exist for this context.

The approach described above comprises a sound and complete method for solving ALPs in ANNs. It is interesting to distinguish two special cases of this problem which are of practical importance: first, given a context in which IC and A are both empty, ALP reduces to the problem of deciding whether G follows from T; second, given a context in which G, IC and A are all empty, ALP reduces to the problem of computing the stable models of T. It is instructive to consider a classic example of this latter problem.

Example 4.2. *Consider the following abductive context:*

$$\left(\left\{ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \right\}, \emptyset, \emptyset, \emptyset \right)$$

As remarked previously, solving this context amounts to computing the stable models of the following program:

$$P = \left\{ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \right\}$$

As observed in [8], this program is not easily handled by many other approaches as it has two stable models: $\{q\}$ and $\{p\}$. Applying ψ to this context results in the transformed context below and the sub-network shown in Figure 1(b) above. [4]

$$\left(\left\{ \begin{array}{l} p \leftarrow q*, true \\ q \leftarrow p*, true \end{array} \right\}, \{ \, true \, \}, \left\{ \begin{array}{l} \leftarrow p, p* \\ \leftarrow q, q* \\ \leftarrow \neg p, \neg p* \\ \leftarrow \neg q, \neg q* \end{array} \right\}, \left\{ \begin{array}{c} p* \\ q* \\ true \end{array} \right\} \right)$$

The reader can verify that the relaxed network converges into a stable state where q, $p*$ and $true$ alone are active – corresponding to the stable model $\{q\}$. Applying a signal to $next$ forces the network to converge to the next stable state where p, $q*$ and $true$ alone are active – corresponding to the stable model $\{p\}$. Applying another signal to $next$ results in the network converging to the final $done$ state – indicating that these are the only two models.

[4] Note that the network representation of C, K and L is not shown because it is identical to that given in Figure 1(a).

5 Conclusion and Future Work

This paper presented a method for abductive reasoning in neural networks. In particular, it proposed an algorithm for translating abductive logic programs into neural networks so that abductive inference can benefit from the massive neural parallelism. The methodology extends the original program with some additional logic to ensure that the fixpoints of the network correspond to the (generalised stable) models of the (abductive logic) program. It also uses a well-known relationship between negation and abduction in order to correctly handle programs with positive and negative cycles. In contrast to earlier work, no restrictions are placed on the programs and, if required, the network can be made to enumerate all explanations. Moreover, because our methodology is a generalisation of existing neurosymbolic techniques, we believe it can be more easily combined with standard learning approaches. In this way, we see our approach as a first step towards the principled subsymbolic integration of abduction and induction – which could eventually have implications in cognitive modelling and scientific discovery.

The approach presented in this paper can be improved in several ways. For example, it is possible to implement the counter using only half the neurons and half the propagation delay per bit. Also, instead of making $true$ an abducible, it is sufficient to define $true \leftarrow \neg next$ in order to relax the program sub-network before each new set of abducibles is presented. In the case of Example 4.1, these optimisations alone reduce the number of time points needed to search the entire hypothesis space from 2560 to 896. In addition it is possible to dispense with the clock and issue a synchronisation pulse by detecting when the program sub-network reaches its fixpoint. This will further improve efficiency as the clock method always assumes the worst case propagation delay.

Even with these modifications, we are still far from realising our goals. One problem with our current approach is that, although parallelism is exploited when checking each individual hypothesis, the number of hypotheses checked is exponential in the number of abducibles. Two complementary strategies should be explored in order to address this problem. The first is to use some form of pruning during the search as in symbolic ALP systems such as [17]; and the second is to use some form of simplification when preprocessing the program as in Answer Set Programming (ASP) systems such as [19]. An important extension of the work involves exploiting the structure of the network to impose a preference on the order in which solutions are found. For example, the counter can be modified to output numbers in the order 0001, 0010, 0100, 1000, 0011, ... with the fewest number of bits high so that explanations will be discovered in order of minimality. In addition, the abducibles topologically far from the goal can be connected to the least significant bits of the counter, so that explanations will also be discovered in order of *basicality* [10].

A key direction for future work is that of integrating abductive reasoning with inductive learning in order to realise the benefits suggested by recent symbolic machine learning systems [15]. By providing a richer formalism for representing and reasoning about partial knowledge and integrity constraints, abduction could help to exercise a finer degree of control over which assumptions can and cannot be made in learning. In this context, it may be more appropriate to use a variation of the methodology presented in this paper, whereby the network's topology is projected onto a single layer recurrent network (computing the immediate consequence operator of the underlying program) and the threshold units are replaced by sigmoid neurons. This should enable an experimental validation of the approach as well as a more detailed comparison with symbolic systems.

