Proceedings of IJCAI International Workshop on Neural-Symbolic Learning and Reasoning
NeSy 2005

A.S. d'Avila J. Elman P. Hitzler

Department of Computing

City University

Technical Report Series

TR/2005/DOC/01

ISSN 1364-4009

NeSy 2005

Neural-Symbolic Learning and Reasoning

Workshop at IJCAI-05

http://ijcai05.csd.abdn.ac.uk/
http://www.neural-symbolic.org/NeSy05/

Edinburgh, Scotland

1st August 2005

NeSy Programme

9.15 Opening

9.30 – 10.30 Keynote: Ron Sun

10.30 – 11.00 coffee break

11.00 – 11.15 (position paper) Pascal Hitzler, Sebastian Bader, Artur Garcez
Ontology Learning as a Use-Case for Neural-Symbolic Integration

11.20 – 11.45 Ernesto Burattini, Edoardo Datteri, Guglielmo Tamburrini
Neuro-symbolic programs for robots

11.50 – 12.15 Laurent Orseau
The Principle of Presence: A Heuristic for Growing Knowledge Structured Neural Networks

12.15 – 13.45 lunch break

13.45 – 14.10 Yuuya Sugita, Jun Tani
Learning Segmentation of Behavior to Situated Combinatorial Semantics

14.15 – 14.40 Sebastian Bader, Pascal Hitzler, Andras Witzel
Integrating First-Order Logic Programs and Connectionist Systems – A Constructive Approach

14.45 – 15.00 (position paper) Li Su, Howard Bowman, Brad Wyble
Symbolic Encoding of Neural Networks using Communicating Automata with Applications to Verification of Neural Network Based Controllers

15.00 – 15.30 coffee break

15:30 – 15.45 (position paper) Henrik Jacobsson, Tom Ziemke
Rethinking Rule Extraction from Recurrent Neural Networks

15.50 – 16.15 Jens Lehmann, Sebastian Bader, Pascal Hitzler
Extracting Reduced Logic Programs from Artificial Neural Networks

16.20 – 17.20 Keynote: Steffen Hölldobler
Logic Programs and Connectionist Systems

17.30 Closing

NeSy Table of Contents

NeSy Organising Committee

Dr. Artur Garcez is a Senior Lecturer at the Department of Computing at City University, London. He has over 50 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 2005. He is an area scientific editor (Logics and Neural Networks) of the Journal of Applied Logic, Elsevier, a member of the editorial board of the International Journal of Hybrid Intelligent Systems, IOS Press, and a member of the editorial board of the Cognitive Technologies book series, Springer-Verlag. He 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 a member of 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

Prof. Jeff Elman joined the UCSD Linguistics Department in 1977 after receiving his Ph.D. from University of Austin at Texas. In 1986, he helped found the Department of Cognitive Science - the first such department in the world - where he served as Chair from 1994 to 1998. He is currently Professor of Cognitive Science, Associate Dean the Division of Social Sciences at UCSD, Co-Director of the Kavli Institute for Brain and Mind, and Director of the Center for Research in Language. Elman is one of the pioneers in the field of artificial neural networks. His early model of speech perception, the TRACE model, remains one of the major theories in the field. In 1990 he developed the Simple Recurrent Network architecture (the so-called “Elman net”) which is today widely used in cognitive science to understand behaviors that unfold over time. His recent book, *Rethinking Innateness: A Connectionist Perspective on Development* (with Bates, Johnson, Karmiloff-Smith, Parisi, Plunkett, 1996), introduces a new theoretical framework for understanding the nature/nurture debate. Currently, Elman’s research focus is on language processing, development, and computational models of cognition. He was President of the Cognitive Science Society from 1999 to 2000 and in 2001 was selected as one of five Inaugural Fellows of the Society. Also in 2001, he was awarded an honorary degree from the New Bulgarian University.

Dr. Pascal Hitzler is project leader and research assistant at the Institute for Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe in Germany, where he is involved in national and international projects on semantic web technologies, including KnowledgeWeb, SEKT, and SmartWeb. He received a PhD in Mathematics from UCC Cork, Ireland, in 2001, and a Diplom in Mathematics and Computer Science from the University of Tübingen, Germany, in 1998. His research record lists over 80 publications in such diverse areas as neural-symbolic integration, semantic web, knowledge representation and reasoning, lattice and domain theory, denotational semantics, and set-theoretic topology. 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.

NeSy Programme Committee

Artur d'Avila Garcez (City University London, UK)
Howard Blair (Syracuse University, USA)
Jeff Elman (University of California San Diego, USA)
Dov Gabbay (Kings College London, UK)
Marco Gori (University of Siena, Italy)
Barbara Hammer (TU Clausthal, Germany)
Pascal Hitzler (University of Karlsruhe, Germany)
Steffen Hölldobler (TU Dresden, Germany)
Luis Lamb (Federal University of Rio Grande do Sul, Brazil)
John Lloyd (The Australian National University, Australia)
Vasile Palade (Oxford University, UK)
Asim Roy (Arizona State University, USA)
Antony K. Seda (University College Cork, Ireland)
Jude Shavlik (University of Wisconsin, USA)
Rudi Setiono (National University, Singapore)
Alessandro Sperduti (University of Padova, Italy)
Stefan Wermter (University of Sunderland, UK)
Gerson Zaverucha (Federal University of Rio de Janeiro, Brazil)

NeSy 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, a journal's dedicated scientific area 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. Ron Sun and Prof. Steffen Hölldobler, the authors of the papers submitted to the workshop, and the IJCAI-05 workshop chair, Dr. Carlos Guestrin, for his assistance in the organisation of the workshop.

Edinburgh, August 2005 Artur d'Avila Garcez , Jeff Elman, Pascal Hitzler

Keynote speaker: Prof. Ron Sun

Abstract: The general idea behind hybrid systems, developing more comprehensive models through integrating a variety of techniques, can be further extended to so called cognitive architectures, that is, cognitive models that encompass a wide range of cognitive capabilities. Building cognitive architectures is a difficult task for artificial intelligence and cognitive science. Issues and challenges in developing cognitive architectures will be outlined, examples of cognitive architectures will be given, and possible future directions will be sketched.

Short Bio: Dr. Ron Sun is Professor of Cognitive Science at Rensselaer Polytechnic Institute, and formerly the James C. Dowell Professor of Engineering and Computer Science at University of Missouri-Columbia. He received his Ph.D in 1992 from Brandeis University. His research interest centers around studies of cognition, especially in the areas of cognitive architectures, human and machine reasoning and learning, multi-agent interaction and social simulation, and hybrid connectionist models. For his paper on integrating rule-based reasoning and connectionist models, he received the 1991 David Marr Award from Cognitive Science Society. He has written or edited eight books by various publishers. He is the founding co-editor-in-chief of the journal Cognitive Systems Research (published by Elsevier). He also serves on the editorial board of Connection Science, Applied Intelligence, and other journals. He is the general chair and program chair for COGSCI 2006. He is a member of the Governing Board of International Neural Networks Society.

Keynote speaker: Prof. Steffen Hölldobler

Abstract: How to represent and reason about structured objects and structure-sensitive processes in a fully connectionist setting is a long-standing open research problem. Recently, we were able to show that various semantic meaning operators associated with first-order logic programs can be approximated arbitrarily well by a feed-forward connnectionist network. By adding recurrent connections these networks can even approximate the least fixed point of a semantic meaning operator. However, the problem of how to practically construct such networks for given logic programs remains. In the talk I will present this approach, discuss its limitations, and present various approaches for practically constructing connectionist networks for first-order logic programs.

Short Bio: Steffen Hölldobler has received the title of a Dr. rer. nat. in 1988 from the University of the Armed Forces Munich. During a post-doctoral fellowship at the International Computer Science Institute in Berkeley, California, from 1989 to 1990 he was introduced to connectionism by Jerome Feldman and developed the core of his post-doctoral thesis on “Automated Inferencing and Connectionist Models”. In 1993 he became Professor for “Knowledge Representation and Reasoning” at the Department of Computer Science of the Technische Universität Dresden. He is currently director of the International Center for Computational Logic and head of the European Masters Program in Computational Logic. Steffen Hölldobler has developed various connectionist models for propositional and first-order reasoning including a connectionist unification algorithm, a connectionist inference system for first-order Horn Logic (CHCL), a recursive propositional connectionist inference system for normal logic programs, and a recursive neural network for reflexive reasoning. He has also shown that the immediate consequence operator of certain classes of first-order logic programs can be approximated arbitrarily well by feedforward neural networks and, consequently, that the least fixed point of such programs can be approximated arbitrarily well by recurrent networks with a feedforward kernel.

Ontology Learning as a Use-Case for Neural-Symbolic Integration (position paper)

Pascal Hitzler[1*], **Sebastian Bader**[2†], **Artur Garcez**[3]
[1]AIFB, University of Karlsruhe, Germany
[2]International Center for Computational Logic, TU Dresden, Germany
[3]Department of Computing, City University London, UK

Abstract

We argue that the field of neural-symbolic integration is in need of identifying application scenarios for guiding further research. We furthermore argue that ontology learning — as occuring in the context of semantic technologies — provides such an application scenario with potential for success and high impact on neural-symbolic integration.

1 Neural-Symbolic Integration

Intelligent systems based on symbolic knowledge processing, on the one hand, and on artificial neural networks (also called connectionist systems), on the other, differ substantially. Nevertheless, these are both standard approaches to artificial intelligence and it would be very desirable to combine the robustness of neural networks with the expressivity of symbolic knowledge representation. This is the reason why the importance of the efforts to bridge the gap between the connectionist and symbolic paradigms of Artificial Intelligence has been widely recognised. As the amount of hybrid data containing symbolic and statistical elements as well as noise increases in diverse areas such as bioinformatics or text and web mining, neural-symbolic learning and reasoning becomes of particular practical importance. Notwithstanding, this is not an easy task, as illustrated in the sequel.

The merging of theory (background knowledge) and data learning (learning from examples) in neural networks has been indicated to provide learning systems that are more effective than purely symbolic and purely connectionist systems, especially when data are noisy [16]. This has contributed decisively to the growing interest in developing neural-symbolic 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, which are usually characterized by simple feature vectors. While significant theoretical progress has recently been made on knowledge representation and reasoning using neural networks, and on direct processing of symbolic and structured data using neural methods, the integration of neural computation and expressive logics such as first order logic is still in its early stages of methodological development.

Concerning knowledge extraction, we know that neural networks have been applied to a variety of real-world problems (e.g. in bioinformatics, engineering, robotics), and they were particularly successful when data are noisy. But entirely satisfactory methods for extracting symbolic knowledge from such trained networks in terms of accuracy, efficiency, rule comprehensibility, and soundness are still to be found. And problems on the stability and learnability of recursive models currently impose further restrictions on connectionist systems.

In order to advance the state of the art, we believe that it is necessary to look at the biological inspiration for neural-symbolic integration, to use more formal approaches for translating between the connectionist and symbolic paradigms, and to pay more attention to potential application scenarios. We will argue in the following that ontology learning provides such an application scenario with potential for success and high impact.

2 The Need for Use Cases

The general motivation for research in the field of neural-symbolic integration (just given) arises from conceptual observations on the complementary nature of symbolic and neural network based artificial intelligence described above. This conceptual perspective is sufficient for justifying the mainly foundations-driven lines of research being undertaken in this area so far. However, it appears that this conceptual approach to the study of neural-symbolic integration has now reached an impasse which requires the identification of use cases and application scenarios in order to drive future research.

[*] Pascal Hitzler is supported by the German Federal Ministry of Education and Research under the SmartWeb project, and by the European Commission under contract IST-2003-506826 SEKT.

[†] Sebastian Bader is supported by the GK334 of the German Research Foundation (DFG).

Indeed, the theory of integrated neural-symbolic systems has reached a quite mature state but has not been tested extensively so far on real application data. From the pioneering work by McCulloch and Pitts [27], a number of systems has been developed in the 80s and 90s, including Towell and Shavlik's KBANN [33], Shastri's SHRUTI [31], the work by Pinkas [29], Hölldobler [21], and d'Avila Garcez et al. [15; 17], to mention a few. The reader is referred to [9; 16; 19] for comprehensive literature overviews. These systems, however, have been developed for the study of general principles, and are in general not suitable for real data or application scenarios that go beyond propositional logic. Nevertheless, these studies provide methods which can be exploited for the development of tools for use cases, and significant progress can now only be expected as a continuation of the fundamental research undertaken in the past.

The systems just mentioned — and most of the research on neural-symbolic integration to date — is based on propositional logic or similarly finitistic paradigms. Significantly large and expressible fragments of first order logic are rarely being used because the integration task becomes much harder due to the fact that the underlying language is infinite but shall be encoded using networks with a finite number of nodes [6]. The few approaches known to us to overcome this problem are (a) the work on recursive autoassociative memory, RAAM, initiated by Pollack [30], which concerns the learning of recursive terms over a first-order language, and (b) research based on a proposal by Hölldobler et al. [23], spelled out first for the propositional case in [22], and reported also in [20]. It is based on the idea that logic programs can be represented — at least up to subsumption equivalence [26] — by their associated single-step or immediate consequence operators. Such an operator can then be mapped to a function on the real numbers that can, under certain conditions, in turn be encoded or approximated e.g. by feedforward networks with sigmoidal activation functions using an approximation theorem due to Funahashi [13], and (c) more recently, the idea of fibring neural networks [12], which follows from Gabbay's fibring methodology to combine logical systems [14], has been used to represent acyclic first order logic programs with infinitely many ground instances in a (simple, by comparison) neural network [5].

In addition to these and a number of other sophisticated theoretical results — reported e.g. in [4; 5; 6; 7; 20; 23] —, first-order neural-symbolic integration still remains a widely open issue, where advances are very difficult, and it is very hard to judge to date to what extent the theoretical approaches can work in practice. We argue that the development of use cases with varying levels of expressive complexity is, as a result, needed to drive the development of methods for neural-symbolic integration beyond propositional logic.

3 Semantic Technologies and Ontology Learning

With amazing speed, the world wide web has become a widespread means of communication and information sharing. Today, it is an integral part of our society, and will continue to grow. However, most of the information available cannot be processed easily by machines, but has to be read and interpreted by humans. In order to overcome this limitation, a world-wide research effort is currently being undertaken, following the vision put forward by Berners-Lee et al. [8] to make the contents of the world wide web accessible, interpretable, and usable by machines. The resulting extension of the World Wide Web is commonly referred to as the *Semantic Web*, and the underlying technological infrastructure which is currently being developed is referred to as *Semantic Technologies*.

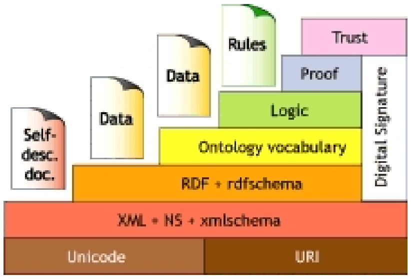


Figure 1: The Semantic Web Layer Cake

In this process, a key idea is that web content should be provided with conceptual background — often referred to as *ontologies* [32] — which allows machines to put information into context, making it interpretable. These research efforts are grouped around the so-called semantic web layer cake, shown in Figure 1; it depicts subsequent layers of functionality and expressiveness, which shall be put in place incrementally. Most recently — having established RDF and RDF-Schema as basic syntax — the OWL Web Ontology Language [2; 28], which is a decidable fragment of first-order logic, has been recommended by the world wide web consortium (W3C) for the ontology vocabulary.

Conceptual knowledge is provided by means of statements in some logical framework, and the discussion concerning suitable logics is still ongoing. Description Logics [3] will most likely play a major role, as they provide the foundation for OWL, but other approaches are also being considered. Currently, the development of an expressive rule-based logic layer on top of OWL for the inference of ontological knowledge is being investigated. But also fragments of OWL, including Horn and propositional languages, are being used, as different application scenarios necessitate different tradeoffs between expressiveness, conceptual and computational complexity, and scalability.

The construction of ontologies in whatever language, however, appears as a narrow bottleneck to the proliferation of the Semantic Web and other applications of Semantic Technologies. The success of the Semantic Web and its technologies indeed depends on the rapid and inexpensive development, coordination, and evolution of ontologies. Currently, however, these steps all require cumbersome engineering processes, associated with high costs and heavy time strain on domain experts. It is therefore desirable to automate the ontology creation and ontology refinement process, or at least to provide intelligent ontology learning systems that aid the ontology engineer in this task.

From a bird's eye's view, such a system should be able to handle terms and synonyms, in order to build abstract concepts and concept hierarchies from text-based websites. This basic ontological knowledge then needs to be further refined using relations and rules, in accordance with established or to-be-established standards for ontology representation. Current systems [10; 11; 25] use only very basic ontology languages, but technological advances are expected soon, since the need for expressive ontology languages is generally agreed upon.

4 Ontology Learning as Use Case

We argue that ontology learning, as just described, constitutes a highly interesting application area for neural-symbolic integration. As a use case, it appears to be conceptually sound, technically feasible, and of potential high impact. Let us now give our arguments in more detail.

4.1 Conceptually Sound

Machine learning methods based on artificial neural networks are known to perform well in the presence of noisy data. If ontologies are to be learned from such uncontrolled data like real existing webpages or other large data repositories, the handling of noise becomes a real issue. At the same time, we can only expect to be able to make reasonable generalizations from data given in the form of html pages if background knowledge is also taken into account. It would be natural for such background knowledge to be ontology-based and therefore symbolic. Furthermore, the required output necessarily has to be in a logic-based format because it will have to be processed by standard tools from the semantic web context. This would require the use of efficient knowledge extraction algorithms to derive compact symbolic representations from large-scale trained neural networks.

It looks as though ontology learning requires the integration of symbolic and neural networks-based approaches, which is provided by the methods developed in the field of neural-symbolic integration. Current results and systems indicate that machine learning of ontologies is a very difficult task, and that the most suitable methods and approaches still remain to be identified. We believe that in the end mixed strategies will have to be used to arrive at practical tools, and due to the above mentioned reasons neural-symbolic learning systems can be expected to play a significant role.

4.2 Technically Feasible

The specific nature of ontology research led to the development of a variety of different ontology representation languages, and various further modifications of these. Some of them are depicted in Figure 2. Standardization efforts are successfully being undertaken, but it is to be expected that a number of ontology languages of different logical expressivity will remain in practical use. This diversity is natural due to the different particular needs of application scenarios.

As we have identified earlier, the different levels of expressivity correspond well to the specific requirements on a use case scenario to drive neural-symbolic integration research. Propositional methods can be applied to the learning of concept hierarchies or DLP ontologies. Decidable fragments such as the different versions of OWL provide more sophisticated challenges without having to tackle the full range of difficulties inherent of first order neural-symbolic integration. As for learning, we also expect that the learning of conceptual knowledge should harmonize naturally with learning paradigms based on Kohonen maps or similar architectures.

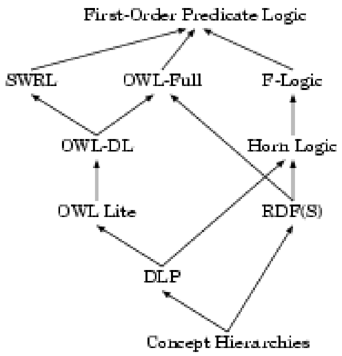


Figure 2: Some ontology languages. Arrows indicate inclusions between the languages. Concept hierarches are simple 'is-a' hierarchies corresponding to certain fragments of propositional logic. The standard OWL [2; 28] already comes in different versions. DLP [18; 34] refers to a weak but practically interesting datalog fragment of OWL. F-Logic [24; 1] provides an alternative ontology paradigm.

4.3 High Potential Impact

The learning of ontologies from raw data has been identified as an important topic for the development of Semantic Technologies. These, in turn, are currently migrating into various research and application areas in artificial intelligence and elsewhere, including knowledge management, ambient computing, cognitive systems, bioinformatics, etc. At the same time, ontology learning appears to be a very hard task, and suitable new learning methods are currently being sought. Neural-symbolic integration has the potential for significant contricution to this area and thus to one of the currently prominent streams in computer science.

5 Conclusions

We have identified ontology learning as a potential use case for neural-symbolic integration. We believe that this would further neural-symbolic integration as a field, and provide significant contributions to the development of Semantic Technologies.

Acknowledgement We are greatful for a number of very interesting and stimulating comments by the anonymous referees, containing substantial further ideas and related thoughts, which we could not incorporate in full in the final version.

References

[1] Jürgen Angele and Georg Lausen. Ontologies in F-logic. In Staab and Studer [32], pages 29–50.

[2] Grigoris Antoniou and Frank van Harmelen. Web Ontology Language: OWL. In Staab and Studer [32], pages 67–92.

[3] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[4] Sebastian Bader and Pascal Hitzler. Logic programs, iterated function systems, and recurrent radial basis function networks. *Journal of Applied Logic*, 2(3):273–300, 2004.

[5] Sebastian Bader, Pascal Hitzler, and Artur S. d'Avila Garcez. Computing first-order logic programs by fibring artificial neural network. In *Proceedings of the 18th International FLAIRS Conference, Clearwater Beach, Florida, May 2005*, 2005. To appear.

[6] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler. The integration of connectionism and knowledge representation and reasoning as a challenge for artificial intelligence. In L. Li and K.K. Yen, editors, *Proceedings of the Third International Conference on Information, Tokyo, Japan*, pages 22–33. International Information Institute, 2004. ISBN 4-901329-02-2.

[7] Sebastian Bader, Pascal Hitzler, and Andreas Witzel. Integrating first-order logic programs and connectionist systems — a constructive approach. In *Proceedings of the IJCAI-05 Workshop on Neural-Symbolic Learning and Reasoning, NeSy'05, Edinburgh, UK*, 2005. To appear.

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[9] Anthony Browne and Ron Sun. Connectionist inference models. *Neural Networks*, 14(10):1331–1355, 2001.

[10] Philipp Cimiano, Andreas Hotho, and Steffen Staab. Comparing conceptual, partitional and agglomerative clustering for learning taxonomies from text. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'04)*, pages 435–439. IOS Press, 2004.

[11] Philipp Cimiano, Andreas Hotho, and Steffen Staab. Learning concept hierarchies from text using formal concept analysis. *Journal of Artifical Intelligence Research*, 200x. To appear.

[12] Artur S. d'Avila Garcez and Dov M. Gabbay. Fibring neural networks. In *Proceedings of 19th National Conference on Artificial Intelligence AAAI'04*, pages 342–347, San Jose, California, USA, July 2004. AAAI Press.

[13] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[14] Dov M. Gabbay. *Fibring Logics*. Oxford Univesity Press, 1999.

[15] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.

[16] Artur S. d'Avila Garcez, Krysia B. Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems — Foundations and Applications*. Perspectives in Neural Computing. Springer, Berlin, 2002.

[17] Artur S. d'Avila Garcez and Gerson Zaverucha. The connectionist inductive lerarning and logic programming system. *Applied Intelligence, Special Issue on Neural networks and Structured Knowledge*, 11(1):59–77, 1999.

[18] Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logics. In *Proc. of WWW 2003, Budapest, Hungary, May 2003*, pages 48–57. ACM, 2003.

[19] Hans W. Güsgen and Steffen Hölldobler. Connectionist inference systems. In Bertram Fronhöfer and Graham Wrightson, editors, *Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Artificial Intelligence*, pages 82–120. Springer, Berlin, 1992.

[20] Pascal Hitzler, Steffen Hölldobler, and Anthony K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 3(2):245–272, 2004.

[21] Steffen Hölldobler. *Automated Inferencing and Connectionist Models*. Fakultät Informatik, Technische Hochschule Darmstadt, 1993. Habilitationsschrift.

[22] Steffen Hölldobler and Yvonne Kalinke. Towards a massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.

[23] Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.

[24] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

[25] Alexander Maedche and Steffen Staab. Ontology learning. In Staab and Studer [32].

[26] Michael J. Maher. Equivalences of logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 627–658. Morgan Kaufmann, Los Altos, CA, 1988.

[27] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[28] Web ontology language (OWL). www.w3.org/2004/OWL/, 2004.

[29] Gadi Pinkas. Propositional non-monotonic reasoning and inconsistency in symmetric neural networks. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 525–530. Morgan Kaufmann, 1991.

[30] Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105, 1990.

[31] Lokenda Shastri. Advances in Shruti — A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence*, 11:78–108, 1999.

[32] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

[33] Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1–2):119–165, 1994.

[34] Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, University of Karlsruhe, 2004.

Neuro-symbolic programs for robots

Ernesto Burattini*, Edoardo Datteri°, Guglielmo Tamburrini*
*Dipartimento di Scienze Fisiche, Università di Napoli "Federico II"
Complesso Universitario Monte S. Angelo, Via Cintia, 80126 Napoli, Italy
°Dipartimento di Filosofia, Università di Pisa
Piazza Torricelli 3a, 56126 Pisa, Italy
{ernb,datteri,tamburrini}@na.infn.it

Abstract

This paper introduces a neuro-symbolic behaviour modelling language (NSBL), which enables one to specify both reactive and deliberative robotic behaviours, and to model both competitive and cooperative control functions. *Core* NSBL is grounded into the neural processing of propositional clauses, crucially involving the non-monotonic processing of *unless* conditions. Core NSBL suffices to model a variety of reactive and deliberative processes, behavioural sequencing, and competitive control functions. *Extended* NSBL is achieved by fibring neural nets. Fibred neural networks enable one to deal with the cooperative control of behaviours by the computation of polynomial functions. A two-stage translation algorithm for core NSBL (rules→neural model→FPGA implementation) provides a powerful tool to meet bounded time response constraints for deliberative robotic systems operating in dynamic environments. The paper also speculates about how to achieve adaptivity and neural learning in NSBL-based agents.

1 Motivations and Background

Developing AI and robotic systems which take a clear advantage from the strengths of both symbolic and neural processing is a major challenge for neuro-symbolic approaches [d'Avila Garcez *et al.*, 2002]. In particular, cognitive robotics has been identified as a suitable test bed for neuro-symbolic integration, insofar as a cognitive robot must be capable of combining knowledge-based reasoning and learning with prompt behavioural responses in dynamic environments [Bader *et al.*, 2005]. Some steps towards neuro-symbolic cognitive robotics are undertaken in this paper, which introduces a neuro-symbolic behaviour modelling language (NSBL) enabling one to specify and control the behaviour of robotic agents. *Core* NSBL (cNSBL, from now on) is grounded into the neural processing of propositional clauses, crucially involving the non-monotonic processing of *unless* conditions [Aiello *et al.*, 1998]. *Extended* NSBL (eNSBL) is achieved by fibring neural nets [d'Avila Garcez and Gabbay, 2004]. Many behaviour-based systems, from the extreme case of pure "stretch-reflex" reactive ones to deliberative or hybrid systems [Arkin, 1998], can be uniformly modelled as eNSBL agents. In addition to that, eNSBL presents distinctive advantages in connection with the relation between logical and neural models: the close correspondence between the neural and the eNSBL level of description allows one to reconstruct the logical model underlying a neuro-symbolic network implemented in a behaviour-based system, revealing the embedded decisional structure. In view of the massive parallelism of rule-based reasoning made possible by neuro-symbolic approaches, NSBL-based agents can be endowed with perceptual and deliberation capabilities, while meeting at the same time bounded time response constraints for operation in dynamic environments.

cNSBL suffices to model a wide variety of reactive behaviours and deliberative processes, behavioural sequencing, and competitive control functions. A two-stage translation algorithm for cNSBL provides a powerful tool to meet bounded time response requirements for deliberation processes carried out by robotic systems. The first stage is the translation of cNSBL programs into neural networks formed by binary threshold neurons. In the second stage, the resulting neural nets are compiled into VHDL (Very High Speed Integrated Circuits) code for the FPGA (Field Programmable Gate Array) hardware. This two-stage translation enables one to move on from the theoretical possibility of parallel computation of rules demonstrated by the neural modelling of cNSBL operators to the actual FPGA parallel execution of cNSBL programs [Burattini *et al.*, 2000]. cNSBL is introduced in section 2, while section 3 explains how a variety of behaviour based structures are modelled in cNSBL. Extended NSBL (eNSBL), introduced in section 4, is achieved by adding fibred neural nets to cNSBL. This extension enables one to model within eNSBL cooperative control of behaviours which involves the computation of polynomial functions – notably potential field navigation [Latombe, 1991].

The neuro-symbolic treatment of sets of propositional rules for both monotonic and non-monotonic reasoning in AI systems is significantly connected to the present work on cNSBL. To begin with, the development of a neuro-symbolic shell for diagnostic expert systems [Burattini and Tamburrini, 1992] paves the way to meeting the bounded time responses required in selective domains of diagnostic problem solving (such as fault diagnosis in nuclear power plants) by means of massively parallel neural computation. A similar motivation suggests the opportunity of adopting a neuro-symbolic approach in cognitive robotics too, for a robotic system endowed with the inferential capabilities of these expert system shells would be able to perform efficiently forms of hypothesize-and-test reasoning that are involved in a variety of perceptual and decision-making tasks, while meeting at the same time the strict temporal constraints imposed by action in real environments. Moreover, translation algorithms for systems of propositional rules are available (see, for example, [Aiello *et al.*, 1995; Aiello *et al.*, 1998]), that enable one to generate neural machinery for the non-monotonic processing of *unless* and *until* operators, in addition to outputting networks of binary threshold units for computing logic programs formed by general propositional clauses, as in [Hoelldobler and Kalinke, 1994]. The neural processing of non-monotonic reasoning is achieved there by proper use of inhibitory neural signals. Finally, the above mentioned two-stage translation algorithm [Burattini *et al.*, 2000] enables one to move from these systems of rules for non-monotonic reasoning to their neural representations, and from these to their FPGA implementations.

The concluding remarks presented in sect. 5 concern the development of basic adaptivity of NSBL agents and proper ways of achieving more extensive adaptivity by neural reinforcement learning.

2 Core NSBL

Behaviour-based robotics is a relatively recent area of robotics research, which focuses on the development of adaptive robots that operate in real (not simulated) environments under restrictive temporal constraints. A distinctive modelling strategy is pursued in behaviour-based robotics, which analyses adaptive capabilities as resulting from the coordinated activities of *behaviours* representing system subcapacities. Adaptive navigation, for instance, is often modelled as coordinated activities of ‘going towards the target’ and ‘avoiding obstacles’. In behaviour-based systems, behaviours are typically parallel and asynchronous, and coordination mechanisms arbitrate between simultaneously active behaviours in order to drive system effectors [Arkin, 1998].

In this section we introduce cNSBL, a neuro-symbolic language for modelling behaviour-based systems. cNSBL is based on the IMPLY and UNLESS operators of the language NSL [Burattini *et al.*, 2000]. With respect to NSL, cNSBL introduces additional constraints on the semantic interpretation of propositional variables (that will be presented in the next section). IMPLY and UNLESS operators take cNSBL sentences as arguments; a cNSBL sentence is a propositional literal, a conjunction or else a disjunction of propositional literals. One associates to each literal l_i a distinct neural element l^i. We stipulate that the truth-value of each propositional literal can be *True*, *False*, or *Undefined*. These values are readily associated to the firing state of weighted–sum non-linear thresholded neurons (NSN), so that if a literal l_i and its negation $\sim l_i$ are represented as distinct NSN neurons l^i and $\sim l^i$, respectively, one can encode the value *True* of l_i by the firing of l^i**,** the value *False* by the firing of $\sim l^i$, and the value *Undefined* by the quiescence of both neural elements. For an early approach following these general guidelines, see [von Neumann 1956].

The main reason for adopting this kind of representation in the present context is that robotic systems may be compelled to act even when no information is available about the truth-value of some given sentence. In particular, the action to undertake in the absence of such information may differ from the actions that the system would undertake if the sentence were either (known to be) true or (known to be) false. Some pertinent examples are discussed in section 3. Similar representational issues arise in connection with neuro-symbolic approaches to diagnostic problem solving [Burattini and Tamburrini, 1992].

Let $P=\{p_i\}$ $(0 < i < n)$ and $Q=\{q_j\}$ $(0 < j < m)$ be sets of propositional literals (for some n, m $\in$N). Let $P_\wedge$ the conjunction of the elements of P, and let $Q_\vee$ be the disjunction of the elements of Q; let s be a literal. IMPLY($P_\wedge$, s) is intuitively interpreted as “IF the conjunction of literals $P_\wedge$ is true THEN s is true” and UNLESS($P_\wedge$, $Q_\vee$, s) is intuitively interpreted as “IF the conjunction of literals $P_\wedge$ is true and the disjunction of literals $Q_\vee$ is false or undefined THEN s is true”. Appropriate three-valued truth-tables for computing the truth-value of literals, conjunctions, and disjunctions of literals are those given in [Kleene, 1952] p. 334. Two additional operators are introduced in NSL: ATLEAST($P_\vee$, k, s) interpreted as “IF $j \geq k$ literals of the disjunction $P_\vee$ are true THEN s is true”, and ATMOST ($P_\vee$, k, s) interpreted as “IF $j \leq k$ literals of the disjunction $P_\vee$ are true THEN s is true”. Well-formed IMPLY and UNLESS statements are called *cNSBL rules*. Let x, y be NSNs; let $U(x) \in \{0,1\}$ be the output of x; and let $W_{x,y} \in \mathcal{R}$ be the weight of the connection from x to y. Each cNSBL literal l_i will be encoded as a distinct NSN neuron l^i. Accordingly, the translation algorithm into NSN for the UNLESS($P_\wedge$, $Q_\vee$, s) rule outputs the NSN netw

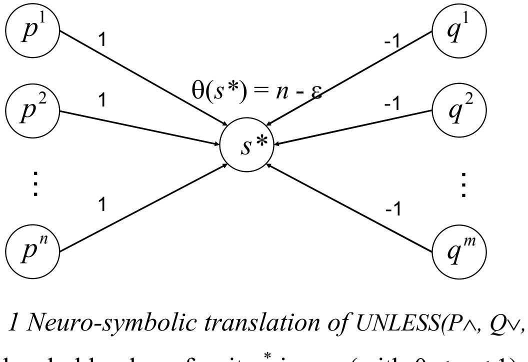


Figure 1 Neuro-symbolic translation of UNLESS(P∧, Q∨, s).

The threshold value of unit s^* is n-ε (with $0 < \varepsilon < 1$), the weights of the connections from each of the n units p^i representing the truth of the elements of $P_\wedge$ and s_t are set to 1, and the weights from all the q^i and s^* are set to -1. This neural network captures the intended semantics of UNLESS(P∧, Q∨, s) insofar as s^* fires (at time t+1) if and only if *all* the units representing the elements of $P_\wedge$ fire at time t, and no unit representing an element of $Q_\vee$ fires at time *t*. Let us note that activation of at least one unit representing an element of $Q_\vee$ inhibits the s^* unit, even when all $P_\wedge$ neurons are simultaneously firing.

It is worth noting that the cNSBL proposition ($p_i \vee \sim p_i$) is not a tautology; in particular, p_i is undefined when both neurons encoding the truth and the falsehood of p_i are inactive. As illustrated in section 3 below, this possibility turns out to be relevant for developing non-monotonic inference modules embedded in behaviour-based systems. In the underlying NSN net, both neurons encoding p_i and $\sim p_i$ are inactive if they have not been activated by some other neurons of the net or by an external source. On the basis of an epistemic interpretation, we refer to this state of affairs as the *lack of support* for the truth-value of p_i or the truth-value of $\sim p_i$. The lack of support for the truth-value of a variable is clearly distinct from its falsehood, and paves the way for non-monotonic cNSBL processing of rules.

3 cNSBL behaviour-based agents

Let us turn now to describe how behaviour-based agents are modelled in cNSBL. These agents are represented as a combination of two layers, one modelled as a set of cNSBL rules that we call for short "cNSBL layer",, the other one including *sensory transduction* and *motor actuation* mechanisms. At each time *t* (assuming discrete time), the state of the cNSBL layer is given by the truth-values of *n* propositional variables $Q=\{q_1,\ldots q_n\}$. Distinct subsets *S* and *A* of *Q* play the following distinctive roles (see Figure 2):

- the values of the variables belonging to *S* (the set of *sensory variables*) are determined by a sensory transduction mechanism;
- the values of the variables that belong to *A* (the set of *motor variables*) determine the behaviour of the motor actuation mechanism(s), e.g., the motors of the system run forward if and only if the variable $a_1 \in A$ is true. A one-one correspondence between these motor variables and the actions in the motor actuation layer is required.

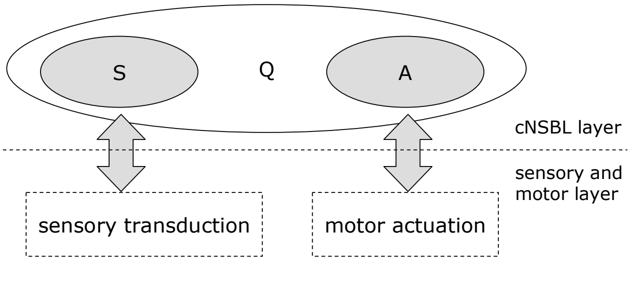


Figure 2 The structure of cNSBL behaviour-based agents.

Actions can be durative or discrete, as in [Nilsson, 1994]. If M is a *durative* action, its execution starts when $M_{on} \in A$ becomes true, and continues on until M_{on} is no longer true (see Figure 3). Each *discrete* action is connected to two motor variables of the cNSBL layer. M_{on} triggers action M, and M_{end} becomes true when M has been completed (see Figure 4). It is required that the truth-value of the latter variable is determined by the motor actuation layer alone, and cannot be controlled by other cNSBL variables.

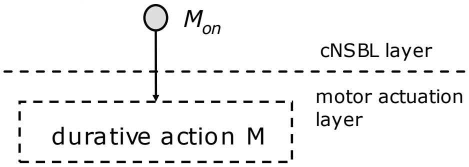


Figure 3 Durative actions.

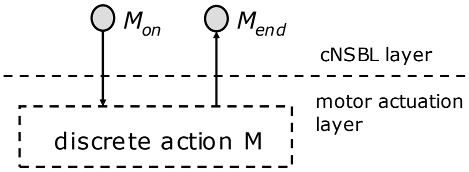


Figure 4 Discrete actions.

No additional constraints on sensory transduction and motor actuation mechanisms are imposed at this level of generality. Just to make one concrete example, these mechanisms can be implemented on the basis of the GNU-licensed ARIA environment for programming mobile robots [ARIA] (the sensory layer is obtained with the *getSonarReading* function, and the motor actuation is based on the *move* function, issuing a discrete action).

A finite set of cNSBL propositions (a cNSBL *program*) specifies how the values of some cNSBL variables in Q at time *t+1* depends on the value of the variables in Q at time *t*. Two sub-sets of Q are not controlled by cNSBL rules: the sensory variables, which are fixed by the sensory transduction mechanism, and a subset of motor variables that signal the end of a discrete action (like the M_{end} in Figure 4). A simple version of Braitenberg vehicle #1 [Braitenberg, 1984], that goes forward if there is light, can be modelled using the IMPLY operator only, with the rule IMPLY(*light*, *go*), where the sensory variable *light* is true iff the sensor detects light, and the system motor goes forward iff the motor variable *go*, triggering a durative action, is true. The operator UNLESS can be used to stop motion in case the robot bumper senses an obstacle; this is achieved by the rule UNLESS(*light*, *b*, *go*), where *b* (a sensory variable) is true iff the bumper hits an obstacle. The value of some state variables at time *t*+1 (in this example *go*) depends on the value of the state variables at the previous time instant, in a way that is defined by the cNSBL program.

In more general terms, each behaviour *b* of the system is construed as a function from a set $I_b \in Q$ of input variables to a set $O_b \in Q$ of output variables. Clearly, cNSBL behaviours may compute trivial I_b/O_b transformations like "if there is light, then turn on the motor". It is worth noting, however, that a cNSBL behaviour can embody expert system capabilities, thus making action dependent on more complex inference processes, which take internal and sensory variables as inputs. A robotic system endowed with these inferential capabilities would be able to carry out a wide variety of knowledge-based perceptual processes (such as classification of objects presented in various perceptual modalities) and decision-making processes (such as inferences to the best explanation). This possibility is demonstrated in [Burattini and Tamburrini, 1992; Aiello *et al.*, 1998], where similar neuro-symbolic machinery is used to implement shells for diagnostic expert systems carrying out hypothesize-and-test inference processes.

Attempts to model perception-action coordination behaviours with cNSBL require that inputs and outputs of behaviours should be coded as finite sets of propositional variables. This constraint does not prevent one to interface the cNSBL layer with continuous-state sensory and motor mechanisms, via quantization mechanisms; sensory cNSBL variables, in that case, may be used to encode discrete conditions directly depending on sensory data, which include both local conditions like "there is a door in front of the robot" as well as global conditions like "the door of the kitchen is open". And cNSBL behaviours may be used to model the decision modules of the system, that infer the right action to do based on the current situation. A discrete-state approach for modelling the decision (inferential) modules of behaviour-based robots is typically adopted in competitive, winner-takes-all arbitration mechanisms for action selection in robotic systems [Tyrrell, 1993]. It is also supported by investigations in cognitive ethology that are relevant for biologically inspired robotic approaches: prominent models of animal behaviour are distinctively based on discrete (mutually incompatible) behavioural repertoires, and discrete sets of internal conditions causally related to behaviour [McFarland and Sibly, 1975]. More significantly, the quantization of variables involved in action selection is adopted in behaviour-based robotics as well. In [Nilsson, 1994], production rules are used to connect discrete conditions, evaluated against sensory data, with distinct actions, to achieve perception-action coordination in robots. And Arkin [Arkin, 1998] points to discrete encoding of behaviours by production rules as well (section 3.3.1). We enrich this picture here, by showing how many control strategies typically used in behaviour-based robotics are modelled in cNSBL. As remarked above, competitive coordination mechanisms are easily obtained by means of UNLESS and IMPLY rules. An example is inhibition of behaviours, a key element of subsumption architectures [Brooks, 1986]. Behaviour B inhibits behaviour A if, intuitively, B prevents the output of A from being sent to the agent's effectors. Let A be, for example, a light-detector behaviour that sets variable a to true iff the corresponding sensor perceives light; and let B be an obstacle-detector mechanism, which sets b to true when there is an obstacle in front of the robot; the variable f_{on} triggers the durative action *forward*. Inhibition of the link between A and the *forward* action by the behaviour B is straightforwardly obtained by the rule UNLESS(a, b, f_{on}). See Figure 5.

By combining UNLESS and IMPLY rules, behaviourally richer systems can be obtained. For example, by adding the rule IMPLY(b, t_{on}) to the previous UNLESS rule, where t_{on} triggers a 'turn' movement, the system stops forward motion and turns, when it comes across an obstacle (thus obtaining a basic action selection mechanism, that arbitrates between two mutually exclusive behaviours). But the possibility of distinguishing the falsehood of cNSBL variables from the lack of support for their truth-value paves the way for other interesting implementations of inhibition mechanisms. In the example shown in Figure 5, the action *forward* is chosen when b is false, and is blocked when b is true. In some cases, one might want the link between a and f_{on} to be inhibited *if behaviour B is active at all* – that is, if B renders b true or false – and to be restored when behaviour B is inactive. This is useful, for example, when the higher layer B is endowed with more efficient and intelligent motor competences, controlling the same actions as the lower layer, where the lower layer is only a back-up controller to be activated in case B is damaged. In such cases, we might want to undertake different actions (incompatible with *forward*) that are associated to the truth or falsehood of b. The rule UNLESS(a, ($b \vee \sim b$), f_{on}) is to be used in this case. The possibility of distinguishing when a behaviour return a *false* value from its inactivity is fundamental for additivity (the possibility of adding other layers on top of the existing ones) and robustness (the capacity of the system to maintain basic functionalities when higher layers don't work) [Brooks, 1986]. This possibility is distinctively offered by cNSBL.

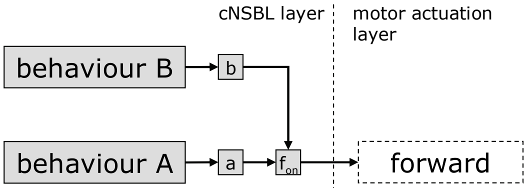


Figure 5 Basic inhibition

Moreover, the possibility of detecting the lack of support for the truth or falsehood of some propositional variable enables one to enrich the inferential capabilities of behaviour-based systems towards non-monotonic reasoning. Figure 6 shows a sketch of an assistive behaviour-based robot, capable of pushing and moving heavy objects in collaboration with humans. The system chooses actions to execute next on the basis of several factors, including the motion of the human user and various properties of the environment. It includes also a module that detects if the user intends to change the previously planned trajectory (because, for example, the environment changed in a way that is not perceivable by the robotic system, or because the user has to engage in a more urgent task). The *human intention detector* module controls a propositional variable a, that is true if the human's plan is (supposedly) changed, and false if the human intention is to continue with the previous plan. All of these reasoning capabilities are modelled as modules taking as inputs many sensory variables, and drawing chains of deductively valid conclusions ultimately resulting in the values of $\{a, \ldots, g\}$ variables. On the basis of those variables, the module *action generator* sets to true a variable included in a set Act=$\{h, \ldots, z\}$ of action triggering variables. It is desirable that the system reacts to the lack of support for *a* in a way that is distinct from the reaction to its truth and its falsehood; for example, if the system cannot detect the intention of the user – because of incomplete information – a cautious reaction would be that of slowing the speed of the end-effector motion, and possibly ask explicitly if the user wants to abandon the current plan. As in the previous example, we have three distinct ways of reacting to three different conditions –true, false, or undefined *a*; and the choice depends non-monotonically on the presence or on the absence of a definite truth-value for *a* among the premise of the inference.

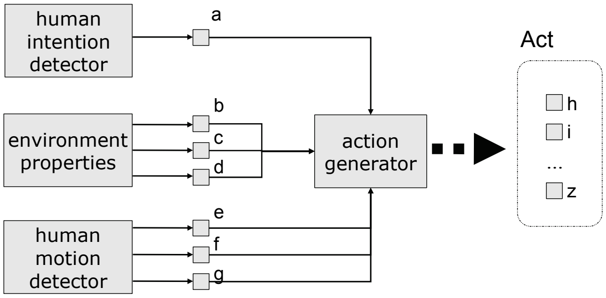


Figure 6 Inhibition and non-monotonic inference

Let's turn now to show how other coordination mechanisms are modelled by means of cNSBL. Discrete actions can be linearly *sequenced* [Arkin and MacKenzie, 1994] by appropriately combining IMPLY rules, and exploiting the motor variables that signal the end of the action. With the following rules a sequence of going backward, turning, and going forward is obtained:

IMPLY(a, $backward_{on}$)
IMPLY($backward_{end}$, $turn_{on}$)
IMPLY($turn_{end}$, $forward_{on}$)

Conditional sequences, like those encoded in FSA diagrams [Arkin, 1989], in Robot Schemas [Lyons and Arbib, 1989] and in teleo-reactive programs [Nilsson, 1994] are achieved by a combination of UNLESS and IMPLY rules that switch between competitive actions when some internal or sensory conditions obtain. Note that behaviours can be inhibited and released by other behaviours, by simply setting their output variables to "neither true nor false"; the same possibility is implemented in many languages for behaviour-based programming, such as the Behavior Language [Brooks, 1990] where behaviours are activated by preconditions. In the following section we describe how to obtain other coordination mechanisms by means of eNSBL.

4 Extended NSBL

cNSBL is not sufficiently powerful to specify some familiar robotic behaviours and control functions. Cooperative coordination mechanisms (as used in potential field navigation, see [Latombe, 1991]) are a prominent case in point. Additional computational tools are introduced here, in order to develop a wholly neuro-symbolic approach to this robotic design problem. In the resulting extended NSBL framework, behaviours are modelled as NSN (corresponding to sets of cNSBL rules), as fibred Neural Nets, (fNN for short, introduced in [d'Avila Garcez and Gabbay, 2004]), or as a combination of both. eNSBL is obtained by representing fNNs as real-valued variables, and by extending the semantics of IMPLY and UNLESS statements so as to admit real-valued variables as arguments. Let A and B be feedforward networks of *n* and *m* neurons respectively, with single output neurons; for each neuron *i* (with 0<*i*<*n* and 0<*i*<*m* respectively), $I_i(t) = \{x_1^i(t),...,x_{k^i}^i(t)\}$ is the vector of k^i inputs of *i*; W_i=$\{W_{ji}\}$, $0<j<k^i$, is the set of the weights of the connections from other neurons to *i*; $U_i(t) = g_i(I_i(t), W_i)$ is the *input potential* of *i*; and $O_i(t+1)= h_i(O_i(t), U_i(t), \Theta_i)$, where Θ_i is a threshold, is the output of *i* at time *t+1*. Let us recall the fibring function and fNN definition given in [d'Avila Garcez and Gabbay, 2004]:

- A fibring function φ_i from A to B maps the weights W_j of B to new values, depending on the values of W_j and on the input potential I_i of the neuron *i* in A;
- B is said to be *embedded* into A if φ_i is a fibring function from A to B, and the output of neural unit *i* in A is given by the output of network B. The resulting network, composed of networks A and B, is said to be a *fibred neural network* (see Figure 7).

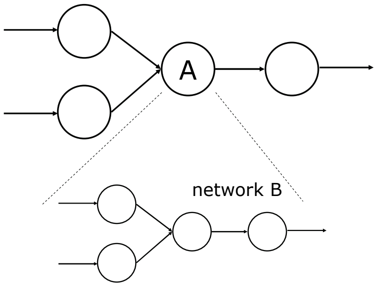


Figure 7 A Fibred Neural Network

eNSBL is obtained from cNSBL by allowing neurons to embed other neural networks via fibring functions. For each embedding neuron *i* of the network, there is a nested fibred neural network *N*N_i, that is, a set $\{N_1, \ldots, N_n\}$ of feedforward neural networks such that $N_i \in \{N_1, \ldots, N_n\}$ is embedded into a neuron of N_{i-1}, $2 \le i \le n$. The network N_1 (at the top of the nesting hierarchy) has its own inputs, possibly connected to sensors or other state variables of the system. The fibring function φ_i associated to the embedding neuron *i* is such that the weights of N_1, …, N_n are set to 0 if the input potential of *i* is below the threshold, otherwise they are set to default values; in the latter case, the default values are such that the nested fibred neural network N_i, calculates a desired function of its inputs. According to the definition of fNN, the output of *i* is given by the output of N_i,. As proved in [d'Avila Garcez and Gabbay, 2004], fibred neural network can approximate any polynomial function to any desired degree of accuracy. More specifically, fNNs may be used to calculate attractive or repulsive potentials, or cooperative coordination among behaviours; and, for each fibred neural network N_i,, the corresponding embedding neuron *i* turns “on” or “off” the embedded network, by intervening on its weights.