Although many problems remain to be solved, we have presented some new techniques that may eventually lead to a fruitful synthesis with other approaches. Some interesting features of our methodology include combining the neural hardware description with the object logic program and using abduction to handle negation and cycles in the object program. It remains to be seen how these can be usefully integrated into a neural network learning framework.

REFERENCES

[1] A. Abdelbar, M. El-Hemaly, E. Andrews, and D. Wunsch II, 'Recurrent neural networks with backtrack-points and negative reinforcement applied to cost-based abduction', *Neural Networks*, **18**(5-6), 755–764, (2005).

[2] B. Ayeb, S. Wang, and J. Ge, 'A Unified Model For Neural Based Abduction', *IEEE Transactions on Systems, Man and Cybernetics*, **28**(4), 408–425, (1998).

[3] K. Eshghi and R.A. Kowalski, 'Abduction compared with negation by failure', in *Proceedings of the 6th International Conference on Logic Programming*, eds., G. Levi and M. Martelli, pp. 234–254. MIT Press, (1989).

[4] F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli, 'Multistrategy Theory Revision: Induction and Abduction in INTHELEX', *Machine Learning*, **38(1/2)**, 133–156, (2000).

[5] N. Fraser. BrainBox Neural Network Simulator (v. 1.8), 2006. At http://neil.fraser.name/software/brainbox/.

[6] A. d'Avila Garcez, K. Broda, and D. Gabbay, *Neural-Symbolic Learning Systems: Foundations and Applications*, Perspectives in Neural Computing, Springer, 2002.

[7] A. Goel and J. Ramanujam, 'A Neural Architecture for a Class of Abduction Problems', *IEEE Transactions on Systems, Man and Cybernetics*, **26**(6), 854–860, (1996).

[8] S. Höllbler and Y. Kalinke, 'Towards a massively parallel computational model for logic programming', in *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pp. 68–77, (1994).

[9] *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, eds., V. Honavar and L. Uhr, Boston Academic Press, 1994.

[10] A.C. Kakas, R.A. Kowalski, and F. Toni, 'Abductive Logic Programming', *Journal of Logic and Computation*, **2**(6), 719–770, (1992).

[11] A.C. Kakas and P. Mancarella, 'Generalized Stable Models: a Semantics for Abduction', in *Proceedings of the 9th European Conference on Artificial Intelligence*, pp. 385–391. Pitman, (1990).

[12] A.C. Kakas and F. Riguzzi, 'Abductive concept learning', *New Generation Computing*, **18**(3), 243–294, (2000).

[13] P. Lima, 'Logical Abduction and Prediction of Unit Clauses in Symmetric Hopfield Networks', in *Artificial Neural Networks, 2*, eds., I. Aleksander and J. Taylor, volume 1, pp. 721–725. Elsevier, (1992).

[14] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural approach to abductive multiadjoint reasoning, 2002.

[15] O. Ray, *Hybrid Abductive-Inductive Learning*, Ph.D. dissertation, Department of Computing, Imperial College London, UK, 2005.

[16] O. Ray. BrainBox Neural Network Abduction Demo Files, 2006. At http://www.doc.ic.ac.uk/~or/neural/abduction/demo.

[17] O. Ray and A. Kakas, 'ProLogICA: a practical system for Abductive Logic Programming', in *Proceedings of the 11th International Workshop on Non-monotonic Reasoning*, (2006). to appear.

[18] J. Reggia, Y. Peng, and S. Tuhrim, 'A Connectionist Approach to Diagnostic Problem-Solving Using Causal Networks', *Information Sciences*, **70**, 27–48, (1993).

[19] P. Simons, I. Niemelä, and T. Soininen, 'Extending and implementing the stable model semantics', *Artificial Intelligece*, **138**(1-2), 181–234, (2002).

[20] G. Towell and J. Shavlik, 'Knowledge-based artificial neural networks', *Artificial Intelligence*, **70**(1-2), 119–165, (1994).

[21] R. Vingrálek, 'A connectionist approach to finding stable models and other structures in nonmonotonic reasoning', in *Proceedings of the Second International Workshop on Logic Programming and Non-Monotonic Reasoning*, eds., L. Pereira and A. Nerode, pp. 60–81. MIT Press, (1993).

[22] C. Zhang and Y. Xu, 'A Neural Network Model for Diagnostic Problem Solving with Causal Chaining', *Neural Networks and Advanced Control Strategies*, **54**, 87–92, (1999).