The output of neuron *i* is represented as an eNSBL real value (which we refer to by the superscript ‘e’). IMPLY and UNLESS statements are interpreted in eNSBL in a special way. The statement IMPLY(a, b^e) is interpreted as “if *a* is true, then the feedforward network embedded in the NSN neuron represented by b^e will fire, and the value will be stored in b^e itself”. No additional constraints are imposed on the other neurons of embedded networks. Let’s consider the following example of a potential field navigation mechanism based on eNSBL. In potential fields systems, the overall heading of the system is obtained by a vectorial sum of two virtual “potentials”, one attracting the system towards the target position, the other one repelling it from obstacles. Typically, (a) the attractive potential is mathematically represented as a vector, whose direction points towards the goal, and whose magnitude is directly proportional to the distance between current point and goal or some sensory cue; and (b) the repulsive potential is represented as a vector whose direction points away from perceived obstacles, and whose magnitude is inversely proportional to the distance between robot and obstacles. A typical equation for the calculation of the repulsive vector magnitude is

$$V_{magnitude} = \left\{ \begin{array}{ll} 0 & \text{for} \quad x > d \\ 1 - \dfrac{x}{d} & \text{for} \quad 0 \le x \le d \end{array} \right\}$$

where x is the distance perceived by a range detector device (a ultrasonic or infrared sensor), and d is the maximum distance that the sensor can perceive. Figure 8 shows a sketch of the neural circuitry for calculating the potential fields.

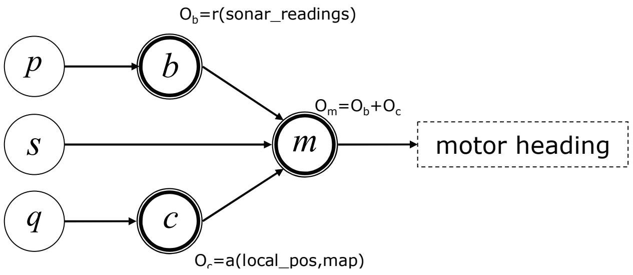


Figure 8 fNN for potential fields navigation

This example includes six NSN neurons, three of which (*b*, *c*, and *m*) embed nested fNNs. The nested fNN embedded in *b* calculates a repulsive potential, with sonar readings as input. As for the vector magnitude, the network embedded in *b* is composed by two neural sub-nets, one calculating the $0 \le x \le d$ part of the equation, and the other being a neuron which fires iff $x > d$ and inhibits the result of the other sub-net (see Figure 9).

The nested fNN embedded into *c* calculates an attractive potential, taking as input the local position of the robot and a map that represents the target position; and the fNN embedded into *m* blends the repulsive and attractive potentials by vectorial sum into one heading to be sent to the motors. Each of the three computations is triggered by a NSN neuron, that is, by a cNSBL variable. The eNSBL program for this network is:

IMPLY(p, b^e)
IMPLY(q, c^e)
IMPLY(s, m^e)

where the truth of propositional variables *p*, *q*, and *s* enables the networks embedded in neurons *b*, *c*, and *m*, respectively, to fire and compute a value for the eNSBL variables b^e, c^e, and m^e (the latter being connected to the motor layer). In particular, by virtue of the last IMPLY statement, variable *s* activates the fNN that calculate motor commands by cooperative coordination [Arkin, 1998].

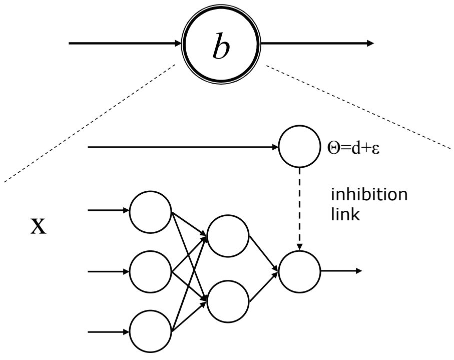


Figure 9 The network for the repulsive potential

By linking each embedding neuron to a triggering neuron (and respectively, to a propositional variable) via IMPLY statements as above, it is possible to design action selection circuitry (including inhibition and sequencing) by cNSBL rules. In Figure 10 the sketch of a simple eNSBL circuitry is shown that arbitrates between four behaviours – avoid obstacles, wander, move to goal, and escape predators. The labels marked with * specify the computation carried out by embedded nested fNNs, whose result is stored in the corresponding variable. Dashed arrows represent inhibitory connections (see section 3).

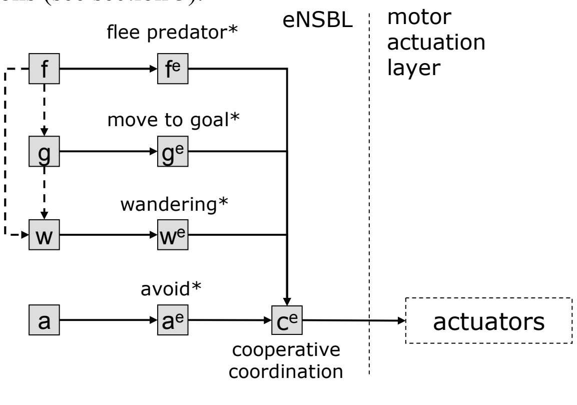


Figure 10 An action selection eNSBL circuitry.

The eNSBL rules

IMPLY(a, a^e)
UNLESS(w, ($g \vee f$), w^e)
UNLESS(g, f, g^e)
IMPLY(f, f^e)

store in variables a^e, w^e, g^e, f^e a repulsive potential, a wandering heading, an attractive potential, and a flee heading respectively; the second rule determines inhibition of the 'wandering' behaviour by the 'move to goal' and the 'flee predator'; the third rule determines the inhibition of 'move to goal' by 'flee predator'. The neural translation of this circuitry is straightforward from the description of eNSBL presented above.

5 Concluding remarks

In this paper we have presented cNSBL, a neuro-symbolic modelling language for behaviour-based systems, and eNSBL, an extended version of cNSBL obtained by fibring neural networks. The expressiveness of cNBSL and eNSBL as a behaviour-based system modelling language has been illustrated by means of various examples of action-selection mechanisms, inference systems for hybrid agents, and potential fields navigation. The close relationship between the cNSBL-eNSBL level and the logical level, inherited by NSL and shown in section 2, represents one of the distinctive advantages with respect to other languages proposed in the field of behaviour-based robot modelling.

Future work will explore limitations and potentialities of cNSBL and eNSBL for behaviour-based modelling. In particular, we will be concerned with adaptation and learning. Adaptation in eNSBL systems can be obtained by changing the perception-action transformation embedded in the eNSBL programs, so as to better fit environmental and task constraints. A straightforward form of adaptation for a competitive coordination mechanism is to modify the priority list of behaviours, by selectively modifying (adding and removing) inhibition links among behaviours. This is readily done in cNSBL, assuming that the set of the different priority orderings of behaviours is reasonably small: a propositional variable is introduced for each priority list; this variable can be activated by some sensory configuration detecting mechanism; IMPLY and UNLESS statements are added to the program, that selectively activate one behavioural output (and inhibit the others) when this propositional variable is true.

More interesting, dynamic forms of adaptation and learning are allowed by eNSBL, by devising proper fibring functions for embedded networks. Reinforcement learning [Nehmzow and Mitchell, 1995] is a case in point. Fibring functions map the current weights of the embedded networks into other weights, depending both on the current weights and the inputs to the embedding neurons. Suppose that an 'avoid' network (calculating a repulsive potential) is embedded in the *avoid* neuron of Figure 11, receiving 0 or 1 inputs from synaptic connections with other neurons of the NSN. If $W_{avoid}(t)$ is the array of weights of the embedded network at time t, a and b are the activation values of presynaptic neurons, and c_1, c_2 are reinforcement constants, a simple form of reinforcement learning operating at run-time is obtained by the following fibring function:

$$W_{avoid}(t+1)=(W_{avoid}(t)+(a*c_1)-(b*c_2))$$

Every time a or b neurons are active, a reinforcement learning step is issued; in particular, if a is active, each weight is incremented by c_1, and if the neuron b is active then the weights are decremented by c_2. Neurons a and b issue positive and negative reinforcement learning, respectively. This simple form of reinforcement learning, similar to the *learning momentum* strategy in behaviour-based robotics [Clark *et al.*, 1992], is readily obtained by fibring functions and embedded networks; it exploits the remarkable feature of fibred neural networks, in which the training of embedded networks follows directly from the firing of the embedding networks [d'Avila Garcez and Gabbay, 2004].

Other reinforcement factors are easily added by taking into account other pre-synaptic neurons in the fibring functions. And higher-level cNSBL modules can guide the learning and adaptation of lower-level embedded networks, by activating and inhibiting reinforcement neurons. This opportunity paves the way to the development of multi-layered systems akin to hybrid behaviour-based architectures [Gat, 1998].

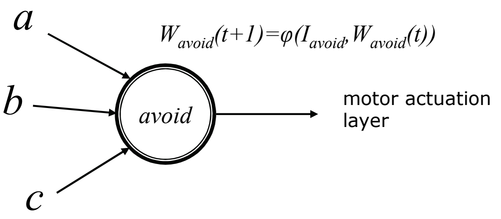


Figure 11 Reinforcement learning in embedded networks

Backpropagation learning may be applied to a knowledge base solely consisting of IMPLY operators of cNSBL, if one modifies in a suitable way the neural representation of these operators. This possibility is demonstrated by the translation algorithm introduced in [d'Avila Garcez and Zaverucha, 1999] from general logic programs to neural networks with bipolar semi-linear neurons, instead of the networks of binary threshold neurons used here for the translation of IMPLY clauses. Both kinds of network provide parallel processing models for sets of IMPLY operators, but only the neural net specified in [d'Avila Garcez and Zaverucha, 1999] can be trained by the standard backpropagation algorithm. This learning procedure can be adopted in order to refine the knowledge base formed by the IMPLY operators that were originally fed into the translation algorithm [Towell and Shavlik, 1994].

Future work on NSBL will be concerned with these learning problems, adopting the distinctive perspective of cognitive robotics, which requires one to perform learning and reasoning without violating bounded-time response constraints. From this perspective, challenging issues are now being posed in perceptual cognition, concerning the development of modules which allow robotic systems to perform categorical classification and object recognition. And crucially, the design and processing opportunities afforded in this domain of investigation by neuro-symbolic approaches in general, and by NSBL in particular, will have to be probed on the basis of both computer simulations and actual robotic platforms.

References

[Aiello *et al.*, 1995] Aiello, A., Burattini, E., Tamburrini, G. (1995), "Purely neural, rule-based diagnostic systems. I, II" *International Journal of Intelligent Systems*, Vol. 10, pp. 735-769.

[Aiello *et al.*, 1998] Aiello, A., Burattini, E., Tamburrini, G. (1998), "*Neural Networks and Rule-Based Systems*", in Leondes C. D. (ed.), *Fuzzy Logic and Expert Systems Applications*, Academic Press, Boston, MA.

[Arkin, 1989] Arkin, R.C. (1989), "Motor Schema-Based Mobile Robot Navigation", *International Journal of Robotics Research*, Vol. 8, No. 4, pp. 92-112.

[Arkin, 1998] Arkin, R.C. (1998), *Behavior-based robotics*, The MIT Press.

[Arkin and MacKenzie, 1994] Arkin, R.C., MacKenzie, D. (1994), "Temporal Coordination of Perceptual Algorithms for Mobile Robot Navigation", *IEEE Transactions on Robotics and Automation*, Vol. 10, No. 3, pp. 276 - 286.

[ARIA] ActivMedia Robotics Interface for Applications (ARIA), see http://robots.activmedia.com/ARIA/index.html

[Bader *et al.*, 2005] Bader S., Hitzler P., Hoelldobler S. (2005), "The integration of connectionism and first-order representation and reasoning as a challenge for artificial intelligence", manuscript.

[Braitenberg, 1984] Braitenberg V. (1984), *Vehicles*, MIT Press, Cambridge, MA.

[Brooks, 1986] Brooks, R.A. (1986), "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, pp. 14-23.

[Brooks, 1990] Brooks, R.A. (1990), "The behavior language", *A.I. Memo 1227, Artificial Intelligence Laboratory, MIT*, Boston, MA.

[Burattini *et al.*, 2000] Burattini, E., De Gregorio, M., Tamburrini, G. (2000), "*NeuroSymbolic Processing: non-monotonic operators and their FPGA implementation*", in *Proceedings of the Sixth Brazilian Symposium on Neural Networks (SBRN 2000)*, IEEE Press.

[Burattini and Tamburrini, 1992] Burattini, E., Tamburrini, G. (1992), "*A pseudo-neural system for hypothesis selection*", *International Journal of Intelligent Systems*, vol. 7, pp. 521-545.

[Clark *et al.*, 1992] Clark, R.J., Arkin, R.C., Ram, A. (1992), "Learning momentum: online performance enhancement for reactive systems", in Proceedings of the 1992 IEEE International Conference on Robotics and Automation, Vol. 1, Nice, France, pp. 111-116.

[d'Avila Garcez *et al.*, 2002] d'Avila Garcez, A. S., Lamb, L. C., Gabbay, D. M. (2002), "A Connectionist Inductive Learning System for Modal Logic Programming", in *Proceedings of 9th IEEE International Conference on Neural Information Processing (ICONIP'02)*, Singapore, November 2002.

[d'Avila Garcez and Gabbay, 2004] d'Avila Garcez, A. S., Gabbay, D. M. (2004), "Fibring Neural Networks", in *Proceedings of 19th National Conference on Artificial Intelligence (AAAI 04)*, San Jose, California, USA, AAAI Press.

[d'Avila Garcez and Zaverucha, 1999] Avila Garcez, A.S., Zaverucha, G. (1999), "The connectionist inductive learning and logic programming system", *Applied Intelligence*, Vol. 11, pp. 59-77.

[Gat, 1998] Gat, E. (1998), "On Three-Layer Architectures", in Kortenkamp, D., Bonasso, R.P., Murphy, R. (eds.), Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems, The AAAI Press, pp. 195-210.

[Hoelldobler and Kalinke, 1994] Holldobler, S., Kalinke, Y. (1994), "Toward a new massively parallel computational model for logic programming" in *Proceedings of the Workshop on Combining Symbolic and Connectionist Processing, ECAI 94.*

[Kleene, 1952] Kleene, S. C. (1952), *Introduction to Metamathematics*, North-Holland, Amsterdam.

[Latombe, 1991] Latombe, J.C. (1991), *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA.

[Lyons and Arbib, 1989] Lyons, D.M., Arbib, M.A. (1989), "A Formal Model of Computation for Sensory-Based Robotics", *IEEE Transactions on Robotics and Automation*, Vol. 5, No. 3, pp. 280-293.

[McFarland and Sibly, 1975] McFarland, D.J., Sibly, R.M. (1975), "The Behavioural Final Common Path", *Philosophical Transactions of the Royal Society of London Series B, Biological Sciences*, Vol. 270, No. 907, pp. 265-293.

[Nehmzow and Mitchell, 1995] Nehmzow, U., Mitchell, T. (1995), "The Prospective student's Introduction to the Robot Learning Problem", *Technical Report Series, UMCS95 -12-6*, the Department of Computer Science, Manchester University.

[Nilsson, 1994] Nilsson, N. (1994), "Teleo-reactive programs for agent control", *Journal of Artificial Intelligence Research*, Vol. 1, pp. 139-158.

[Towell and Shavlik, 1994] Towell G.G. , Shavlik J. W. (1994), "Knowledge-based artificial neural networks", *Artificial Intelligence*, Vol. 70, pp. 119-165.

[Tyrrell, 1993] Tyrrell, T., *Computational Mechanisms for Action Selection*, PhD Thesis, University of Edinburgh, Centre for Cognitive Science, 1993.

[von Neumann, 1956] von Neumann, J. (1956), "Probabilistic logics and the synthesis of reliable organisms from unreliable components", in C.E. Shannon, J. Mc Carthy (eds.), *Automata Studies*, Princeton U. P.

The Principle of Presence:
A Heuristic for Growing Knowledge Structured Neural Networks

Laurent Orseau
INSA/IRISA
35000 Rennes, France
lorseau@irisa.fr

Abstract

Fully connected neural networks such as multilayer perceptrons can approximate any given bounded function provided they have sufficient time. But this time grows quickly with the number of connections. In lifelong learning, the agent must acquire more and more knowledge in order to solve problems growing in complexity. In this purpose, it does not sound reasonable to fully connect huge networks. By applying the point of view of locality, we hypothesize that memorization only takes what one perceives and thinks into account. Based on this principle of presence, a neural network is constructed for structuring knowledge online. Advantages and limitations are discussed.

1 Introduction

We take the lifelong learning point of view [Thrun, 1998]. An agent must be able to re-use its knowledge to learn several problems, which is a key issue for solving more and more complex problems. Re-using knowledge does not only mean to know when to use it, but also to organize knowledge so that it is easily accessible to be re-used. In this paper, we do not focus on how to solve complex problems, but principally on how such organization can be made. In this objective, knowledge representation should be understandable, not only for the user, but more importantly for the agent itself, so that it can access to well-defined concepts. At a given time, the number of concepts the agent knows can be gigantic and *a priori* not necessarily correlated. Fully connected neural networks (FCNN) that learn with backpropagation are interesting tools for solving single problems, but have severe drawbacks in terms of lifelong learning and continual interaction with the environment [Robins and Frean, 1998]. Such networks are often said to require learning times in W^3, W being the number of connections; for big nets, having thousands or even more inputs, outputs and neurons, such times are prohibiting. FCNN are also prone to catastrophic forgetting: learning a single new item can make the network forget all it has learned. Whereas knwoledge may grow unbounded, neurons can even not be added online since weights are randomly initialized and connected to all inputs and outputs: knowledge is thus temporarily lost, if not worse, until the new neuron is optimized. Therefore, we seek for locality [Bottou and Vapnik, 1992] in learning (memorization and optimization).

Based on everyday life observation, we hypothesize that what one memorizes only depends on the concepts that are active at that time and call it the principle of presence. After having investigated the consequences of such a strong hypothesis, a neural network is constructed which advantages and disadvantages are discussed.

2 The Principle of Presence

Simon is playing chess against one of his friends. He is confident he will soon win the game. Unfortunately, his opponent plays a move that crosses his plans. Surprised and disappointed, Simon gives up. Later, in another chess game against his friend, the same situation happens. Simons recognizes it and avoids making the same mistake. He has obviously learned something at the end of the first game. But what has Simon memorized? Has he taken all his knowledge into account, even if he knows billions of things? One should be aware of the really fast learning Simon performed. Does this seem plausible that billions of modification have occured for this single learning example?

A possible answer is meta-learning, e.g. [Hochreiter *et al.*, 2001], which describes how to learn the learning algorithm itself, adapting it to different situations. But this kind of learning requires not only examples of functions, but also examples for each function. Infants learn also very quickly, but they can't have sufficient data to create a good learning algorithm. There must be some kind of heuristic prior to meta-learning.

We argue that Simon has only memorized what is most significant to the situation: what he sees, hears, etc. and what he thinks (see Fig. 1). But what he does not think (music, sport or whatever) is in great quantity and is of little relevance. This is due to the fact that most of one's acquired concepts are totally uncorrelated. This principle is in fact a focusing mechanism.

2.1 Definitions

We first define the principle of presence in a very general framework:

Definition 1 *Concept: a concept is a basic perception skill or action skill or a combination of concepts (macro-concept).*

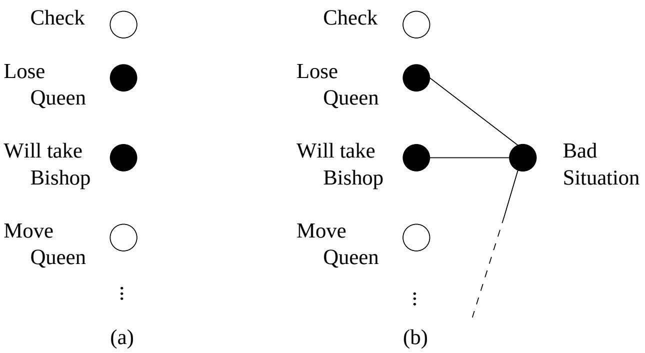


Figure 1: (a) Simon has just seen that willing to take his opponent's bishop led himself to lose his queen. (b) Simon memorizes this new situation by taking only active present perception events and active ideas into account.

A concept cannot be made of itself, though.

Definition 2 *Activity: a concept is said to be active if it is presently used or has been used in the short past.*

It can be considered as still being in a short term memory. If a concept is embodied in a computation unit, using it means that the corresponding unit has delivered a non-zero computation result.

Definition 3 *Principle of presence: when memorising an unknown situation, only active concepts are taken into account.*

This means links, no matter what their function is, are only created between active concepts.

The main advantage of this principle is that the size of the agent's knowledge does not directly influence the number of connections a concept can have.

The principle of presence has reasonable pratice plausibility. Since the number of concepts can be huge and potentially infinite, memorizing a concept by linking it to all the other concepts (no matter how), basic or compound, needs as many resources, which number increases with knowledge. Memorizing only active concepts requires drastically less memory resources: it does not depend on what the agent knows, but rather on what the agent "thinks" at the time of memorization.

In this article, action events will not be considered and only a limited class of macro-concepts will be described.

2.2 Implications

The first consequence is that the working space is not the same for each concept. Links to other concepts depend on which ones were active when the situation was being memorized.

The second consequence is that concepts have only a positive form *at the time of their creation*: they can be in an active state, possibly fuzzy, or in an inactive state. Since inactive concepts are not taken into account, inactivity must exist. However, inhibitory links could be created between existing concepts but for now only the positive form is discussed. Basic concepts (inputs) may only be active to signify the presence of an interesting event; the rarer the better. Higher level concepts must also be created in a similar fashion.

Another consequence concerns generalization. A concept represents the co-activation of other concepts. So they seem to be a conjunction of concepts, thus being monomials *at the time of creation*. Then, generalization is the fact of deleting links from unusefull concepts that are only noise. The goal is to determine which right conjunction of monomials should activate the given concept.

In fact, it can be a bit more than this. Suppose the concept E to learn is **(**A **and** B **and (**C **or** D**))**, A, B, C and D being known concepts. There can be different cases. If A,B,C and D are all present at the time of memorization, then the new concept will be made of these four ones, and first stands for their co-activation. It is then possible to generalize to E, because all needed links to the four concepts already exist. But if the first given co-activation is ABC, then memorization does not take D into account and it becomes harder to model E without adding links. For homogeneity with the worst case, the links created at the same time as the concept only form a monomial (conjunction). Generalization on these links will also keep a monomial form. The final purpose is that once the concept is created and has a stable meaning, new links can be added one by one toward it.

Also, the principle of presence is adapted to online learning since creation of concepts depends on what is *presently* active. It is then possible to quickly learn a new situation, which is important for an agent that is continually interacting with its environment and needs fast reactivity.

Of course, what is active is sometimes not sufficient to take the best decision, and the heuristic of presence loses information: it might happen that the inactivity of a concept is also relevant to the target concept. How to take inactivity into account? One way is to add, for each concept, the complementary concept. But all the interest of the principle is lost, since it is like taking every concept into account. Another way is to create inhibitory connections between a concept which must be active and another one which must not be. The issue is not trivial, though.

In this article, we will focus only on the positive case. Generalization capabilities are reduced, but this is only the first part of a system where dynamics enhance the expression power[1]. So we need generalization skills for representing concepts by eliminating noise links, but it is not needed to be the best possible.

3 Neural Network Implementation

Neural networks (NN) are interesting computing and learning devices since they use the connectivity property of graphs, which is a very general framework, and knowledge acquisition can be statistical. They are also universal approximators and have high generalization capabilities.

The principle of presence has a natural expression in terms of graphs, since concepts depend one on the other. Adding weights on connections enables generalization capabilities. Nodes, embodying monomial concepts, have a threshold to decide when to be active in function of their input concepts.

[1] Note that only the dynamic part of a Turing Machine can generate generalization, since the static part is a raw lookup table.

An interesting propertie of such neurons is that two different concepts cannot (or hardly) be embodied in a single neuron. For example, usual neurons in FCNN can learn to be activated by two independent concepts, e.g. **(A and B) or (C and D)**. This neuron, having then multiple functions, is ambiguous. How can knowledge be re-used if its structure is ambiguous? Every higher neuron must disambiguate the meaning by adding connections to take the context into account. Knowledge is hard to re-use.

Interestingly, all weights being equal, the same neuron also models **(A and C) or (B and D)**, and **(A and D) or (B and C)** even if those examples were not in the training set. This can be easily seen in a MOFN rule [Towell and Shavlik, 1993], which have a similar generalization power as neurons and are of the form:

if (M of the following N antecedents are true) then ...

The neuron computes then:

if 2 **of** $\{A, B, C, D\}$ **then** ***activate***.

If the concepts were really initially independent, they now overlap. In fact, when M is close to $N/2$ there is a combinatorial explosion $\binom{N}{N/2}$ in the number of monomials the neuron can embody, and this is the source of its generalization power: a neuron with one more input can represent twice as many formulas. This is a common issue with subset knowledge extraction algorithms, e.g. [Saito and Nakano, 1988].

Moreover, if during learning the two concepts are actually not independent, they have to be redistributed among other neurons. This kind of operation can lead to temporary loss of knowledge and need re-learning. This is not acceptable in lifelong learning, because if an already re-used concept is moved, not only this concept have to be re-learned, but also all the concepts that were using it. Generalization and ambiguity are very correlated.

Constraining neurons to be either disjunctive or conjunctive avoids ambiguity and the explosion of representation and allows knowledge to be easily re-used. As mentioned by [Towell and Shavlik, 1993], this is equivalent to setting M to either 1 or N.

In monomials, generalization is done by deleting noise variables. Stochastically, this means slightly lowering the weights of variables thought to be noise, and increasing the others.

The principle of presence creates monomial concepts. Some other neurons could represent only disjunctions, where links could be added from monomial neurons one by one. It seems difficult to see how links could be added to create disjunctions if nothing can tell toward which concept such a link should be created. Fortunately, in supervised learning, this is easily done since the teacher points out the target concept; only this kind of macro-concept will be described.

3.1 A First Implementation

In a first approximation, intermediate concepts (in the hidden layer) will only be composed of input events. Only the target concept will have links from intermediate concepts, thus creating a simple 3-layer feedforward network, where hidden nodes are intermediate concepts in monomial form, whereas outputs are targets in disjunctive form. The global network is thus in Disjunctive Normal Form (DNF). DNF are widely used formulas, e.g. [Oyama *et al.*, 2004; Bojarczuk *et al.*, 2004], because they are easy to understand. Knowledge extraction [Tickle *et al.*, 1998; Darbari, 2001] should then be very easy.

In respect of locality, we choose the $\max$ disjunction instead of the usual $+$: indeed, the $+$ operator tends to distribute knowledge among hidden neurons whereas the $\max$ operator does not, since only one neuron is chosen for activating the output. This is rather similar to a Winner Takes All model.

Hidden neurons thus represent monomials. But they are not totally symbolic: as for usual neurons, they compute the weighted sum of input concepts and have an activation function. The constraint is that they must have the possibility to be inactivated. Activation function is then piecewise continuous. The threshold θ is set to $1 - 1/N$, N being the number of incoming links and need not change. The sum of the weights is normalised. Initially, all weights are set equals to $1/N$. In this way, if all links have the same weight (and thus the same importance), their corresponding source neuron must all be activated to activate the neuron. If one link is unusefull, its weight will decrease to zero, whereas the weights of the other links will grow. Since each remaining weight is higher than $1/N$, all of the corresponding inputs must be activated to activate the neuron.

However, if two weights are small enough, the neuron can still have the form of **(A and B and (C or D))**. With four links, the threshold is set to $\theta = 3/4$. If, after learning, $w_A + w_B = 3/4$ and $w_C = w_D = 1/8$, then ABC and ABD activates the neuron to 0.5. Anyway, this is not often meant to happen and the learning algorithm will tend to avoid this.

In what follows, indice k will always be for an output neuron, j for hidden ones and i for inputs.

The hidden neuron j computes its output value on the input vector x by following (2). C_j is the set of neurons which are connected in output to neuron j with weight w_j.

$$S_j(x) = \sum_{i \in C_j} w_{ji}\, x_i \qquad (1)$$

$$y_j(x) = \begin{cases} 0 & \text{if } S_j(x) \leq \theta_j \\ 1 & \text{if } S_j(x) \geq 1 \\ \frac{S_j - \theta_j}{1 - \theta_j} & \text{otherwise} \end{cases} \qquad (2)$$

Ouput neurons compute their value with (3).

$$y_k(x) = \max_{j \in C_k}(0, w_{kj}\, y_j(x)) \qquad (3)$$

In supervised learning, there are two cases of prediction error on a target concept: either it is not sufficiently activated or it is not activated *at all* when it should be. In the first case, the hidden neuron that activated the output neuron is modified by algorithm 1. In the second case, there are two possibilities:

- a new hidden neuron is created,
- an existing hidden neuron is modified.

A criterion could be defined to select which action to do, probably based on a threshold on the minimal distance between the monomial formed by the activated inputs and each hidden neurons. We make the extreme choice of creating a new neuron *and* optimizing the closest existing one.

These mechanics are described in the following sections. Since knowledge is structured *a priori*, there is no need to restructure it later, so that already re-used concepts remain consistent with respect to target concepts.

3.2 Creating Neurons

Initially, the network contains only inputs and outputs, no hidden neuron and no connection.

When a neuron is created, representing a new concept, it is connected to any active input with each weight equal to $1/N$. An output link toward the target neuron is also added, initially set to the value that the concept should have had to correctly activate the target concept: typically 1 for the $\max$ disjunction and $u_k(x) - y_k(x)$ for the $+$, $u_k(x)$ being the desired value of output neuron k for the input vector x.

The principle of presence enables the weights not to be randomized. Adding neurons also makes the system not to be prone to local minima as for backpropagation. At worst, the learning set is entirely memorized. This may not seem interesting for systems that search for the best generalization capabilities, but when addressing the lifelong learning problem, learning totally specific cases *is* necessary.

This is also interesting as the network will not generate random outputs because of unmodified weights: the rejection rate is initially high. This can also be useful for the system *to know if it knows* something, which is a particularly interesting scheme for autonomous agents.

3.3 Optimizing Neurons

When a neuron is inactive, its output is zero. The goal of optimizing a neuron is to eliminate the noise parameters.

Since a newly created neuron is a monomial that is only active for a specific case, neurons are inactive when they should be modified. Backpropagation cannot work in this case.

For each output neuron, since the $\max$ function selects only one hidden neuron, only this one will be modified. This is, in fact, a heuristic for locality in sight of lifelong learning. If the output neuron had a $+$ function, error would be distributed between several hidden units, thus tending to distribute knowledge.

For an output neuron k, if there exists a hidden neuron a such that

$$a = \arg\max_{j \in C_k}(w_{kj}\ y_j(x)), \tag{4}$$

it is modified by algorithm 1, otherwise if there exist a hidden neuron a_p such that

$$a_p = \arg\max_{j \in C_k}(w_{kj}\frac{S_j(x)}{\theta_j}), \tag{5}$$

it is modified by algorithm 1, otherwise no modification is done.

This means that if there exists a hidden neuron which output was used for setting the value of the output neuron, then it is modified. Otherwise, we seek for a neuron that may not be yet general enough.

3.4 Optimization Algorithm

For an output k, once the hidden neuron j has been chosen, then if it exists, it is modified by algorithm 1. α is the learning rate.

Algorithm 1

```
$Er = u_k(x) - y_k(x)$
$\delta = \alpha\ |Er|$
if ($y_i > 0$ and $Er > 0$) or ($y_i = 0$ and $Er < 0$)
   $H_{ji} = \min(\delta, (1 - w_{ji}))$
else
   $H_{ji} = -\min(\delta, w_{ji})$

$D^+ = \sum_{i \in \{n \mid n \in C_j \ \wedge \ y_n > 0\}} |H_{ji}|$
$D^0 = \sum_{i \in \{n \mid n \in C_j \ \wedge \ y_n = 0\}} |H_{ji}|$
$D = \min(D^+, D^0)$

if ($D \neq 0$) $\forall i \in C_j$ :
   if ($y_i > 0$)
      $\Delta w_{ji} = H_{ji}\frac{D}{D^+}$
   else
      $\Delta w_{ji} = H_{ji}\frac{D}{D^0}$
```

The output weight is modified by (6), but is bounded in $[0, 1]$.

$$\Delta w_{kj} = \alpha\ Er\ y_j \tag{6}$$

The boundaries prevent the concept from deviating from the type of meaning it is intended to have. Inhibitory output connections would be in contradiction with the meaning of the hidden neuron. Excitatory links do not need to grow above 1 either.

The key idea is that a fraction of the weights from the “faulty” neurons is re-distributed among the other weights. D^+ (respectively D^0) is the maximum “amount” of weights that can be transfered from active (inactives) ones to (active) inactive ones. If $Er > 0$ ($Er < 0$), inactive (active) weights are moved toward active (inactive) ones.

This algorithm ensures $\sum_{i \in C_j} w_{ji} = 1$, so that $y_j(x_I) = 1$ if x_I is the situation that was used to create the neuron.

It has many differences with the standard backpropagation. The main one is that it is semi-hebbian: weights can be modified upward or downward with only positive or only negative feedback. We call it semi-hebbian because for a given error, some weights are lowered and others are raised although there is no negative input. This can be a very important feature for a robot in continual interaction with its environment: as it must avoid negative reinforcements, it should be able to generalize with only positive examples. Furthermore, if a robot learns by reinforcement learning, it must avoid negative reinforcements (the negative examples) but must still generalize. However, negative examples are sometimes needed.

Another difference is that usually the number of hidden neurons is limited in order to avoid overfitting. Our algorithm generalizes “freely”: for an unknown case, it tries to generalize *and* learns it by heart. Thus generalization is not a property of limiting the number of resources (but still tends to limit it). This is very interesting for lifelong learning, since one never knows if cases are specific or can be generalized. It must then be able to learn by heart. But this may not be interesting if one has to find out a complex approximated function from very limited data: it might rote learn the training set with little generalization. We will see that there are some tricks to force generalization is such cases, though.

The error is also not backpropagated on multiple hidden layers. This is based on the idea that if a concept c_1 uses another concept c_2 in input, c_2 has its own meaning and does not depend on the error generated by c_1. We will also see later that the meaning of c_2 should be stable. But this does not appear in our case since there is only one hidden layer.

This algorithm looks like incremental symbolic concept learning algorithms, e.g. [Sanchez *et al.*, 2002], but the weights allow smooth and very local modifications: only one neuron is slightly optimized after a given example. This can be more accurate for fast on-line statistical learning. This also implies a better robustness to noise compared to non-statistical methods. On the other hand, convergence proofs may be harder to provide.

3.5 A Simple Example

Suppose the system must learn the monomial concept AB and that it is provided with examples such as ABC, ABD, ABE, but not AB.

Initially, inputs are the letters of the alphabet, there is no hidden neuron and the output neuron T is the target concept AB to be learned. There can be as many inputs as wanted: as long as they are not activated, they are not taken into account in the learning process. There is no connection.

The sample ABC is provided to the inputs, thus activating A, B and C but not the other inputs. The network infers 0 because no hidden neuron can give an answer. Since the output should be 1, a new neuron is added and, obviously, no neuron is optimized. The new neuron N_1 has 3 connections, from inputs A, B and C, each having its weight set to 1/3. θ_{N_1} is set to 2/3 and the ouput weight w_{TN_1} is set to 1 (see Fig. 2a). N_1 will then only be activated if A, B and C are simultaneously active, but it will be so also for any vector subsumed by ABC, e.g. ABCD, ABCEF. The network then answers 1 and no modification is needed.

The second sample ABD does not activate N_1: $S_{N_1}(ABD) = 2/3$ and $y_{N_1}(ABD) = 0$. The target neuron is still not activated, so a new neuron N_2 is added with links from A, B, and D and a link toward the output neuron. But this time, an existing neuron can be optimized: $S_{N_1}/\theta_{N_1} = 1$. w_{N_1A} and w_{N_1B} are increased, whereas w_{N_1C} is lowered. w_{TN_1} is not modified because it already has its maximum value (see Fig. 2b). Because weights are kept normalized, the input vector ABC always activate N_1 to 1.

When next presenting ABE, $S_{N_1} > 2/3$ and $0 < y_{N_1} < 1$: N_1 and T are activated, so no neuron is added. But N_1 is optimized, again lowering w_{N_1C} and increasing the two other weights. N_2 is not modified (see Fig. 2c).

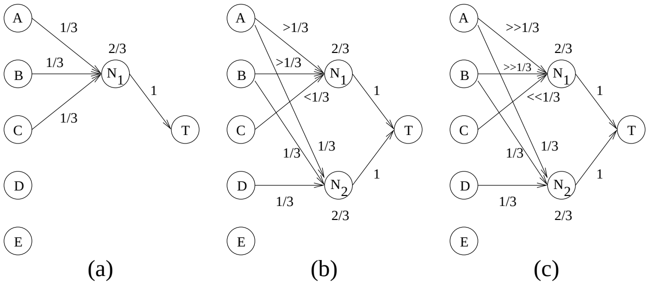


Figure 2: (a) The new neuron N_1 is created with 3 connections. (b) N_1 is generalized while N_2 is created. (c) N_1 is generalized, N_2 is not modified.

After a few more samples, w_{N_1C} will reach 0, whereas w_{N_1A} and w_{N_1B} will be equal to 1/2. N_1 now represents the concept AB as does the target neuron.

Because the learning algorithm is semi-hebbian, generalization has occured only with positive examples, whereas backpropagation needs negative examples to lower weights.

3.6 Neuron Deletion

The neural network may grow fast because it is initially very selective (conjunctions) and generalizes while adding neurons. Many neurons contribute to generalize the others, and are no more needed. They ought to be deleted, as they are redundant. It should be noted, though, that some neurons may stay totally specialized and are never generalized while being important. A deletion criterion has to be defined in this purpose. The algorithm allows to count the number of times each neuron/concept has been used. A simple criterion can then take this into account. A more complex one can also be based on the ratio between the number of times the neuron has received a positive error, meaning it needs to generalize, and a negative error, for specialization. Examples will be given in the experiments.

4 Experiments

In order to determine whether this principle is plausible or not, we tested it on artificial tasks and on a more cognitively plausible one.

4.1 Monks

The artificial Monks tasks are often used to test learning systems [Thrun *et al.*, 1991]. There are six variables x_1 to x_6 with respectively 3, 3, 2, 3, 4 and 2 modalities. The output is binary. Each variable is 1-of-k encoded on inputs. Results are averaged on 10 trials per task.

First Task

The rule to learn is ($x_1 = x_2$ or $x_5 = 1$). There are 124 learning examples selected from the 432 possibilities, with no noise.

The problem is in DNF and should be easily learned in 4 rules. But data is sparse and there are few positive examples, which the system needs to generalize. To simulate forcing generalization by limiting resources, a criterion for neuron deletion is used between each pass through the training set. A neuron is deleted if it has not been activated more than 5 times or if $N^+/(N^- + 10) \leq 0.3$, N^- and N^+ being the number of negative and positive errors $u_k(x) - y_k(x)$ the neuron received. α is set to 0.3 and training stops when there are no more errors on the training set.

80% of the time, the network learns the minimal rule set with 4 neurons, acheiving 100% correctness after a mean of 6 passes through the training set. This is 50 times faster than for backpropagation, which also acheives 100% correctness. Sometimes a rule is divided in two parts or specific cases are not deleted. The remaining times (20%), the network have more difficulty to find a stable state and often delete generalized neurons. Around 28 passes are then needed for a maximum of 5% of error. There is no error on the training set, but a rule is split and leaves some unseen cases aside. Maybe a better deletion criterion could avoid this.

Second Task

The rule to learn is **exactly two of the six attributes have their *first* value**.

This is in MOFN form and the system cannot generalize to the desired formula. All it does is learning the training set of 169 examples by heart. When $\alpha = 0.1$ and without deletion criterion, the rate of success is around 76.6% with 54 neurons. After adding the complementary inputs and with the same deletion criterion as for the first task, the mean rate is 85.4% with 21 neurons.

Third Task

The rule to learn is **($x_5 = 3$ and $x_4 = 1$) or ($x_5 \neq 4$ and $x_2 \neq 3$)**. There was 5% noise (misclassifications) in the training set of 122 examples.

α is set to 0.3 and the learning phase is stopped after 50 passes. At the end of learning, neurons are deleted if they have been activated less than 5 times. Here, the criterion of the first task is not valid because of the misclassifications: neurons with many errors cannot be considered as unusefull. Since the rule contains negative parts, the principle of presence alone is not sufficient to learn it: there were about 26 errors on the training set and the system acheived 68.8% correctness on the test set.

If complementary inputs are added, it rises to 87.2% in a average of 34 passes, sometimes acheiving exactly as good as backpropagation with weight decay on a standard network with 97.2% of correctness. In fact, in this latter case, there is only one neuron with two non-zero connections representing ($x_5 \neq 4$ and $x_2 \neq 3$). Again, a better deletion criterion may lead to more frequent high results.

4.2 The NETtalk Task

[Sejnowski and Rosenberg, 1988] proposed a neural architecture for mapping input letters to phonemes. Several tasks are described but we focus on learning from the 20,012 words dictionary. A window of seven letters is presented to the network, which has to predict the phoneme corresponding to the central letter. The inputs are the 26 letters of the alphabet plus one “Silence” input. A distributed output representation of phonemes was used. For simplicity outputs are the 54 phonemes and intonations. This does not seem to change the results. Words are selected in random order and are moved trough the window. The best results were obtained with 120 hidden neurons, reaching a performance of 90% after more than 5 passes through the dictionary. The purpose is not to beat NETtalk performance, but to show the validity of the principle of presence, even with the basic implementation.

Table 1: Results for the NETtalk task; N_F is the number of frozen neurons, N_N is the total number of neurons at the end of the sequence of words, N_S is the number of connections.

Words	N_F	N_N	N_S	%
1000	0	3939	31512	68.5
2000	458	4445	35560	74.5
3000	841	2124	16992	74.6
4000	1049	3283	26264	76.0
5000	1266	4640	37120	76.6
9000	2055	2055	16440	78.3

Table 2: Results for the NETtalk task after 5000 words; C_D is the minimum number of activations a neuron must have, N_N is the number of neurons in the net, all of them satisfying C_D, N_S is the total number of connections.

C_D	N_N	N_S	%
0	10543	84344	77.0
1	4419	35352	75.4
2	2516	20128	75.4
16	753	6024	74.0
256	240	1920	69.6
2048	76	608	63.7
4096	48	384	59.2
8192	30	240	46.9

In NETtalk, each neuron had 210 connections, making a total of 25200. In our case, each neuron has 8 connections, because there is only one active letter at the same time. Of course, the expression powers are completely different.

α is set to 0.05. When the number of non-frozen hidden neurons exceeds 4000, those having less than 5 activations are deleted and the others are frozen. Neurons have also been deleted after the last step. Such a buffered long-term memory is interesting for lifelong learning agents: it allows to quickly memorize very specific examples and to keep them only if they are useful enough. Results are reported in Table 1. After 5000 words, performance did not improve sufficiently compared to the number of neurons added: it was learning the set by heart.

Table 2 shows the consequences on the performance when deleting neurons. Neurons have not been frozen during learning of 5000 words. Those with too few activations are then deleted. Many neurons are used only for very specific cases and have been added to help generalizing another neuron. This criterion seems accurate because performance decreases slowly compared with the large number of neurons deleted. The effects of free generalization are appearing: despite the number of neurons, the system has really generalized.

Of course, if the complementary inputs had been given instead of the letters – the unit is active if and only if the letter is not present – the results would have been dramatic. The principle of presence seem to be seriously correlated with the way people represent things.

Results show that our basic implementation generalizes quite well and may have a real cognitive plausibility.

5 Discussion

The temporal framework is a typical case where this principle can eliminate many *a priori* unusefull weights. For examples, a fully connected feedforward network with the 26 letters of the alphabet encoded 1-of-k in input, with 1 hidden neuron, and taking k time steps into account, needs $26k$ connections. With the principle of presence, each neuron would have k connections: for the same number of weights, there can be 26 times more neurons. If the problem to learn needs the generalization skills we described, then the principle of presence is an interesting heuristic.

If two unrelated problems use completely different input sets, the principle of presence will not make them interact. This allows an agent to learn different basic concepts a lot more quickly than if the same ambiguous learning resources were used.

Even if the implementation we give yet achieves fair results, it does not address some important issues. It seems that our more important concern will be to define the stability of the representation of a concept and how it can then be re-used. It could be a refined definition of the freezing process described in the NETtalk task.

5.1 Limitations

If the user has a fixed number of inputs, a limited training set and needs the best generalization possible, the principle of presence is not appropriate.

The purpose of the described system is not to address learning as it is generally done for single problems. It is not intended to be used for finding the best approximation from a limited training set. On the contrary, the framework it is meant to be used in is lifelong learning with continual interaction with the environment, which is quite the opposite of problems addressed by FCNN and backpropagation.

5.2 Possible Extensions

From the basic implementation, some extensions are being developped in order to enable the full representation power of the framework described in section 2.

Disjunctions and Inhibitory Links

The target concept is in disjunctive form. But when it was first created, it was a simple monomial. For example, it can first be an abstract concept such as a word; later, a disjunctive link from the concept of the image representing the meaning of that word can be added. A concept is then a disjunction between its own monomial and the other concepts.

For simplicity, newly created concepts have been reduced to monomials. However, it may be interesting to lower the threshold θ, so that even incomplete information can activate the proper concept. But we emphasize that independent concepts must not be embodied in a single unit.

The expression power is yet limited by the fact that connections have only positive weights: if a neuron must represent a general rule to which it exists an exception case, it cannot learn to exclude a single fact and will only learn the general rule with a lowered ouput weight. To solve this problem, we mentioned the fact that the system will be used with dynamics, but there is also a static way to address this issue. Still in the supervised learning paradigm, it is possible to add a new concept representing the exception case and to inhibit the general one. As for disjunctions, this may be easier with supervised learning because the target concept to be modified is pointed out.

Partially connecting neurons generate an automatic subsuming, which is sometimes not wanted: if neurons N_1 and N_2 represent respectively the two *different* concepts AB and ABC, then ABC activates both N_1 and N_2. The $\max$ output function can then select N_2 if its output weight is greater than the one of N_1. Currently, even if the learning phase will tend to do this as in the NETtalk task, it does not sound acceptable for knowledge representation: N_1 should not be active. Here again, an inhibitory link from N_2 to N_1 could handle this.

Action Concepts

The system we propose does not take action concepts into account, since this requires an action selection scheme we do not present here. Actions are not outputs because the purpose of the system is to learn with reinforcements.

Macro-concepts

Macro-concepts allow to re-use already acquired knowledge and thus it is not needed to re-learn complex concepts that are used in other contexts: a simple link from this concept to the new one is sufficient. Any neuron can then be considered as an input and the principle of presence can create a link from this concept toward a new one.

A concept should be re-used only once its meaning is stable, because the meaning of the higher concept would change at the same time the meaning of the lower concept is changed. As there can be many levels, knowledge would then be very unstable.

Another problem is that locality is not entirely respected: when a macro-concept is active, the concepts it is made of are also active. When a new concept is created, links from the macro-concept and all its components are all taken into account. The number of connections would thus grow with the size of knowledge (but still very slowly compared to FCNN). One would prefer to create only a link from the macro-concept, saying that activations of its components are “eaten” by the macro.

But suppose one has three concepts AB, AC, AD. If a new concept EA must be learned from the examples EAB, EAC, EAD, since activations of A are eaten by the concepts AB, AC, AD, then the new concept cannot generalize to EA because there is no link from the concept A. If EA alone never appears in the environment, it cannot be created as a concept. Other alternatives are being considered.

Unsupervised Learning

It would be interesting to integrate unsupervised learning in order to create general concepts that often occur in the environment, but which are not directly targeted by supervised learning. By combining it with macro-concepts, it could seriously limit the number of connections by finding regularities.

Before learning words, it can learn regularly appearing sequences of words, and then use it in higher level concepts.

6 Related Work

The consequences of the principle of presence has an interesting relation with Minsky's knowledge lines (K-lines) [Minsky, 1980]. K-lines can be made of K-lines or of basic agents, as concepts can be made of concepts or of basic ones. His K-Recursion principle is quite similar to the principle of presence: "it will suffice to attach the new K-line [. . .] to just the currently-active K-nodes!". However, there are yet differences since our concepts are totally feedforward and do not re-activate lower-level concepts. In some way, one might see the principle of presence as a mini-theory of K-lines. Minsky does not describe how generalization occurs.

Béroule [Béroule, 2004] also uses a kind of principle of presence in a temporal framework, creating paths for coincidence detection. Each new node creates a new branch of the path and has connections from the presently active input (not past) and from the last active concept which represent the past. In our framework, it can be seen as adding a neuron after each time step, creating very specialized markovian paths, whereas we wait for some error feedback, which allows to take more past concepts into account. His network is not weighted and does not generalize by deleting meaningless connections.

7 Conclusion

The principle of presence is a heuristic for creating new neurons with connections only from active neurons or inputs. Its most important advantages are that the number of connections does not depend on the number of inputs nor on the size of the knowledge, and structures it. This may be an important principle for lifelong learning robots, continually interacting with their environment.

The main drawback is that the neurons cannot take inactive inputs into account. Actual work aims to fill this gap.

Results show that some tasks are well adapted to this strong heuristic. We believe these are general cognitive tasks. Moreover, the purpose of the proposed architecture is to be used in a system where generalization is also available via dynamics. Based on this architecture, dynamical problems can be addressed with a growing long-term memory and fast on-line generalization capabilities.

References

[Béroule, 2004] D. Béroule. An instance of coincidence detection architecture relying on temporal coding. *IEEE Transactions on Neural Networks, Special Issue on Temporal Coding for Neural Information Processing, DeLiang Wang & al. (Eds.),* 15(5):963–979, 2004.

[Bojarczuk *et al.*, 2004] C. C. Bojarczuk, H. S. Lopes, A. A. Freitas, and E. L. Michalkiewicz. A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. *Artificial Intelligence in Medecine*, 30(1):27–48, January 2004.

[Bottou and Vapnik, 1992] L. Bottou and V. L. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[Darbari, 2001] A. Darbari. Rule extraction from trained ANN: A survey. Technical Report WV-2000-03, Knowledge Representation and Reasoning Group, Department of Computer Science, University of Technology, Dresden, Germany, July 2001.

[Hochreiter *et al.*, 2001] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *Lecture Notes on Comp. Sci. 2130, Proc. Intl. Conf. on Artificial Neural Networks (ICANN-2001),* pages 87–94. Springer: Berlin, Heidelberg, 2001.

[Minsky, 1980] M. L. Minsky. K-lines: A theory of memory. *Cognitive Science*, 4:117–133, 1980.

[Oyama *et al.*, 2004] S. Oyama, T. Kokubo, and T. Ishida. Domain-specific web search with keyword spices. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):17–27, 2004.

[Robins and Frean, 1998] A. V. Robins and M. R. Frean. Local learning algorithms for sequential learning tasks in neural networks. *Journal of Advanced Computational Intelligence*, 2(6):107–111, 1998.

[Saito and Nakano, 1988] K. Saito and R. Nakano. Medical diagnostic expert system based on PDP model. In *Proceedings of IEEE International Conference on Neural Networks*, volume 1, pages 255–262. San Diego, CA: IEEE, 1988.

[Sanchez *et al.*, 2002] S. Nieto Sanchez, E. Triantaphyllou, J. Chen, and T. W. Liao. An incremental learning algorithm for constructing boolean functions from positive and negative examples. *Computers and Operations Research*, 29(12):1677–1700, 2002.

[Sejnowski and Rosenberg, 1988] T. J. Sejnowski and C. R. Rosenberg. NETtalk: a parallel network that learns to read aloud. In *Neurocomputing: foundations of research,* pages 661–672. MIT Press, 1988.

[Thrun *et al.*, 1991] S. Thrun, T. Mitchell, and J. Cheng. *The MONK's comparison of learning algorithms. Introduction and survey.* S. Thrun, J. Bala, E. Bloedorn and I. Bratko, Pittsburg, Carnegie-Mellon Univ., 1991.

[Thrun, 1998] S. Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Kluwer Academic Publishers, 1998.

[Tickle *et al.*, 1998] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich. The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6):1057–979, November 1998.

[Towell and Shavlik, 1993] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, October 1993.

Learning Segmentation of Behavior to Acquire Situated Combinatorial Semantics

Yuuya SUGITA **Jun TANI**

Lab. for Behavior and Dynamic Cognition
Brain Science Institute, RIKEN
Hirosawa 2-1, Wako-shi, Saitama 3510198, JAPAN
Email:{sugita, tani}@bdc.brain.riken.jp

Abstract

We present a novel connectionist model for acquiring the semantics of a simple language through the behavioral experiences of a real robot. We focus on the "combinatoriality" of semantics and examine how it can be generated through experiments. Our experimental results showed that the essential structures for situated semantics can self-organize themselves through dense interactions between linguistic and behavioral processes whereby a certain generalization in learning is achieved. Our analysis of the acquired dynamical structures indicates that an equivalence of compositionality appears in the combinatorial mechanics self-organized in the neuronal nonlinear dynamics. The paper discusses the essential differences between the mechanisms of compositionality based on conventional linguistic or computational approach and that based on our proposed dynamical systems approach.

1 Introduction

Compositionality of mental process is essential in various complex cognitive tasks, including linguistic processing. The diversity of linguistic meaning is explained in terms of the infinite number of possible combinations of finite concepts. In other words, the meaning of an unseen sentence can be understood as a combination of the meanings of the known words.

The conventional AI-based models as well as cognitive theories, employ pre-defined "symbols" to generate combinatorial expressive power. The symbolic systems assume that the continuous interaction process between an agent and its environment can be segmented clearly into a set of atomic concepts *a priori* [Roy, 2002; Iwahashi, 2003]. Based on this assumption, various categorization methods are investigated in order to articulate the concepts from the analogue spatio-temporal patterns [Baillie and Ganascia, 2000; Siskind, 2001].

However, we consider that the mechanisms for both conceptual segmentation and composition should be acquired at the same time in a co-dependent way in order to learn the situated combinatorial semantics. Here our question is how a cognitive agent can realize the combinatorial mental manipulations based on the analogue sensory-motor competency.

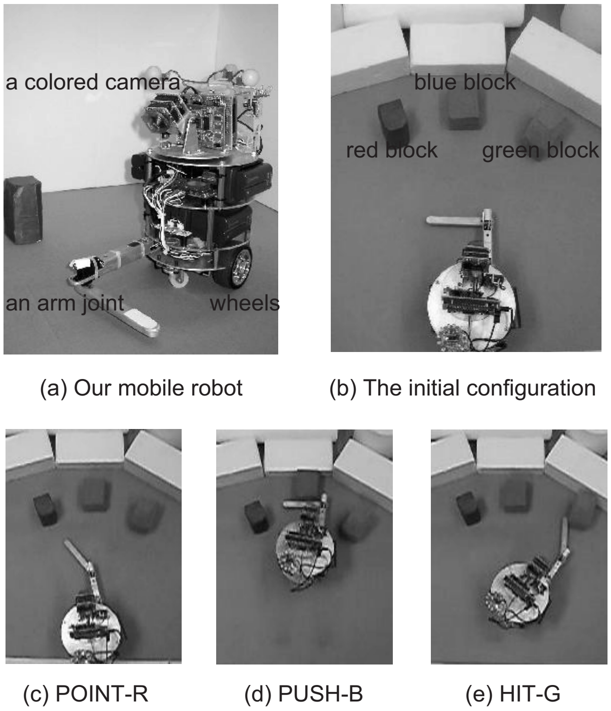
In this paper, we evaluate the generalization capability of our connectionist scheme [Sugita and Tani, 2005], in which a simple embodied language can be acquired without providing any symbolic representations *a priori*. In this scheme, learning is achieved by means of mutual interactions between the linguistic process, dealing with given word sequences, and the behavioral process, dealing with the experienced sensory-motor flow. The hallmark of this approach is the self-organization of the necessary structures for embodied language as the result of such interactions. The proposed scheme is examined by conducting behavior-language acquisition experiments using a real mobile robot. We analyze the types of structures that should be self-organized in order to acquire situated semantics that can exhibit generalization in learning. Our discussion of the results leads to alternative interpretations of the symbol grounding problem and compositionality based on the dynamical systems perspective [Schoner and Kelso, 1988; Beer, 1995; van Gelder, 1998].

2 Task

The experiments are conducted using a real mobile robot with an arm and various sensors, including a vision system. The robot learns a set of behaviors by acting on some objects associated with two-word sentences consisting of a verb followed by a noun. Although our experimental design is limited, it suggests an essential mechanism for acquiring situated compositional semantics through the minimal combinatorial structure of this finite language [Evans, 1981].

The robot experiments consist of the training phase and the testing phase. In the training phase, our neural network model learns a part of possible associations between sentences and corresponding behavioral sensory-motor sequences of a robot in a supervised manner. In the testing phase, the network's ability to generate the corresponding correct behavior by recognizing the given sentences is examined. We also evaluate the system's generalization ability by examining whether appropriate behaviors can be generated from unlearned sentences based on learned sentences.

The mobile robot was built for this experiment in our laboratory. The mobile robot is equipped with three actuators for two wheels and a rotational joint on the arm, a colored vision sensor, and three torque sensors on both the wheels



(a) Our mobile robot (b) The initial configuration

(c) POINT-R (d) PUSH-B (e) HIT-G

Figure 1: The mobile robot (a) starts from a fixed home position in the environment and (b) ends each behavior by pointing at (c), pushing (d), or hitting (e) either the red, blue, or green object.

and the arm (Figure 1a). The robot operates in an environment where three colored objects (red, blue, and green) are placed on the floor (Figure 1b). The positions of these objects can be varied as long as the robot sees the red object (R) on the left-hand side of its field of view, the blue object in the middle (B), and the green object (G) on the right-hand side at the start of every trial of behavioral sequences. We adopt a fixed arrangement of the objects for simplifying behavioral learning, particularly to reduce the required training and computation time for learning. Despite this limitation, our experimental setting still preserves enough complexity to observe the minimal combinatorial properties in associations between sentences and behavioral patterns. The color information is still important for robust behavior generation because the narrow sight of the robot ensures (about 60 degrees) that at least one of the objects is out of sight, except near the starting position.

There are nine behavioral categories that the robot is expected to learn: pointing at, pushing, and hitting each of three objects located on the floor. These categories are denoted as POINT-R, POINT-B, POINT-G, PUSH-R, PUSH-B, PUSH-G, HIT-R, HIT-B, and HIT-G (Figure 1c,d,e). The robot learns these behavioral categories through supervised learning. In order to gather data for supervised training, the sensory-motor sequences corresponding to each of these behavioral categories are generated through the manual-steering of the robot using a remote controller. It should be noted that no categorical cues are provided to the robot in learning, instead the categorical structures should be self-organized only through experiencing various sensory-motor sequences and associated sentences provided during training.

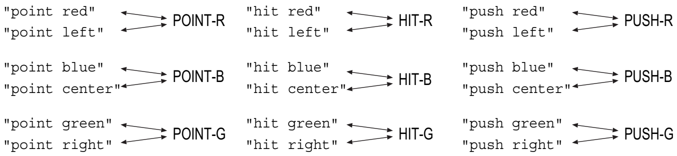


Figure 2: The correspondences between sentences and behavioral categories. For each behavioral category, there are two corresponding sentences.

The robot learns sentences that consist of one of the three verbs: `point`, `push`, and `hit` followed by one of the six nouns: `red`, `left`, `blue`, `center`, `green`, and `right`. We note that the labels are introduced for the ease of our understanding. From the robot's point of view, they are merely nine unknown lexical symbols and they should be labeled as w_1, w_2, $\cdots$, w_9. Therefore, the robot cannot get any information regarding the meaning of the word from the word itself. The meanings of these 18 possible sentences are defined in terms of fixed correspondences with the nine behavioral categories (Figure 2). For example, "`point red`" and "`point left`" correspond to POINT-R, "`point blue`" and "`point center`" to POINT-B, and so on.

In these correspondences, because of the fixed arrangement of the objects in the environment, "`left`," "`center`," and "`right`" have exactly the same meaning as "`red`," "`blue`," and "`green`," respectively. These synonyms are introduced to observe how the behavioral similarity affects the acquired linguistic semantic structure. Moreover, it should be noted that any isolated concepts concerning the objects are not presented to the robot. The objects are taught as targets of actions in which their information, such as color, shape, and weight, is inseparably embedded in the sensory-motor flow associated with behaviors. The robot should understand the meanings of the objects in terms of the possible actions carried out upon them, such as pointing at, approaching, pushing, and hitting.

3 Proposed Model

3.1 General Scheme

We propose a connectionist model which acquires the embodied semantics of a simple language on the task design outlined in the previous section. First, this subsection describes the basic ideas of our proposed connectionist model. The details of each computational algorithm, as well as the module architectures employed in the proposed scheme, will be described in the subsequent subsections.

Our model is composed of two loosely coupled connectionist networks referred to as the recurrent neural network with parametric bias nodes (RNNPB) [Tani, 2003; Tani and Ito, 2003], one for the linguistic module and the other for the behavioral module, as shown in Figure 3. The linguistic module learns to recognize a set of sentences, which is represented as sequences of words, while the behavioral module learns a set of sensory-motor sequences. The association be-

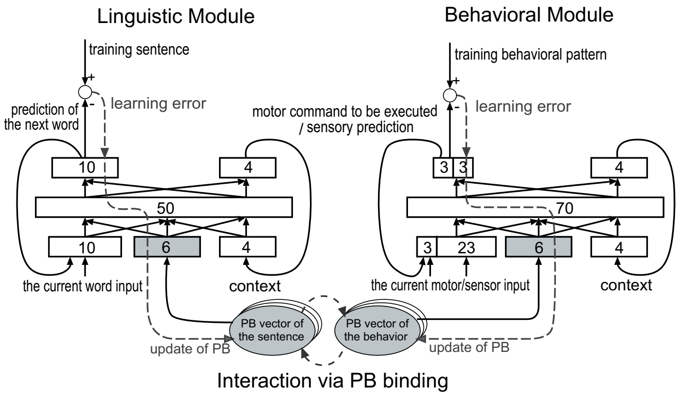

Linguistic Module
Behavioral Module
training sentence
training behavioral pattern
prediction of the next word
learning error
motor command to be executed / sensory prediction
learning error
10
4
50
10
6
4
3
3
4
70
3
23
6
4
the current word input
context
the current motor/sensor input
context
PB vector of the sentence
PB vector of the behavior
update of PB
update of PB
Interaction via PB binding

Figure 3: Our model is composed of two RNNPBs, one for a linguistic module and the other for a behavioral module. Each square represents a set of nodes, and the associated digits denote the number of nodes used in the task. The solid lines denote the information flow in the forward computation and dotted lines denote the flow of the learning error back-propagated to the PB nodes. In the learning process, the PB nodes are iteratively computed through interactions between both the modules.

tween a sentence and its corresponding behavior is achieved by means of self-organization of both modules through their mutual interactions.

The RNNPB is based on the Jordan-type recurrent neural network (RNN) [Jordan and Rumelhart, 1992] but is enhanced with a specialized mechanism for modulating its own dynamic function using the so-called parametric bias (PB) nodes allocated in the input layer. The RNNPB can both generate and recognize sequences, and therefore these functions of RNNPB can be interpreted as an abstract modeling of mirror systems [Rizzolatti *et al.*, 1996]. In the current setting, the RNNPB generates word sequences or sensory-motor sequences in terms of forward models [Kawato *et al.*, 1987] predicting the next state from the current state. In this specific model, the values of the PB nodes are kept constant throughout a run of the generation of a time sequence once their values are fixed at the initial time step.

When the PB vectors are set to different values, the RNNPB exhibits different forward dynamics, i.e., generating different output sequences, whose mechanism is equivalent to the parametric bifurcation, which is well known in nonlinear dynamical systems theory [Wiggins, 1990]. A set of different target output sequences are embedded in an RNNPB by self-organizing an adequate mapping between the PB vector and the output sequences in the learning process. All the training sequences are learned simultaneously through batch training. It should be noted that each PB vector value of a target sequence is self-determined rather than assigned by an experimenter. Once such a mapping is generated, the corresponding PB vector for a given output sequence can be computed inversely by minimizing the prediction error in the output sequence. This can be regarded as the recognition of given sequences in terms of the associated PB vector. This PB computation scheme will later be described in greater detail.

The learning process utilizes the error signal back-propagated [Rumelhart *et al.*, 1986] to the PB nodes, as will be described in detail later. We implement the constraint that the PB vector for corresponding behavioral patterns and sentences should converge to approximately the same value in both modules. In the testing phase, a sentence is first passed to the linguistic module and then the corresponding PB vector value is inversely computed. The obtained PB value is then set in the behavioral module and the corresponding behavioral pattern is generated by the robot. Although the opposite process, i.e., recognizing behavioral patterns and then generating the corresponding sentences, is actually possible, it is beyond the scope of the current paper.

3.2 Algorithmic Description of RNNPB

We review the algorithmic description of the RNNPB [Tani, 2003] before describing the details of each module. The description focuses on how the PB nodes function in generating, learning and recognizing a sequence. In this experiment, every node of the RNNPB yields a real-numbered value between 0.0 to 1.0.

The RNNPB learns multiple sequences $q_0, \cdots, q_{s-1}$ in a supervised manner through modification of two different types of parameters: (1) connection weights W which are common to all training sequences, and (2) PB vectors $p_0, \cdots, p_{s-1}$, each of which is dedicated for a specific training sequence. Both are simultaneously computed using the conventional back-propagation through time (BPTT) algorithm [Jordan and Rumelhart, 1992; Rumelhart *et al.*, 1986] to minimize the value of the total learning error function E over all the training sequences defined as follows:

$$E(W, p_0, \cdots, p_{s-1}) = \sum_{k=0}^{s-1} E_k(W, p_k) \ , \tag{1}$$

$$E_k(W, p_k) = \sum_{t=0}^{l_k-1} \|r_k(t) - o_k(W, p_k, t)\|^2 \ , \tag{2}$$

where E_k is the learning error function of the training sequence q_k, W is a set of all the connection weight values of the network, p_k is a PB vector corresponding to a specific training sequence q_k, s is the number of training sequences, l_k is the length of the training sequence q_k, and $r_{kn}(t)$ and $o_{kn}(W, p_k, t)$ are the target and output values of node n in the training sequence q_k at a time step t, respectively. It should be noted that the output values during generating a sequence q_k depend on both the connection weight values W and the corresponding PB vector p_k, but do not depend on the other PB vectors $p_{k'}, k' \neq k$.

The connection weight values are iteratively computed to minimize the total learning error E as in the conventional RNN. Every connection weight value is initialized randomly and then iteratively updated at every training iteration T in a gradient descent manner by using the BPTT algorithm.

In contrast, the PB vector p_k is computed to minimize the learning error E_k of each training sequence q_k. Each j-th element p_{kj} of a PB vector p_k is initially set to 0.5, and then it is iteratively updated at every training iteration T as follows:

$$\begin{aligned}
\delta p_{kj}^{(T)} &= -\frac{\partial E(W^{(T)}, p_0^{(T)}, \cdots, p_{s-1}^{(T)})}{\partial p_{kj}} \\
&= -\frac{\partial E_k(W^{(T)}, p_k^{(T)})}{\partial p_{kj}} \quad \left(\because\ \frac{\partial E_k}{\partial p_{k'j}} = 0,\ \forall k' \neq k\right), \quad (3) \\
\Delta p_{kj}^{(T)} &= \eta_p \cdot \Delta p_{kj}^{(T-1)} + \varepsilon_p (1-\eta_p) \cdot \delta p_{kj}^{(T)}, \quad (4) \\
p_{kj}^{(T)} &= p_{kj}^{(T-1)} + \Delta p_{kj}^{(T)}, \quad (5)
\end{aligned}$$

where $\delta p_{kj}^{(T)}$ is the delta error back-propagated to the j-th PB node at a training iteration T, which is computed by using the BPTT algorithm. ε_p and η_p are positive coefficients that determine the learning rate and the time constant of the modification of the current update, $\Delta p_{kj}^{(T)}$, of $p_{kj}^{(T)}$, the j-th element of the PB vector p_k.

The recognition algorithm basically follows the same update rules for the PB vectors, shown in equations (3) to (5), where it only updates the PB vector for a given sequence while the connection weight is constant. For the purpose of avoiding local minima, it is effective to introduce a Gaussian noise term that is proportional to the prediction error in equation (5).

3.3 Linguistic Module

The linguistic RNNPB learns and recognizes the sentences. Similar to Elman [1990]'s previous work employing the conventional RNN , our linguistic module is trained to predict the next words in the output nodes from the current word received via the input nodes. A set of sentences can be learned by differentiating the PB vector for each different sentence. This module has 10 input nodes, 6 PB nodes, 4 context nodes, 50 hidden nodes, and 10 prediction output nodes (cf., Figure 3).

The sentences are represented as sequences of words, which always start with a fixed starting symbol. The module has 10 input nodes allocated for nine words (`point`, `push`, `hit`, `red`, `left`, `blue`, `center`, `green`, and `right`) and one starting symbol. Each word is locally represented, such that each input node corresponds to a specific word exclusively activated with 1.0. Although this input representation scheme is almost similar to that of Elman's model, the internal representations of the word sequences are very different. The Elman's model learns the probabilistic distribution of the next possible words while our model learns each sentence as a deterministic sequence encoded in a distinct PB vector (cf., [Miikkulainen, 1993]).

3.4 Behavioral Module

The behavioral module learns the behavioral patterns in order to regenerate them. The module is trained to produce as an output a prediction of the next motor values as well as part of the sensory inputs when it receives the current sensory and motor values as input. All the training sequences are manually prepared by hand-steering the robot in the workspace. They are then used in the off-line learning phase. A training behavioral sequence is sampled with three sensory-motor steps per second during the manual-steering of the robot. The duration of typical behavioral sequences are of approximately 5 to 25 s, and therefore consist of approximately 15 to 75 sensory-motor steps as shown in Figure 4.

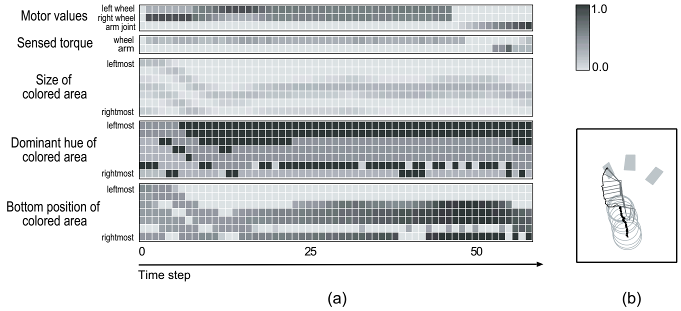


Figure 4: The sensory-motor sequence representation (a) and the corresponding robot trajectory (b) for HIT-R is shown as an example. The robot starts from the home position (step 0). As the robot turns to the red object, the green object soon disappears (step 4), and the red object is at the center of the view (step 10-25). The blue object is still to the right of the view. Subsequently, as the robot moves directly towards the red object (step 25-47), the distance between the red object and robot decreases (the size and the bottom position increases). After this, the robot stops (step 48) and HITs the red object with its arm (step 49-58).

A sensory-motor vector is a real-numbered 26-dimensional vector consisting of 3 current motor values denoting the angular velocities of the 2 wheels and an angle of the arm joint, 2 measured torque values (an average torque value of both wheels, and a torque value of the arm), and 21 values encoding the visual image. The visual field is divided vertically into seven regions, and each region is represented by (1) the fraction of the region covered by the object, (2) the dominant hue of the object in the region, and (3) the bottom border of the object in the region, which is proportional to the distance of the object from the camera. For the region in which there is no colored area, the hue takes a pre-defined constant value 1.0, and the bottom border position takes 0.0, which is designated as distant. In particular, we note that the visual information is not a priority for the acquisition of semantics. It occupies 21 of the 26 dimensions in the sensory-motor vector only due to the characteristic nature of visual information.

The module has 26 input nodes for the sensory-motor vector, 6 PB nodes, 4 context nodes, 70 hidden nodes, 6 output nodes for the 3 motor commands, 2 predicted torque values which will be sensed, and a predicted hue value for the center region of the visual field (cf., Figure 3). The rest of the values of the sensory vector are not predicted in order to reduce the learning time. The module can enable the robot to generate behavior appropriately without predicting the entire sensory-motor vector.

In order to robustly generate each behavioral category, each category has to be trained with multiple samplings of manually guided robot trajectories in which each trajectory is slightly different from the others. This training variability is needed because the robust generation of behavior requires generalization in learning sensory-motor sequences. In order to generate different sensory-motor sequences within the same behavioral categories, the positions of the objects in the workspace are slightly varied (within 20 percent of the distance traveled by the robot) to generate each training sensory-motor sequence.

After successful learning, the robot can generate a learned behavioral pattern from an obtained PB vector. In the actual behavior generation process, the module takes the actual sensory-motor vector as input three times per second and generates the motor commands on the fly. The predicted motor values are used as the actual motor commands for the robot in the next time step.

3.5 PB Binding Method

We have already discussed how the linguistic and behavioral modules learn sentences and behavioral patterns, respectively. In this subsection, the novel associative learning mechanism referred to as PB binding is explained in detail. Both modules are trained at the same time and interact with each other during the learning process. As noted above, the PB binding method imposes the constraint that the PB vectors, for a sentence in the linguistic module and for the corresponding behavioral sequence in the behavioral module, should converge as close as possible to the same value. This constraint is implemented by introducing an interaction term into part of the update rule for the PB vectors in equation (5). During the learning process, the PB vector $p_{s_k}^{(T)}$ of the sentence s_k and the PB vector $p_{b_k}^{(T)}$ of the corresponding behavioral sequence b_k are updated at every training iteration T by means of both the back-propagated error and the mutual interaction as follows:

$$p_{s_k}^{(T)} = p_{s_k}^{(T-1)} + \Delta p_{s_k}^{(T)} + \gamma_L \cdot (p_{b_k}^{(T-1)} - p_{s_k}^{(T-1)}), \quad (6)$$

$$p_{b_k}^{(T)} = p_{b_k}^{(T-1)} + \Delta P_{b_k}^{(T)} + \gamma_B \cdot (p_{s_k}^{(T-1)} - p_{b_k}^{(T-1)}), \quad (7)$$

where γ_L and γ_B are positive coefficients that determine the strength of the binding. Equations (6) and (7) are the constrained update rules for the linguistic module and the behavioral module, respectively. Under these rules, the PB vectors of sentence s_k and behavioral sequence b_k attract each other. In particular, the corresponding PB vectors need not be completely equalized to acquire a correspondence at the end of the learning process. The epsilon errors of the PB vectors can be neglected because of the continuity of the PB spaces.

This binding learning is performed off-line, where the training of both modules is conducted by using all the presented pairs of sentences and the corresponding behavioral sensory-motor time sequences in a single batch. At each iteration in the training, the forward computation and the subsequent backward computation for the BPTT are conducted for all linguistic and behavioral sequences, one at a time. Subsequently, the PB vector for each linguistic and behavioral sequence is updated by equations (6) and (7) and the connection weights of both module networks are updated. This forward and the backward computation for each linguistic and behavioral sequence, computed one at a time does not necessarily require the time step of the sensory-motor sequence and of the word sequence to be synchronized.

In the testing phase, the linguistic and the behavioral module do not work simultaneously. First, the recognition of a given sentence is performed in the linguistic module by iteratively computing the PB vector. Subsequently, the obtained PB vector is set in the behavioral module in order to generate the corresponding behavior.

4 Results

A group of identical learning experiments were conducted with seven different sets of unbound sentences presented in Figure 5 in order to clarify the generalization capabilities of our model. In each experiment, the associations between the behaviors and the sentences were learned by utilizing both modules through 50,000 iterations of learning. The linguistic module learned with a part of 18 possible sentences. The behavioral module learned with 90 sensory-motor sequences, covering all the 9 behavioral categories. These two modules were trained simultaneously using the binding scheme described previously. During this training, each different sentence was bound five times with five slightly different sensory-motor sequences within its corresponding behavioral category. In addition, the behavioral module learned the same 90 behavioral sequences without binding. Without this additional unbound training, the acquired structure in the behavioral module tends to be fragile. This method was necessary because the linguistic regularity is much stronger than the behavioral regularity.

The experiments are classified into three groups based on the training data that is provided. In the experiments belonging to Group 1, four sentences corresponding to two selected behavioral categories that share neither the same target object nor the same action such as pointing at, pushing, or hitting are excluded from the training data (Figures 5a-c). In the experiments belonging to Group 2, the robot is trained with more unbalanced training data with the same number of excluded sentences as in Group 1. Each set of unbound sentences consists of four sentences corresponding to the two selected behavioral categories that share either the same target object or the same action (Figures 5d,e). Group 3 is similar to the first one except with regard to the decreased number of presented sentences. The robot learns twelve sentences apart from the six sentences corresponding to the three selected behavioral categories that share neither the same target object nor the same action (Figures 5f,g). With regard to every set of training data, two trials of experiments were conducted with the different initial connection weight values of the networks.

After the training phase, the network's ability to generate the corresponding correct behavior by recognizing the given sentences is examined. We also evaluate the system's generalization ability by examining whether appropriate behaviors can be generated from unlearned sentences based on learned sentences. With regard to Group 1, five out of six trials were successful. In the successful trials, the robot was able to recognize all the four novel sentences and could generate the corresponding behaviors. With regard to Group 2, none of the four trials were successful. In all the trials, the robot was not able to generalize the training data that was provided, although it was able to understand the sentences that was presented during the learning phase and was able to generate the

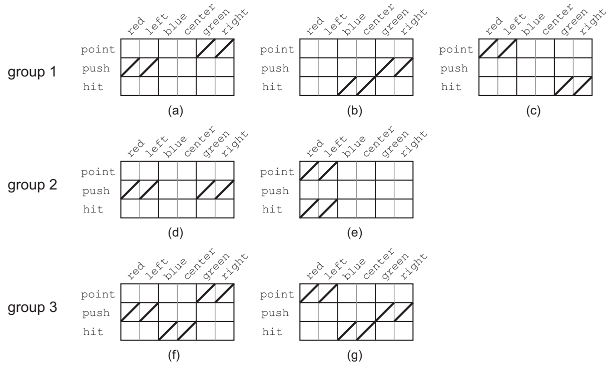


Figure 5: Seven different sets of training data are employed to test the generalization capability of the learning model. The excluded sentences of each training set are represented by dashed boxes. For example, the training data (a) contains 14 sentences apart from "`point green`," "`point right`," "`push red`," and "`push left`."

corresponding behaviors. The network could not extract the underlying regularity from the training data because of the unbalanced presentation of the bindings but not of the sentences since the linguistic module could learn the syntactic regularity from the same set of sentences without interacting with the behavioral module in the additional experiments. With regard to Group 3, only one out of the four trials was successful. In the failure trials, the robot could not generate correct behavior from one or two unlearned sentences.

In summation, we conclude that our connectionist model can acquire a minimal situated combinatorial semantics in very limited conditions. The model is sensitive to both the number of the given sentences and the symmetry of the given examples of bindings, and requires a relatively large fraction of possible bindings in order to realize the correct generalization.

5 Analysis

In order to investigate a possible underlying mechanism for generating the situated combinatorial semantics, we analyze the results of a successful experiment in which the robot learned the training data presented in Figure 5a. In this experiment, four sentences ("`point green,`" "`point right,`" "`push red`," and "`push left`") were eliminated from the training data. As mentioned above, the robot could generate the appropriate behaviors robustly for all the sentences including unlearned ones.

PB mappings for both modules are examined, as shown in Figure 6. All the plots are projections of the PB spaces onto the same surface determined by the PCA method. In this case, the cumulative contribution ratio is approximately 73%. Figure 6a shows the PB vectors obtained as the result of recognizing all 18 legal sentences. The PB vectors for 4 unlearned sentences are surrounded by circles. Figure 6b shows the PB vectors obtained as the result of training the 90 behavioral sequences.

The comparison of the PB mappings between both modules shows that they indeed share a common structure as a result of the binding. The PB vectors of the unlearned sentences and those of the corresponding behavioral sequences successfully coincide without binding during learning. This means that the common global structure emerges from the local PB bindings of each corresponding pair of a sentence and a behavioral pattern. The generalization capabilities of the both modules underlies the self-organization of the situated combinatorial semantics.

In these figures, we can observe some structural properties which reflect the underlying regularities of the provided training data. In Figure 6a, we can find two different groups of congruent constellations among the plots of the PB vectors encoding the sentences. All three PB constellations of the first group, each of which is made up of six PB plots for the sentences which have a specific verb, appear to be congruent. Similarly, six congruent constellations of the second group can be observed for the three sentences which have the same noun. Thus, the combinatorial relationship between the verbs and the nouns is well represented in the product of these two congruent structures. It should be noted that this structure was self-organized without learning all 18 possible sentences. This sort of generalization is accomplished since each sentence is acquired in the form of relational structure among others, rather than as an independent instance.

The same geometric regularity is observed in the behavioral PB mapping shown in Figure 6b. Clusters of PB vectors can be seen for each behavioral category, and the congruent structure can be seen among them. The distributions in each cluster are due to the perturbation in the sensory-motor sequences in the training data.

The linguistic module affects the behavioral module, allowing the organization of the observed congruent structure. This congruent structure among the clusters cannot self-organize when the behavioral module does not have an appropriate structural interaction with the linguistic module because of the lack of binding or the unbalanced binding.

On the other hand, the behavioral constraints can also affect the structure self-organized in the linguistic module. In Figure 6a, the PB vectors for pairs of sentences ending with "`red`" and "`left`", "`blue`" and "`center`", and "`green`" and "`right`", are quite close. This is due to the fact that these pairs of nouns have the same meaning in terms of the associated behavior in our specific task. Recall that the clusters of PB vectors are found for each behavioral category in the behavioral PB mapping. Without this behavioral constraints, six PB vectors for sentences which have a specific verb are located at each vertex of a 5-dimensional regular hyper-polyhedron since every noun has identical syntactic role.

In addition with that, the linguistic module learns the underlying syntax, or the correct order of words in a sentence. The module can not correctly recognize and generate ungrammatical sequences of words, which do not consist of a verb followed by a noun. There exists no PB vector which can generate any ungrammatical sequences of words precisely, although the module can generate every unlearned sentences as well as learned ones. As described in the earlier section, a given word sequence is recognized by means of searching the optimal PB vector for minimizing the error between the tar-

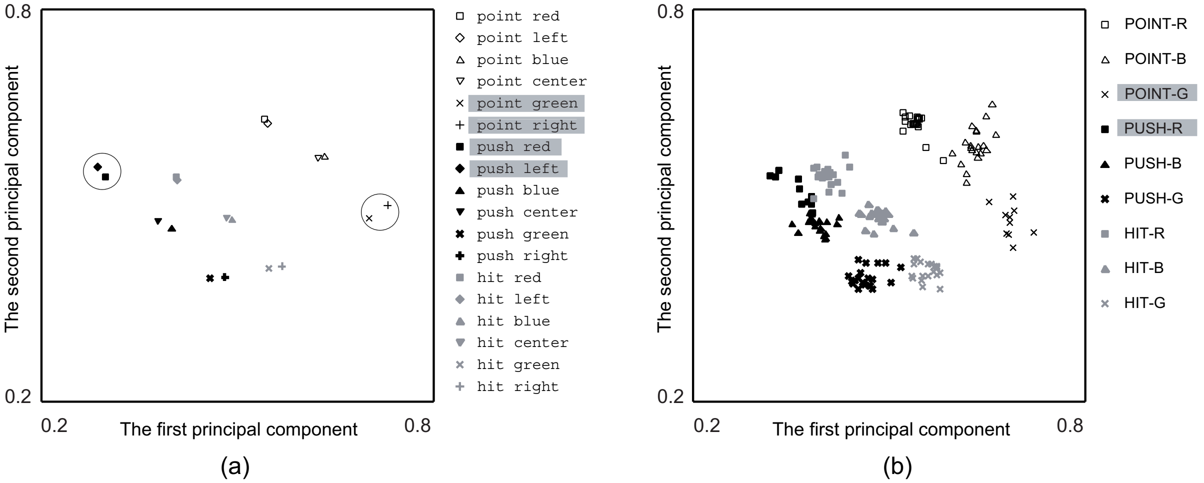


Figure 6: PB plots of the linguistic module (a) and the behavioral module (b) are shown. Both plots are projections of the PB spaces onto the same surface determined by the PCA method. Unlearned sentences and their corresponding behavioral categories are hatched.

get sequence and the output sequence, referred to as a recognition error. However, the module generates a significantly larger recognition error for every illegal sentences than for the grammatical ones.

It is also noted that the robot achieved each goal-directed behavior quite robustly against perturbations. For example, even when the object was moved slightly during executing some behaviors, the robot could follow the directional changes of the target as long as the target was within its vision sight. This robust behavioral generation is realized by generalizing the provided multiple sensory-motor sequences into the sensory-motor mapping through the learning process.

Based on these observations, one may conclude that certain generalizations within every module involve the intermodule associative generalization. Especially, the introduction of combinatorial structure into the behavioral PB mapping realizes the combinatorial interpretation of the behavioral patterns. This can be regarded as the segmentation of a behavioral pattern into verb and noun parts, nevertheless no explicit segmentation techniques is employed. This will be discussed intensively in the next section.

6 Discussions and Summary

Finally, we examine how a cognitive agent can situate the meaning of a sentence or a combinatorial semantic concept in a certain task through the learning process. For this purpose, we clarify the difficulties of this problem by referring to some conventional symbolic AI studies. The symbolic approach employs two important theoretical foundations in order to realize the situated combinatorial semantics: (1) the symbol grounding problem [Harnad, 1990], and (2) the principle of compositionality [Evans, 1981]. According to this formalization, the model is decomposed into two mechanisms: (1) the representation of a grounded atomic concept, and (2) the rules for combining multiple elemental concepts into a grounded combinatorial concept (cf., [Roy, 2002; Iwahashi, 2003; Steels and Baillie, 2003]).

We identify one of the difficulties in the embodied linguistic processing as the inevitable dependence of these two mechanisms in terms of the segmentation of the flow of the (sensory-motor) information involving the continuous interaction of a cognitive agent in an environment [Tani, 1996; 2003]. One assumption about the semantic representation can easily affect the mechanism of the whole system because of the dependence. Consider the case that a concept of "an object in an environment" is provided for conceptualizing some temporal event patterns, such as a manipulation of objects (e.g., [Baillie and Ganascia, 2000]). The introduction of a concept of an object is based on the idea that a temporal event can be represented as a combination of an action concept and its target, namely, an object concept. This means that both an action concept and a rule for combining an action and objects need to be introduced at the same time as an object is assumed as being a segmented entity.

In a learning task similar to ours, *a priori* segmentation can make it difficult to acquire the embodied language, nevertheless every symbolic model employs *a priori* segmentation. Even when every elemental concept is grounded to some plausible physical entities or events, their combination can not be always grounded easily. In other words, it is quite difficult to learn the situated semantic role of a composition rule, since it is often too abstracted to be learned from the provided examples in a grounded manner. For instance, some symbolic models predefine how the elemental concepts should be combined into the complex one in the form of the typed argument structure [Baillie and Ganascia, 2000; Iwahashi, 2003]. The symbolic approach appears to be plausible from the theoretical point of view, however has essential difficulties at least in our task in which a robot generates a behavioral spatio-temporal pattern from a mental representation.

We proposed an alternative mechanism of the acquisition of the situated combinatorial semantics based on the dynamical systems approach. In order to avoid the difficulty mentioned above, our model does not assume any segmented semantic concepts. The dynamical systems perspective provides a holistic view, where neither the whole can be divided into its parts, nor can the meaning of a sentence be divided into those of its individual words. We could argue that the meaning of "red" can be understood only as being associated with the relevant actions of "point," "push," and "hit," but not by "red" itself. Our examination of the PB mapping has shown that the meaning of each sentence is structured relative to other sentences. The observation of the congruence in the PB mapping indicates that the combinatorial characteristics between verbs and nouns are well extracted and embedded in the neuronal dynamics by means of self-organizing combinatorial mechanics.

In our scheme, there exist no representational entities to be grounded. Instead, the necessary mechanism of combinatoriality can be self-organized in the RNNPB by utilizing the nonlinear dynamical characteristics of the parametric bifurcation. Our objective is not to achieve seamless connections from the elemental representational entities of "concepts" to the physical world. In our approach, both the linguistic and behavioral processes are constructed on analogue dynamical systems. Since these two processes share the same metric space, they can interact in an organic manner. Consequently one dynamical structure emerges as a core mechanism of this cognitive process. Since there are no "symbols" to be grounded in our approach, the symbol grounding problem might be regarded as a pseudo-problem in our dynamical systems views. Since there are no "parts" to be combined, the principle of the compositionality might deserve reconsideration in our view.

To the summary, the current study has shown that the semantic combinatoriality can be realized without introducing any *a priori* segmented concepts but with employing the self-organization capabilities of the dynamical networks in a minimal task setting. We would like to further investigate the other underlying dynamical mechanics of the symbolic phenomena, such as the recursive semantic concepts as appeared in an embedding sentence, the incremental and one-shot learning, the language acquisition from sparse training data, and so on. Also, we point out that our current model does not consider the communicative aspects of language at all, although the meaning of a sentence needs to be situated in a communicative context actually [Vogt, 2003; Steels and Baillie, 2003]. Our final goal is achieved when multiple dynamical communicative agents "see" the symbolic process in their opponents one another (aka., intersubjectivity).

References

[Baillie and Ganascia, 2000] J.-C. Baillie and J.-G. Ganascia. Action categorization from video sequences. In *Proceedings of ECAI 2000*, Amsterdam, 2000. IOS Press.

[Beer, 1995] R.D. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72(1):173–215, 1995.

[Elman, 1990] J.L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[Evans, 1981] G. Evans. Semantic Theory and Tacit Knowledge. In S. Holzman and C. Leich, editors, *Wittgenstein: To Follow a Rule*, pages 118–137. Routledge and Kegan Paul, London, 1981.

[Harnad, 1990] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[Iwahashi, 2003] N. Iwahashi. Language acquisition by robots – Towards New Paradigm of Language Processing –. *Journal of Japanese Society for Artificial Intelligence*, 18(1):49–58, 2003.

[Jordan and Rumelhart, 1992] M.I. Jordan and D.E. Rumelhart. Forward models: supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[Kawato *et al.*, 1987] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural network model for the control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987.

[Miikkulainen, 1993] R. Miikkulainen. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. MIT Press, Cambridge, MA, 1993.

[Rizzolatti *et al.*, 1996] G. Rizzolatti, L. Fadiga, B. Galles, and L. Fogassi. Promotor cortex and the recognition of motor actions. *Cognitive Brain Research*, 3:131–141, 1996.

[Roy, 2002] D.K. Roy. Learning visually grounded words and syntax for a scene description task. *Computer Speech and Language*, 16:353–385, 2002.

[Rumelhart *et al.*, 1986] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. Mclelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986.

[Schoner and Kelso, 1988] S. Schoner and S. Kelso. Dynamic Pattern Generation in Behavioral and Neural Systems. *Science*, 239:1513–1519, 1988.

[Siskind, 2001] J.M. Siskind. Grounding the Lexical Semantics of Verbs in Visual Perception using Force Dynamics and Event Logic. *Artificial Intelligence Research*, 15:31–90, 2001.

[Steels and Baillie, 2003] L. Steels and J.-C. Baillie. Shared grounding of event descriptions by autonomous robots. *Robotics and Autonomous Systems*, 43:163–173, 2003.

[Sugita and Tani, 2005] Y. Sugita and J. Tani. Learning Semantic Combinatoriality from the Interaction between Linguistic and Behavioral Processes. *Adaptive Behavior*, 13(1):33–52, 2005.

[Tani and Ito, 2003] J. Tani and M. Ito. Self-organization of behavioral primitives as multiple attractor dynamics: a robot experiment. *IEEE Trans. on Sys. Man and Cybern. Part A*, 33(4):481–488, 2003.

[Tani, 1996] J. Tani. Model-Based Learning for Mobile Robot Navigation from the Dynamical Systems Perspective. *IEEE Trans. on SMC (B)*, 26(3):421–436, 1996.

[Tani, 2003] J. Tani. Learning to generate articulated behavior through the bottom-up and the top-down interaction process. *Neural Networks*, 16:11–23, 2003.

[van Gelder, 1998] T.J. van Gelder. The dynamical hypothesis in cognitive science. *Behavior and Brain Sciences*, 21(5):615–628, 1998.

[Vogt, 2003] P. Vogt. Iterated learning and grounding from holistic to compositional languages. In S. Kirby, editor, *Language Evolution and Computation, Proceedings of the workshop/course at ESSLLI*, pages 76–86. 2003.

[Wiggins, 1990] S. Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Springer-Verlag, New York, NY, 1990.

Integrating First-Order Logic Programs and Connectionist Systems — A Constructive Approach

Sebastian Bader[1*], **Pascal Hitzler**[2†], **Andreas Witzel**[3]
[1]International Center for Computational Logic, Technische Universität Dresden, Germany
[2]AIFB, Universität Karlsruhe, Germany
[3]Department of Computer Science, Technische Universität Dresden, Germany

Abstract

Significant advances have recently been made concerning the integration of symbolic knowledge representation with artificial neural networks (also called connectionist systems). However, while the integration with propositional paradigms has resulted in applicable systems, the case of first-order knowledge representation has so far hardly proceeded beyond theoretical studies which prove the existence of connectionist systems for approximating first-order logic programs up to any chosen precision. Advances were hindered severely by the lack of concrete algorithms for obtaining the approximating networks which were known to exist: the corresponding proofs are not constructive in that they do not yield concrete methods for building the systems. In this paper, we will make the required advance and show how to obtain the structure and the parameters for different kinds of connectionist systems approximating covered logic programs.

1 Introduction

Logic programs have been studied thoroughly in computer science and artificial intelligence and are well understood. They are human-readable, they basically consist of logic formulae, and there are well-founded mathematical theories defining exactly the meaning of a logic program. Logic programs thus constitute one of the most prominent paradigms for knowledge representation and reasoning. But there is also a major drawback: Logic programming is unsuitable for certain learning tasks, in particular in the full first-order case.

On the other hand, for connectionist systems — also called artificial neural networks — there are established and rather simple training or learning algorithms. But it is hard to manually construct a connectionist system with a desired behaviour, and even harder to find a declarative interpretation of what a given connectionist system does. Connectionist systems perform very well in certain settings, but in general we do not understand why or how.

Thus, logic programs and connectionist systems have contrasting advantages and disadvantages. It would be desirable to integrate both approaches in order to combine their respective advantages while avoiding the disadvantages. We could then train a connectionist system to fulfil a certain task, and afterwards translate it into a logic program in order to understand it or to prove that it meets a given specification. Or we might write a logic program and turn it into a connectionist system which could then be optimised using a training algorithm.

Main challenges for the integration of symbolic and connectionist knowledge thus centre around the questions (1) how to extract logical knowledge from trained connectionist systems, and (2) how to encode symbolic knowledge within such systems. We find it natural to start with (2), as extraction methods should easily follow from successful methods for encoding.

For propositional logic programs, encodings into connectionist systems like [11] led immediately to applicable algorithms. Corresponding learning paradigms have been developed [7; 6] and applied to real settings.

For the first-order logic case, however, the situation is much more difficult, as laid out in [4]. Concrete translations, as in [3; 2], yield nonstandard network architectures. For standard architectures, previous work has only established non-constructive proofs showing the existence of connectionist systems which approximate given logic program with arbitrary precision [12; 9]. Thus the implementation of first-order integrated systems was impossible up to this point.

In this paper, we will give concrete methods to compute the structure and the parameters of connectionist systems approximating certain logic programs using established standard architectures.

First, in Section 2, we will give a short introduction to logic programs and connectionist systems. We also review the standard technique for bridging the symbolic world of logic programs with the real-numbers-based world of connectionist systems, namely the embedding of the single-step operator, which carries the meaning of a logic program, into the real numbers as established for this purpose in [12]. In Section 3, we will then approximate the resulting real function by

[*] Sebastian Bader is supported by the GK334 of the German Research Foundation (DFG).

[†] Pascal Hitzler is supported by the German Federal Ministry of Education and Research (BMBF) under the SmartWeb project, and by the European Union under the KnowledgeWeb Network of Excellence.

a piecewise constant function in a controlled manner, which is an important simplifying step for establishing our results. We will then construct connectionist systems for computing or approximating this function, using sigmoidal activation functions in Section 4 and radial basis function (RBF) architecture in Section 5. Section 6 will conclude the paper with a short discussion of some open problems and possibilities for future work.

2 Preliminaries

In this section, we shortly review the basic notions needed from logic programming and connectionist systems. Main references for background reading are [13] and [14], respectively. We also review the embedding of T_P into the real numbers as used in [12; 9], and on which our approach is based.

2.1 Logic Programs

A *logic program* over some first-order language $\mathcal{L}$ is a set of (implicitly universally quantified) *clauses* of the form $A \leftarrow L_1 \wedge \cdots \wedge L_n$, where $n \in \mathbb{N}$ may differ for each clause, A is an *atom* in $\mathcal{L}$ with variables from a set $\mathcal{V}$, and the L_i are *literals* in $\mathcal{L}$, that is, atoms or negated atoms. A is called the *head* of the clause, the L_i are called *body literals*, and their conjunction $L_1 \wedge \cdots \wedge L_n$ is called the *body* of the clause. As an abbreviation, we will sometimes replace $L_1 \wedge \cdots \wedge L_n$ by body and write $A \leftarrow \mathrm{body}$. If $n = 0$, A is called a *fact*. A clause is *ground* if it does not contain any variables. *Local variables* are those variables occurring in some body but not in the corresponding head. A logic program is *covered* if none of the clauses contain local variables.

Example 2.1. *The following is a covered logic program which will serve as our running example. The intended meaning of the clauses is given to the right.*

$e(0).$ *% 0 is even*
$e(s(X)) \leftarrow \neg e(X)$ *% the successor $s(X)$*
% of a non-even X is even

The *Herbrand universe* $\mathcal{U}_P$ is the set of all ground terms of $\mathcal{L}$, the *Herbrand base* $\mathcal{B}_P$ is the set of all ground atoms. A *ground instance* of a literal or a clause is obtained by replacing all variables by terms from $\mathcal{U}_P$. For a logic program P, $\mathcal{G}(P)$ is the set of all ground instances of clauses from P.

A *level mapping* is a function $\|\cdot\| : \mathcal{B}_P \rightarrow \mathbb{N} \setminus \{0\}$. In this paper, we require level mappings to be injective, in which case they can be thought of as enumerations of $\mathcal{B}_P$. The *level* of an atom A is denoted by $\|A\|$. The level of a literal is that of the corresponding atom.

A logic program P is *acyclic with respect to a level mapping* $\|\cdot\|$ if for all clauses $A \leftarrow L_1 \wedge \cdots \wedge L_n \in \mathcal{G}(P)$ we have that $\|A\| > \|L_i\|$ for $1 \leq i \leq n$. A logic program is called *acyclic* if there exists such a level mapping. All acyclic programs are also covered under our standing condition that level mappings are injective, and provided that function symbols are present, i.e. $\mathcal{B}_P$ is infinite. Indeed the case when $\mathcal{B}_P$ is finite is of limited interest to us as it reduces to a propositional setting as studied in [11; 7].

Example 2.2. *For the program from Example 2.1, we have:*

$$\begin{aligned}
\mathcal{U}_P &= \{0, s(0), s^2(0), \dots\} \\
\mathcal{B}_P &= \{e(0), e(s(0)), e(s^2(0)), \dots\} \\
\mathcal{G}(P) &= e(0). \\
&\quad e(s(0)) \leftarrow \neg e(0). \\
&\quad e(s^2(0)) \leftarrow \neg e(s(0)). \\
&\quad \vdots
\end{aligned}$$

With $\|e(s^n(0))\| := n + 1$, we find that P is acyclic.

A *(Herbrand) interpretation* is a subset I of $\mathcal{B}_P$. Those atoms A with $A \in I$ are said to be *true*, or to *hold*, under I (in symbols: $I \models A$), those with $A \notin I$ are said to be *false*, or to *not hold*, under I (in symbols: $I \not\models A$). $\mathcal{I}_P = 2^{\mathcal{B}_P}$ is the set of all interpretations.

An interpretation I is a *(Herbrand) model* of a logic program P (in symbols: $I \models P$) if I is a model for each clause $A \leftarrow \mathrm{body} \in \mathcal{G}(P)$ in the usual sense. That is, if of all body literals I contains exactly those which are not negated (i.e. $I \models \mathrm{body}$), then I must also contain the head.

Example 2.3. *Consider these three Herbrand interpretations for P from Example 2.1:*

$$\begin{aligned}
I_1 &= \{e(0), e(s(0))\} \\
I_2 &= \{e(0), e(s^3(0)), e(s^4(0)), e(s^5(0)), \dots\} \\
I_3 &= \{e(0), e(s^2(0)), e(s^4(0)), e(s^6(0)), \dots\} \\
I_4 &= \mathcal{B}_P
\end{aligned}$$

$I_1 \not\models P$ since $e(s^3(0)) \leftarrow \neg e(s^2(0)) \in \mathcal{G}(P)$ and $e(s^2(0)) \notin I_1$, but $e(s^3(0)) \notin I_1$. I_2 is neither a model (for a similar reason). Both I_3 and I_4 are models for P.

The *single-step operator* $T_P : \mathcal{I}_P \rightarrow \mathcal{I}_P$ maps an interpretation I to the set of exactly those atoms A for which there is a clause $A \leftarrow \mathrm{body} \in \mathcal{G}(P)$ with $I \models \mathrm{body}$. The operator T_P captures the semantics of P as the Herbrand models of the latter are exactly the pre-fixed points of the former, i.e. those interpretations I with $T_P(I) \subseteq I$. For logic programming purposes it is usually preferable to consider fixed points of T_P, instead of pre-fixed points, as the intended meaning of programs. These fixed points are called *supported models* of the program [1]. The well-known stable models [8], for example, are always supported. In example 2.1, $I_3 = \{e(0), e(s^2(0)), e(s^4(0)), \dots\}$ is supported (and stable), while $I_4 = B_P$ is a model but not supported.

Example 2.4. *For P from Example 2.1 and I_1, I_2 from Example 2.3, we get the following by successive application (i.e. iteration) of T_P:*

$$I_1 \overset{T_P}{\mapsto} I_2 \overset{T_P}{\mapsto} \{e(0), e(s^2(0)), e(s^3(0))\} \overset{T_P}{\mapsto} \dots$$
$$\overset{T_P}{\mapsto} \{e(0), e(s^2(0)), \dots, e(s^{2n}(0)), e(s^{2n+1}(0))\} \overset{T_P}{\mapsto} \dots$$

For a certain class of programs, the process of iterating T_P can be shown to converge[1] to the unique supported Herbrand

[1] Convergence in this case is convergence with respect to the Cantor topology on $\mathcal{I}_P$, or equivalently, with respect to a natural underlying metric. For further details, see [10], where also a general class of programs, called Φ-*accessible programs*, is described, for which iterating T_P always converges in this sense.

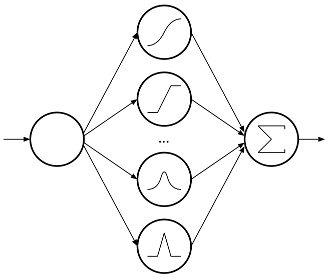


Figure 1: A simple 3-layered feed-forward connectionist system, with different activation functions depicted in the hidden layer.

model of the program, which in this case is the model describing the semantics of the program [10]. This class is described by the fact that T_P is a contraction with respect to a certain metric. A more intuitive description remains to be found, but at least all acyclic programs[2] are contained in this class. That is, given some acyclic program P, we can find its unique supported Herbrand model by iterating T_P and computing a limit. In example 2.4 for instance, the iterates converge in this sense to $I_3 = \{e(0), e(s^2(0)), e(s^4(0)), \ldots\}$, which is the unique supported model of the program.

2.2 Connectionist Systems

A *connectionist system* — or *artificial neural network* — is a complex network of simple computational units, also called *nodes* or *neurons*, which accumulate real numbers from their inputs and send a real number to their output. Each unit's output is *connected to* other units' inputs with a certain real-numbered *weight*. We will deal with feed-forward networks, i.e. networks without cycles, as shown in Figure 1. Each unit has an *input function* which merges its inputs into one input using the weights, and an *activation function* which then computes the output. If a unit has inputs $x_1, \ldots, x_n$ with weights $w_1, \ldots, w_n$, then the *weighted sum* input function is $\sum_{i=1}^{n} x_i w_i$. A locally receptive *distance* input function is $\sqrt{\sum_{i=1}^{n}(x_i - w_i)^2}$. In the case of one single input, this is equivalent to $|x_1 - w_1|$. Those units without incoming connections are called *input neurons*, those without outgoing ones are called *output neurons*.

2.3 Embedding T_P in $\mathbb{R}$

As connectionist systems propagate real numbers, and single-step operators map interpretations, i.e. subsets of $\mathcal{B}_P$, we need to bridge the gap between the real-valued and the symbolic setting. We follow the idea laid out first in [12], and further developed in [9], for embedding $\mathfrak{I}_P$ into $\mathbb{R}$. For this purpose, we define $R : \mathfrak{I}_P \to \mathbb{R}$ as $R(I) := \sum_{A \in I} b^{-\|A\|}$ for some base $b \geq 3$. Note that R is injective. We will abbreviate $R(\{A\})$ by $R(A)$ for singleton interpretations. As depicted in Figure 2, we obtain f_P as an *embedding of* T_P *in* $\mathbb{R}$: $f_P : D_f \to D_f$ with $D_f := \{R(I) | I \in \mathfrak{I}_P\}$, is defined as $f_P(x) := R(T_P(R^{-1}(x)))$. Figure 3 shows the graph of the embedded T_P-operator associated to the program discussed in Examples 2.1 to 2.4.

$$\begin{tikzcd}
I \in \mathfrak{I}_P \arrow[r, "T_P"'] & I' \in \mathfrak{I}_P \arrow[d, "R"'] \\
x \in D_f \arrow[u, "R^{-1}"'] \arrow[r, dotted, "f_P"] & x' \in D_f
\end{tikzcd}$$

Figure 2: Relations between T_P and f_P

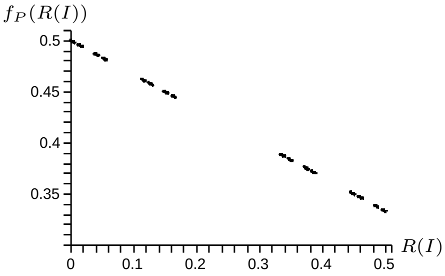


Figure 3: The graph of the embedded T_P-operator from Example 2.1, using base 3 for the embedding. In general, the points will not be on a straight line.

3 Constructing Piecewise Constant Functions

In the following, we assume P to be a covered program with bijective level mapping $\|\cdot\|$ which is, along with its inverse $\|\cdot\|^{-1}$, effectively computable. As already mentioned, we also assume that $\mathcal{B}_P$ is infinite. However, our approach will also work for the finite case with minor modifications. Furthermore, R and f_P denote embeddings with base b as defined above.

3.1 Approximating one Application of T_P

Within this section we will show how to construct a ground subprogram approximating a given program. I.e., we will construct a subset P_l of the ground program $\mathcal{G}(P)$, such that the associated consequence operator T_{P_l} approximates T_P up to a given accuracy ϵ. This idea was first proposed in [15].

Definition 3.1. For all $l \in \mathbb{N}$, the set of atoms of level less than or equal to l is defined as $\mathcal{A}_l := \{A \in \mathcal{B}_P | \|A\| \leq l\}$. Furthermore, we define the *instance of* P *up to level* l as $P_l := \{A \leftarrow \text{body} \in \mathcal{G}(P) | A \in \mathcal{A}_l\}$.

Since the level mappings are required to be enumerations, we know that $\mathcal{A}_l$ is finite. Furthermore, it is also effectively computable, due to the required computability of $\|\cdot\|^{-1}$. It is clear from the definition that P_l is ground and finite, and again, can be computed effectively.

Definition 3.2. For all $l \in \mathbb{N}$, the *greatest relevant input level* with respect to l is

$$\hat{l} := \max\left\{\|L\| \middle| L \text{ is a body literal of some clause in } P_l\right\}.$$

Obviously, we can compute $\hat{l}$ easily, since P_l is ground and finite. The following lemma establishes a connection between the consequence operators of some ground subprogram P_k and the original program P.

[2] In this case the level mapping does not need to be injective.

Lemma 3.3. For all $l, k \in \mathbb{N}$, $k \geq l$, and $I, J \in \mathfrak{I}_P$, we have that $T_{P_k}(I)$ and $T_P(J)$ agree on $\mathcal{A}_l$ if I and J agree on $\mathcal{A}_{\hat{l}}$, i.e.

$$I \cap \mathcal{A}_{\hat{l}} = J \cap \mathcal{A}_{\hat{l}} \quad \text{implies} \quad T_{P_k}(I) \cap \mathcal{A}_l = T_P(J) \cap \mathcal{A}_l.$$

Proof. This follows simply from the fact that I and J agree on $\mathcal{A}_{\hat{l}}$, and that T_{P_k} contains all those clauses relating atoms from $\mathcal{A}_{\hat{l}}$ and $\mathcal{A}_l$. Taking this into account we find that T_P and T_{P_k} agree on $\mathcal{A}_l$. □

Definition 3.4. The *greatest relevant output level* with respect to some arbitrary $\epsilon > 0$ is

$$o_\epsilon := \min\left\{ n \in \mathbb{N} \middle| \sum_{\|A\| > n} R(A) < \epsilon \right\}$$
$$= min\left\{ n \in \mathbb{N} \middle| n > -\frac{\ln(b-1)\epsilon}{\ln b} \right\}$$

The following theorem connects the embedded consequence operator of some subprogram with a desired error bound, which will be used for later approximations using neural networks.

Theorem 3.5. For all $\epsilon > 0$, we have that

$$\left| f_P(x) - f_{P_{o_\epsilon}}(x) \right| < \epsilon \quad \text{for all } x \in D_f.$$

Proof. Let $x \in D_f$ be given. From Lemma 3.3, we know that $T_{P_{o_\epsilon}}(R^{-1}(x)) = R^{-1}(f_{P_{o_\epsilon}}(x))$ agrees with $T_P(R^{-1}(x)) = R^{-1}(f_P(x))$ on all atoms of level $\leq o_\epsilon$. Thus, $f_{P_{o_\epsilon}}(x)$ and $f_P(x)$ agree on the first o_ϵ digits. So the maximum deviation occurs if all later digits are 0 in one case and 1 in the other. In that case, the difference is $\sum_{\|A\|>n} R(A)$, which is $< \epsilon$ by definition of o_ϵ. □

3.2 Iterating the Approximation

Now we know that one application of $f_{P_{o_\epsilon}}$ approximates f_P up to ϵ. But what will happen if we try to approximate several iterations of f_P? In general, $\hat{o}_\epsilon$ might be greater than o_ϵ, that is, the required input precision might be greater than the resulting output precision. In that case, we lose precision with each iteration. So in order to achieve a given output precision after a certain number of steps, we increase our overall precision such that we can afford losing some of it. Since the precision might decrease with each step, we can only guarantee a certain precision for a given maximum number of iterations.

Theorem 3.6. For all $l, n \in \mathbb{N}$, we can effectively compute $l^{(n)}$ such that for all $I \in \mathfrak{I}_P$, $m \leq n$, and $k \geq l^{(n)}$:

$$T^m_{P_k}(I) \text{ agrees with } T^m_P(I) \text{ on } \mathcal{A}_l.$$

Proof. By induction on n. Let $l \in \mathbb{N}$ be given.

base $n = 0$**:** Obviously, $T^0_{P_k}(I) = I = T^0_P(I)$. We set $l^{(0)} := l$.

step $n \rightsquigarrow n+1$**:** By induction hypothesis, we can find $l^{(n)}$ such that for all $I \in \mathfrak{I}_P$, $m \leq n$, and $k \geq l^{(n)}$, $T^m_{P_k}(I)$ agrees with $T^m_P(I)$ on $A_{\hat{l}}$. With $l^{(n+1)} := \max\{l, l^{(n)}\}$, we then have for all $I \in \mathfrak{I}_P$, $m \leq n$, and $k \geq l^{(n+1)}$:

$$T^m_{P_k}(I) \text{ agrees with } T^m_P(I) \text{ on } A_{\hat{l}} \qquad (k \geq l^{(n)})$$
$$\Rightarrow \quad T^{m+1}_{P_k}(I) \text{ agrees with } T^{m+1}_P(I) \text{ on } \mathcal{A}_l \qquad (3.3)$$

$T^0_{P_k}(I) = I = T^0_P(I)$ completes the Induction Step.

□

$$x = 0.\underbrace{0010101101010010}_{\hat{l} \text{ digits are equal}}000000\ldots_b$$

$$x' = 0.\overbrace{0010101101010010}111111\ldots_b$$

Figure 4: Example for the endpoints of a range $[x, x']$ on which f_{P_l} is constant

It follows that for all $\epsilon > 0$, we can effectively compute $o^{(n)}_\epsilon$ such that $\left| f^n_P(x) - f^n_{P_{o^{(n)}_\epsilon}}(x) \right| < \epsilon$ for all $x \in D_f$.

This result may not seem completely satisfying. If we want to iterate our approximation, we have to know in advance how many steps we will need at most. Of course, we could choose a very large maximum number of iterations, but then the instance of P up to the corresponding level might become very large. But in the general case, we might not be interested in so many iterations anyway, since T_P does not necessarily converge.

For acyclic programs, however, T_P is guaranteed to converge, and additionally we can prove that we do not lose precision in the application of T_{P_l}. Due to the acyclicity of P we have $\hat{l} < l$, and hence, with respect to $\mathcal{A}_l$, we obtain the same result after n iterations of T_{P_l} as we would obtain after n iterations of T_P. Thus we can approximate the fixed point of T_P by iterating T_{P_l}. To put it formally, we have that $T^n_{P_l}(I)$ agrees with $T^n_P(I)$ on $\mathcal{A}_l$ for acyclic P and all $n \in \mathbb{N}$. Thus, in this case we find that $|f^n_P(x) - f^n_{P_{o_\epsilon}}(x)| < \epsilon$ for all $x \in D_f$ and all $n \in \mathbb{N}$.

3.3 Simplifying the Domain

Now we have gathered all information and methods necessary to approximate f_P and iterations of f_P. It remains to simplify the domain of the approximation so that we can regard the approximation as a piecewise constant function. We do this by extending D_f to some larger set D_l.

The idea is as follows. Since only input atoms of level $\leq \hat{l}$ play a role in P_l, we have that all $x \in D_f$ which differ only after the $\hat{l}$-th digit are mapped to the same value by f_{P_l}. So we have ranges $[x, x'] \subseteq \mathbb{R}$ of fixed length with x and x' as in Figure 4 such that all elements of $[x, x'] \cap D_f$ are mapped to the same value. Obviously, there are $2^{\hat{l}}$ such ranges, each of length $\sum_{\|A\|>\hat{l}} R(A)$. So we can extend f_{P_l} to a function $\hat{f}_{P_l}$ which has a domain consisting of $2^{\hat{l}}$ disjoint and connected ranges and is constant on each of these ranges. Additionally, the minimum distance between two ranges is greater than or equal to the length of the ranges.

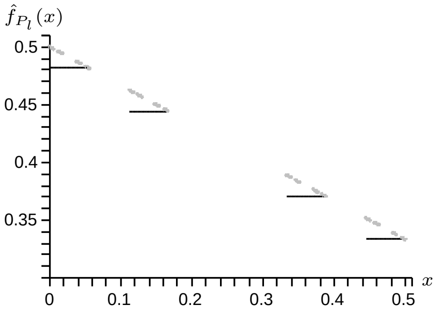


Figure 5: Example for the graph of $\hat{f}_{P_l}$ with $\hat{l} = 2$; f_P is shown in grey.

The resulting graph of $\hat{f}_{P_l}$ will then look similar to the one shown in Figure 5. We formalise these results in the following.

Definition 3.7. An ordered enumeration of all left borders $d_{l,i}$ can be computed as

$$d_{l,i} := \sum_{j=1}^{\hat{l}} \left(\begin{cases} b^{-j} & \text{if } \left\lfloor \frac{i}{\hat{l}-j+1} \right\rfloor \mod 2 = 1 \\ 0 & \text{otherwise} \end{cases} \right).$$

Each of the intervals has length

$$\lambda_l := \sum_{\|A\| > \hat{l}} R(A) = \frac{1}{(b-1) \cdot b^{\hat{l}}}.$$

Finally, we define

$$D_l := \bigcup_{i=0}^{2^{\hat{l}}-1} D_{l,i} \qquad \text{with } D_{l,i} := [d_{l,i}, d_{l,i} + \lambda_l].$$

Thus, D_l consists of $2^{\hat{l}}$ pieces of equal length.

Lemma 3.8. For all $l \in \mathbb{N}$, we have $D_l \supseteq D_f$.

Proof. Let $l \in \mathbb{N}$ and $x \in D_f$. Then there is a $d_{l,i}$ which agrees with x on its $\hat{l}$ digits. But $D_{l,i}$ contains all numbers which agree with $d_{l,i}$ on its $\hat{l}$ digits, thus $x \in D_{l,i} \subseteq D_l$. □

Lemma 3.9. For all $l \in \mathbb{N}$, the connected parts of D_l do not overlap and the space between one part and the next is at least as wide as the parts themselves.

Proof. The minimum distance between two parts occurs when the left endpoints differ only in the last, i.e. $\hat{l}$-th, digit. In that case, the distance between these endpoints is $b^{-\hat{l}}$, which is $\geq 2 \cdot \lambda_l$ since $b \geq 3$. □

Lemma 3.10. For all $l \in \mathbb{N}$ and $0 \leq i < 2^{\hat{l}}$, f_{P_l} is constant on $D_{l,i} \cap D_f$.

Proof. All atoms in bodies of clauses in P_l are of level $\leq \hat{l}$. Thus, T_{P_l} regards only those atoms of level $\leq \hat{l}$, i.e. T_{P_l} is constant for all interpretations which agree on these atoms. This means that f_{P_l} is constant for all x that agree on the first $\hat{l}$ digits, which holds for all $x \in D_{l,i} \cap D_f$. □

Definition 3.11. The *extension of* f_{P_l} *to* D_l, $\hat{f}_{P_l} : D_l \to D_f$, is defined as $\hat{f}_{P_l}(x) := f_{P_l}(d_{l,i})$ for $x \in D_{l,i}$. From the results above, it follows that $\hat{f}_{P_l}$ is well-defined.

Now we have simplified the domain of the approximated embedded single-step operator such that we can regard it as a function consisting of a finite number of equally long constant pieces with gaps at least as wide as their length.

In the following, we will construct connectionist systems which either compute this function exactly or approximate it up to a given, arbitrarily small error. In the latter case we are facing the problem that the two errors might add up to an error which is larger than the desired maximum error. But this is easily taken care of by dividing the desired maximum overall error into one error ϵ' for $f_{P_{o_{\epsilon'}}}$ and another error ϵ'' for the constructed connectionist system.

4 Constructing Sigmoidal Feed-Forward Networks

We will continue our exhibition by considering some arbitrary piecewise constant function g which we want to approximate by connectionist systems. Since $\hat{f}_{P_l}$ is piecewise constant, we can treat this function as desired, and others by the same method. So in the following, let $g : D \to \mathbb{R}$ be given by

$$D := \bigcup_{i=0}^{n-1} [a_i, c_i], \quad c_i = a_i + b, \quad c_i < a_{i+1},$$
$$g(x) := y_i \text{ for } x \in [a_i, c_i].$$

When we construct our connectionist systems, we are only interested in the values they yield for inputs in D. We do not care about the values for inputs outside of D since such inputs are guaranteed not to be possible embeddings of interpretations, i.e. in our setting they do not carry any symbolic meaning which can be carried back to $\mathfrak{I}_P$.

We will proceed in two steps. First, we will approximate g by using connectionist systems with step activation functions. Afterwards, we will relax our approach for the treatment of sigmoidal activation functions.

4.1 Step Activation Functions

We will now construct a multi-layer feed-forward network with weighted sum input function, where each of the units in the hidden layer computes the following step function:

$$s_{l,h,m}(x) := \begin{cases} l & \text{if } x \leq m \\ l+h & \text{otherwise.} \end{cases}$$

As an abbreviation, we will use $s_i(x) := s_{l_i,h_i,m_i}(x)$ for $0 \leq i < n-1$. We want the output to agree with g on its domain, that is, we want $\sum_{i=0}^{n-2} s_i(x) = g(x)$ for all $x \in D$.

An intuitive construction is depicted in Figure 6. For n pieces, we use $n-1$ steps. We put one step in the middle between each two neighbouring pieces, then obviously the height of that step must be the height difference between these two pieces.

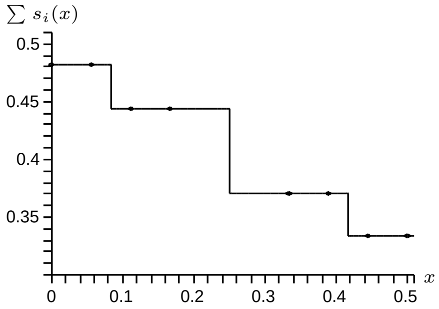


Figure 6: Sum of the step functions.

It remains to specify values for the left arms of the step functions. All left arms should add up to the height of the first piece. So we can choose that height divided by $n-1$ for each left arm. Now we have specified all s_i completely:

Definition 4.1. For $0 \leq i < n-1$,

$$l_i := \frac{y_0}{n-1}; \quad h_i := -y_i + y_{i+1}; \quad m_i := \frac{1}{2}(c_i + a_{i+1})$$

Theorem 4.2. $\sum_{i=0}^{n-2} s_i(x) = g(x)$ for all $x \in D$.

Proof. Let $x \in [a_j, c_j]$. Then

$$\begin{aligned}\sum_{i=0}^{n-2} s_i(x) &= \sum_{i=0}^{j-1}(l_i + h_i) + \sum_{i=j}^{n-2} l_i = \sum_{i=0}^{n-2} l_i + \sum_{i=0}^{j-1} h_i \\ &= y_0 + \sum_{i=0}^{j-1}(-y_i + y_{i+1}) = y_j = g(x).\end{aligned}$$

□

4.2 Sigmoidal Activation Functions

Instead of step activation functions, standard network architectures use sigmoidal activation functions, which can be considered to be approximations of step functions. The reason for this is that standard training algorithms like backpropagation require differentiable activation functions.

In order to accommodate this, we will now approximate each step function s_i by a sigmoidal function σ_i:

$$\sigma_i(x) := \sigma_{l_i,h_i,m_i,z_i}(x) := l_i + \frac{h_i}{1 + e^{-z_i(x-m_i)}}.$$

Note that l_i, h_i, m_i are the same as for the step functions. The error of the i-th sigmoidal is

$$\delta_i(x) := |\sigma_i(x) - s_i(x)|.$$

An analysis of this function leads to the following results (illustrated in Figure 7): For all $x \neq m_i$ we have $\lim_{z_i \to \infty} \sigma_i(x) = s_i(x)$; since both functions are symmetric, we find for all $z_i, \Delta x$,

$$\delta_i(m_i - \Delta x) = \delta_i(m_i + \Delta x);$$

and furthermore, for all z_i, x, x' with $|x' - m_i| > |x - m_i|$,

$$\delta_i(x') < \delta_i(x).$$

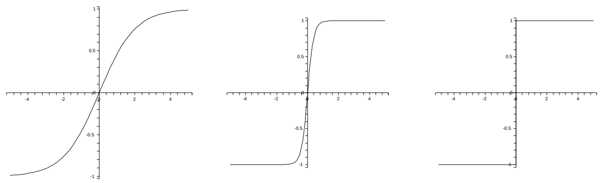


Figure 7: With increasing z, $\sigma_{l,h,m,z}$ gets arbitrarily close to $s_{l,h,m}$ everywhere but at m. The difference between $\sigma_{l,h,m,z}$ and $s_{l,h,m}$ is symmetric to m and decreases with increasing distance from m. Shown here are $\sigma_{-1,2,0,1}, \sigma_{-1,2,0,5}, s_{-1,2,0}$.

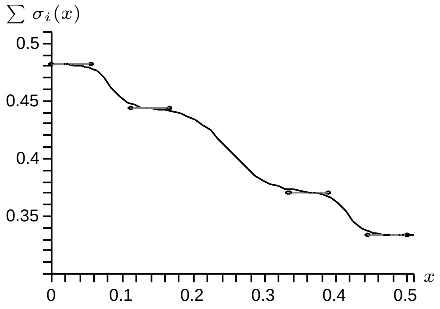


Figure 8: The sigmoidal approximation.

Theorem 4.3. For all $\epsilon > 0$ we can find z_i ($0 \leq i < n-1$) such that $\left|\sum_{i=0}^{n-2} \sigma_i(x) - g(x)\right| < \epsilon$.

Proof. In the worst case, the respective errors of the σ_i add up in the sum. Thus we allow a maximum error of $\epsilon' := \frac{\epsilon}{n-1}$ for each σ_i. With all previous results, it only remains to choose the z_i big enough to guarantee that at those $x \in D$ which are closest to m_i (i.e. c_i and a_{i+1}, which are equally close), σ_i approximates s_i up to ϵ', that is

$$\left[\delta_i(c_i) = \right] \delta_i(a_{i+1}) < \epsilon'.$$

Resolving this we get the following condition for the z_i:

$$z_i > \begin{cases} -\infty & \text{if } |h_i| \leq \epsilon' \\ -\frac{\ln \epsilon' - \ln(|h_i| - \epsilon')}{a_{i+1} - m_i} & \text{otherwise} \end{cases}$$

for $0 \leq i < n-1$. This completes the proof. □

Figure 8 shows the resulting sigmoidal approximation, along with the original piecewise constant function from Figure 6.

Taking g to be $\hat{f}_{P_l}$ and $\epsilon > 0$, the parameters l_i, h_i, m_i as in Definition 4.1 and z_i as in the proof of Theorem 4.3 determine an appropriate approximating sigmoidal network.

5 Constructing RBF Networks

Within the following section we will show how to construct *Radial Basis Function Networks* (RBF Networks). For a more detailed introduction for this type of network we refer to [14]. As in the previous section, we take a stepwise approach and will first discuss triangular activation functions.

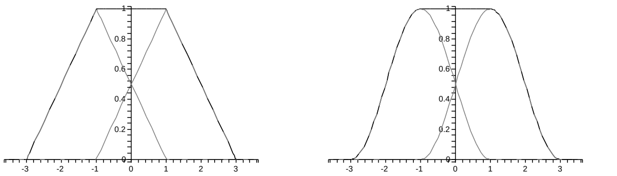


Figure 9: A constant piece can be obtained as the sum of two triangles or two raised cosine functions.

We will then extend the results to so-called raised cosine radial basis functions. We will also briefly discuss how an existing network can be refined incrementally to lower the error bound. The notation is the same as in the previous section. We will again assume that g is a piecewise constant function, this time with the additional requirement that the gaps between the pieces are $\geq$ the length of the pieces (which we proved to hold for $\hat{f}_{P_l}$), i.e. $c_i + b \leq a_{i+1}$ for $0 \leq i < n$.

5.1 Triangular Activation Functions

We will now construct an RBF network with distance input function, where each of the units in the hidden layer computes a triangular function $t_{w,h,m}$:

$$t_{w,h,m}(x) := \begin{cases} h \cdot \left(1 - \frac{|x-m|}{w}\right) & \text{if } |x-m| < w \\ 0 & \text{otherwise} \end{cases}$$

Since the triangular functions are locally receptive, that is, they are $\neq 0$ only on the open range $(m-w, m+w)$, we can handle each constant piece separately and represent it as a sum of two triangles, as illustrated in Figure 9.

For a given interval $[a_i, c_i]$ (with $c_i = a_i + b$), we define

$$t_i(x) := t_{b,y_i,a_i}(x), \qquad t'_i(x) := t_{b,y_i,c_i}(x).$$

Thus, for each constant piece we get two triangles summing up to that constant piece, i.e. for $0 \leq i < n$ and $x \in [a_i, c_i]$ we have $t_i(x) + t'_i(x) = y_i$, as illustrated in Figure 9.

The requirement we made for the gap between two constant pieces guarantees that the triangles do not interfere with those of other pieces.

Theorem 5.1. $\sum_{i=0}^{n-1} (t_i(x) + t'_i(x)) = g(x)$ for all $x \in D$.

Proof. This equality follows directly from the fact that the two triangles add up to a constant piece of the required height, and furthermore, that they do not interfere with other constant pieces as mentioned above. □

5.2 Raised-Cosine Activation Functions

As in the previous section, standard radial basis function network architectures use differentiable activation functions. For our purposes, we will replace the triangular functions t_i and t'_i by raised-cosine functions τ_i and τ'_i, respectively, of the following form:

$$\tau_{w,h,m}(x) := \begin{cases} \frac{h}{2} \cdot \left(1 + \cos\left(\frac{\pi(x-m)}{w}\right)\right) & \text{if } |x-m| < w \\ 0 & \text{otherwise.} \end{cases}$$

Again, we will use the following abbreviations:

$$\tau_i(x) := \tau_{b,y_i,a_i}(x) \qquad \tau'_i(x) := \tau_{b,y_i,c_i}(x)$$

As illustrated in Figure 9, raised cosines add up equally nice as the triangular ones, i.e. for $0 \leq i < n$ and $x \in [a_i, c_i]$ we have $\tau_i(x) + \tau'_i(x) = y_i$. Similar to Theorem 5.1, one easily obtains the following result.

Theorem 5.2. $\sum_{i=0}^{n-1} (\tau_i(x) + \tau'_i(x)) = g(x)$ for all $x \in D$.

As in the case of sigmoidal activation functions, we obtain the required network parameters by considering $\hat{f}_{P_l}$ instead of g.

5.3 Refining Networks

Our radial basis function network architecture lends itself to an incremental handling of the desired error bound. Assume we have already constructed a network approximating f_P up to a certain ϵ. We now want to increase the precision by choosing ϵ' with $\epsilon > \epsilon' > 0$, or by increasing the greatest relevant output level. Obviously we have $o_{\epsilon'} \geq o_\epsilon$ for $\epsilon > \epsilon' > 0$.

For this subsection, we have to go back to the original functions and domains from Section 3. Defining

$$\Delta P_{l_1,l_2} := \left\{ A \leftarrow \text{body} \in \mathcal{G}(P) \middle| l_1 < \|A\| \leq l_2 \right\},$$

one can easily obtain the following result.

Lemma 5.3. If $l_2 \geq l_1$, then $\hat{l}_2 \geq \hat{l}_1$, $D_{l_2} \subseteq D_{l_1}$, $P_{l_2} = P_{l_1} \cup \Delta P_{l_1,l_2}$, and $P_{l_1} \cap \Delta P_{l_1,l_2} = \emptyset$.

Thus, the constant pieces we had before may become divided into smaller pieces (if the greatest relevant input level increases) and may also be raised (if any of the new clauses applies to interpretations represented in the range of that particular piece).

Looking at the body atoms in $\Delta P_{l_1,l_2}$, we can identify the pieces which are raised, and then add units to the existing network which take care just of those pieces. Due to the local receptiveness of RBF units and the properties of D_l stated above, the new units will not disturb the results for other pieces. Especially in cases where $|\Delta P_{l_1,l_2}| \ll |P_{l_1}|$, this method may be more efficient than creating a whole new network from scratch.

We could also right away construct the network for P_l by starting with one for P_1 and refining it iteratively using $\Delta P_{1,2}, \Delta P_{2,3}, \ldots, \Delta P_{l-1,l}$, or maybe using difference programs defined in another way, e.g. by their greatest relevant input level. This may lead to more homogeneous constructions than the method used in the previous subsections.

6 Conclusions and Future Work

In this paper, we have shown how to construct connectionist systems which approximate covered first-order logic programs up to arbitrarily small errors, using some of the ideas proposed in [15]. We have thus, for a large class of logic programs, provided constructive versions of previous nonconstructive existence proofs and extended previous constructive results for propositional logic programs to the first-order case.

An obvious alternative to our approach lies in computing the (propositional) ground instances of clauses of P up to a certain level and then using existing propositional constructions as in [11]. This approach was taken e.g. in [16], resulting in networks with increasingly large input and output layers. We avoided this for three reasons. Firstly, we want to obtain differentiable, standard architecture connectionist systems suitable for established learning algorithms. Secondly, we want to stay as close as possible to the first-order semantics in order to facilitate refinement and with the hope that this will make it possible to extract a logic program from a connectionist system. Thirdly, we consider it more natural to increase the number of nodes in the hidden layer for achieving higher accuracy, rather than to enlarge the input and output layers.

In order to implement our construction on a real computer, we are facing the problem that the hardware floating point precision is very limited, so we can only represent a small number of atoms in a machine floating point number. If we do not want to resort to programming languages emulating arbitrary precision, we could try to distribute the representation of interpretations on several units, i.e. to create a connectionist system with multi-dimensional input and output. For real applications, it would also be useful to further examine the possibilities for incremental refinement as in Section 5.3.

Another problem is that the derivative of the raised-cosine function is exactly 0 outside a certain range around the peak, which is not useful for training algorithms like backpropagation. Gaussian activation functions would be more suitable, but appear to be much more difficult to handle.

We are currently implementing the transformation algorithms, and will report on corresponding experiments on a different occasion. One of our long-term goals follows the path laid out in [7; 5] for the propositional case: to use logic programs as declarative descriptions for initialising connectionist systems, which can then be trained more quickly than randomly initialised ones, and then to understand the optimised networks by reading them back into logic programs.

References

[1] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1988.

[2] Sebastian Bader, Artur S. d'Avila Garcez, and Pascal Hitzler. Computing first-order logic programs by fibring artificial neural networks. In *Proceedings of the 18th International FLAIRS Conference, Clearwater Beach, Florida, May 2005*, 2005. To appear.

[3] Sebastian Bader and Pascal Hitzler. Logic programs, iterated function systems, and recurrent radial basis function networks. *Journal of Applied Logic*, 2(3):273–300, 2004.

[4] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler. The integration of connectionism and knowledge representation and reasoning as a challenge for artificial intelligence. In L. Li and K.K. Yen, editors, *Proceedings of the Third International Conference on Information, Tokyo, Japan*, pages 22–33. International Information Institute, 2004. ISBN 4-901329-02-2.

[5] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.

[6] Artur S. d'Avila Garcez, Krysia B. Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems — Foundations and Applications*. Perspectives in Neural Computing. Springer, Berlin, 2002.

[7] Artur S. d'Avila Garcez and Gerson Zaverucha. The connectionist inductive lerarning and logic programming system. *Applied Intelligence, Special Issue on Neural networks and Structured Knowledge*, 11(1):59–77, 1999.

[8] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[9] Pascal Hitzler, Steffen Hölldobler, and Anthony K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3):245–272, 2004.

[10] Pascal Hitzler and Anthony K. Seda. Generalized metrics and uniquely determined logic programs. *Theoretical Computer Science*, 305(1–3):187–219, 2003.

[11] Steffen Hölldobler and Yvonne Kalinke. Towards a massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.

[12] Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.

[13] John W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.

[14] R. Rojas. *Neural Networks — A Systematic Introduction*. Springer, 1996.

[15] Anthony K. Seda. On the integration of connectionist and logic-based systems. In M. Schellekens T. Hurley, M. Mac an Airchinnigh and A. K. Seda, editors, *Proceedings of MFCSIT2004, Trinity College Dublin, July 2004*, Electronic Notes in Theoretical Computer Science, Elsevier, pages 1–24, 2005.

[16] Anthony K. Seda and Máire Lane. On approximation in the integration of connectionist and logic-based systems. In L. Li and K.K. Yen, editors, *Proceedings of the Third International Conference on Information, Tokyo, Japan*, pages 297–300. International Information Institute, 2004. ISBN 4-901329-02-2.

Symbolic Encoding of Neural Networks using Communicating Automata with Applications to Verification of Neural Network Based Controllers

Li Su, Howard Bowman and Brad Wyble
Centre for Cognitive Neuroscience and Cognitive Systems, University of Kent,
Canterbury, Kent, CT2 7NF, UK
{ls68,hb5,bw5}@kent.ac.uk

Abstract

This paper illustrates a way for applying formal methods techniques to specifying and verifying neural networks, with applications in the area of neural network based controllers. Formal methods have some of the characteristics of symbolic models. We describe a communicating automata [Bowman and Gomez, 2005] model of neural networks, where the standard Backpropagation (BP) algorithm [Rumelhart *et al.*, 1986] is applied. Then we undertake a verification of this model using the model checker Uppaal [Behrmann *et al.*, 2004], in order to predict the performance of the learning process. We discuss broad issues of integrating symbolic techniques with complex neural systems. We also argue that symbolic verifications may give theoretically well-founded ways to evaluate and justify neural learning systems.

1 Introduction

This work is an initial step towards integrating symbolic and neural computation. Our motivations are justified from a cognitive point of view on the one hand, and an engineering application point of view on the other hand.

1.1 Cognitive Viewpoint

[Barnard and Bowman, 2004] suggested that one of the motivations of integrating symbolic and sub-symbolic computation is to specify and justify behavior of complex cognitive architectures in an abstract and suitable form. Firstly, numerous theories assume that mental modules, or their neural substrates, are processing information at the same time. Hence, any realistic architecture of the mind must be concurrent at some level. Secondly, the control of concurrent processing can be distributed. So, cognition should be viewed as the behavior that emerges from interaction amongst independently evolving modules. Most traditional AI approaches fail to acknowledge the requirements of concurrent and distributed control. This is because they tend to be prescriptive and are built around centralized memory and control algorithms. Finally, [Fodor and Pylyshyn, 1988] argued that hierarchical decomposition is needed in order to reflect the characteristics of the mind.

Although connectionist networks are commonly regarded as concurrent and distributed, they are typically limited to only one level of concurrently evolving modules. The primitive elements of neural networks are neurons but not neural networks. To some degree, it is hard to construct and understand large architectures without hierarchical structuring. In certain respects, modeling based on neural networks is low-level in character, i.e. it is hard to relate to primitive constructs and data structures found in high-level notations preferred by the symbolic modeling community.

For these reasons, [Barnard and Bowman, 2004] provided an illustration of modeling a high-level cognitive architecture using formal methods. Their model contains a set of top-level modules that are connected by communication channels. Modules interact by exchanging data items along channels. Control is distributed and each module evolves independently. They also suggested encoding low-level neural networks using the same method, and formally relating models at different levels of abstraction. In this paper, key constructs within neural networks are encoded at two levels of description, which have characteristics of symbolic systems. The low-level descriptions use neural networks encoded in communicating automata. We argue that this formalism sits between classical forms of symbolic systems arising from programming languages such as Lisp and Prolog, and connectionist networks. We will explain these models in section 2. The high-level descriptions contain a set of properties, which are expressed in logical formulae. They are abstract descriptions of global properties, which do not prescribe internal details of how those properties are realised. We will explain these models in section 3. Computer scientists have developed a number of theories and tools to automatically justify the relationship between different levels of description within a formal framework. Our models prescribe low-level internal structure, which we hypothesis can be used to explore complex interactions within neural networks and to justify whether high-level properties can emerge from low-level constructs.

1.2 Application Viewpoint

Symbolic systems are good for manipulating, explaining and reasoning about complex data structures, but neural networks are good at dealing with complex highly non-linear systems, especially in handling catastrophic changes or gradual degradations. It is argued by [Schumann *et al.*, 2003] that neural networks can be applied to extending traditional controllers, which are ineffective in some systems, including aircrafts, spacecrafts, robotics and flexible manufacturing systems. Neural network based controllers have demonstrated a superior ability to control adaptive systems. However, the correctness of adaptive systems must be guaranteed in safety/mission critical domains. This is because it is not possible to adapt toward controllable behaviours when the system has changed beyond a critical point. This is also because the system has to dynamically react to changes within short periods of time. So, this requires that the learning processes converge before a pre-specified deadline.

Unfortunately, the slow speed of learning is one of the greatest limitations of current learning algorithms. For example, the standard BP algorithm often requires the training patterns to be presented hundreds or thousands of times in order to solve a relatively simple task. Furthermore, connectionist networks rarely provide any indication of the accuracy and reliability of their predictions. As long ago as 1988, [Fodor and Pylyshyn, 1988] pointed out that the neural networks approach remained almost entirely experimental. Although a great deal of mathematical work has been done, it is still not sufficient from the analytical point of view to justify that certain configurations of neural networks and their mechanisms are reliable.

In some applications, the control architecture uses pre-trained networks, which are numerical approximations of a function. The correctness of such systems can be verified [Rodrigues *et al.*, 2001], but their verification does not consider the adaptability of the system. Other control architectures use on-line training of neural networks. This approach is attractive because it is able to handle dynamic adaptation, but it requires a high level of stability and correctness of the learning process. There are existing approaches to evaluate the performance of neural networks, such as [Schumann *et al.*, 2003], who proposed a layered approach to verify and validate neural network based controllers. The limitation of their work is that they only focus on monitoring the on-line adaptation but cannot guarantee stability and correctness at system design stages.

This paper describes a case study, which applies formal methods techniques to evaluating the learning speed using automatic analysis (model checking). Formal methods are strongly based on logic. They have rich tool support and have shown their power in software engineering and various areas where correctness and effectiveness of computer systems need to be guaranteed. So they can, for example, be used in designing distributed systems [Bowman and Derrick, 2001]. In these areas, there are similar problems and requirements in respect of modeling complex interactions among components with distributed control.

2 Communicating Automata Specification

In this section, we introduce a communicating automata specification of neural networks, which may be used to specify components of neural network based controllers. The interested reader is referred to [Bowman and Gomez, 2005] for comprehensive definition of communicating automata. A similar framework was presented by [Smith, 1992] in a general mathematical setting. But his work did not consider automatic simulation or verification.

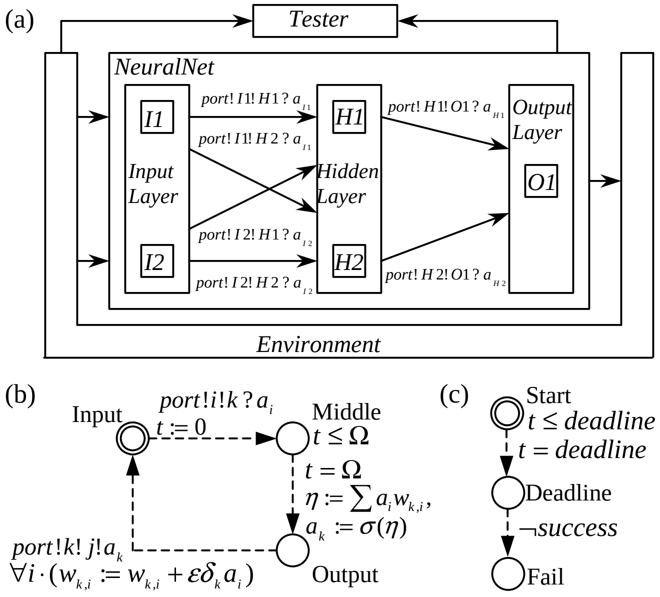


Figure 1: (a) Neural Network that Learns XOR. (b) Example of a Neuron Automaton (c) The Test Automaton, *Tester*.

2.1 Neural Network Description

Communicating automata are Finite State Machines with associated communication channels and mathematical equations. Each neuron is encoded as an automaton, denoted as the smallest square boxes in Figure 1 (a). Automata evolve concurrently and the state of each neuron only depends on the local data structure, which may change as a result of interaction and communication between automata. Activation exchange between neurons is modeled through communication channels. In Figure 1 (a), each arrow denotes a communication channel, such as the channel between neuron *I1* and *H1*:

$$port!I1!H1?a_{I1}$$

Working from left to right, *port* denotes the communication name (which in this case, is shared by all interactions), *I1* denotes the pre-synaptic neuron identity, defining the sender, and *H1* denotes the post-synaptic neuron identity, defining the receiver. The last element a_{I1} denotes the activation passed though the channel.

The system in Figure 1 (a) is described as a hierarchy of components, at the top-level it has three modules: *Environment*, *NeuralNet* and *Tester*. Each of which can be composed from a set of modules. For example, *NeuralNet* itself is composed of *InputLayer*, *HiddenLayer* and *OutputLayer*, each of which is also a module composed of a set of neurons. Neurons are fully connected between adjacent layers and BP learning is applied. The *Environment* automaton provides inputs to and receives outputs from *NeuralNet*. The *Tester* will be used in the verification in section 4.

The BP algorithm is a supervised learning rule widely used in many applications, so study of this algorithm has practical value. We have chosen the XOR problem as our learning task due to its historical position. Although it requires a small number of nodes and connections, it is characteristic of difficult linearly inseparable learning tasks. This simple problem is often used to test the ability of learning algorithms and it has been much discussed. In terms of our larger ambition, analyses of neural network based controllers, this XOR verification serves as a preliminary assessment of our approach, which will be extended to realistic applications in future work.

2.2 Neuron Automaton

We define the neuron automaton based on a set of functions, which describe the network updating dynamics. An example of a neuron automaton is shown in Figure 1 (b), where circles denote locations of the automaton, circles with a smaller circle inside denote initial locations, and arrows with dotted lines denote transitions between two locations. *k* denotes the identity of this neuron, *i* and *j* denote the presynaptic and post-synaptic neuron identities respectively. Note that neuron identities are assumed to be unique.

Briefly, the neuron automaton begins at the *Input* location. When all pre-synaptic activations have been received from input channels, it moves to the next location, *Middle*. Then it evaluates the net input η and the activation a_k. σ is a sigmoid function. At the *Output* location, it sends its activation via output channels, and weights are updated. ε is the learning rate and δ_k denotes the extent to which neuron *k* is in error. It is evaluated externally, an explanation of which is beyond the scope of this paper. For simplicity of presentation, we show a neuron automaton with just one input and one output, but a more general form can be defined.

The timing constraints in this application are the following. $t \le \Omega$ is an invariant of the *Middle* location, and *t* is a local clock. Invariants are timing conditions, and automata can only stay in locations while the condition holds. $t = \Omega$ is a guard, which is a condition allowing the transition to be taken. To summarize the time course of the neuron, it stays at the *Input* and *Output* locations while communication is completing, but it stays at the *Middle* location for exactly Ω units of time, which we assume represents the time required to update net input and activation. In this case, Ω is 5 units of time. This assumption is made only for analytical reasons and is not based on neuron physiology.

3 Requirements Language

The high level behaviour of neural networks is described using a requirements language allowing logical formulae to be expressed. The network of automata evolves through a series of states, which form several paths. The system can evolve through different paths. The requirements language consists of state formulae, which describe individual states and path formulae, which quantify over paths of the model. Assuming φ is a state formula, $\forall$, $\exists$, $\Diamond$ and $\Box$ are operators of path formulae. The property $\forall\Box\varphi$ requires that all states satisfy φ, $\exists\Diamond\varphi$ requires that at least one reachable state satisfies φ, $\exists\Box\varphi$ requires that at least along one path all states satisfy φ and $\forall\Diamond\varphi$ requires that along all paths at least one state eventually satisfies φ.

In this paper we are interested in learning. There is a set of properties, which we want the learning system to satisfy. These properties fall into three categories: Reachability, Safety and Liveness [Behrmann *et al.*, 2004].

- **Reachability Properties**
  These ask whether there eventually exists a state in which something will happen. For example, the formula *success* is *true* when all the output neurons get their activations on the correct side of 0.5. Thus, $\exists\Diamond\,\mathit{success}$ checks if the learning process could eventually allow the network to output the correct answer. These properties validate the basic behaviour of the model, but do not guarantee the correctness of adaptive systems.

- **Safety Properties**
  These ask whether something "bad" will ever happen. For example, the property *deadlock* evaluates to *true* at a state without successors. Thus, $\forall\Box\neg\mathit{deadlock}$ justifies that the system is free from such situations. Safety properties can always be expressed as reachability properties, such as $\neg\exists\Diamond\,\mathit{deadlock}$. Assuming approperate formulation of properties, neural networks, cognitive models or any dynamic systems will never reach "bad" states if they satisfy safety properties.

- **Liveness Properties**
  These ask whether the system eventually does something useful. So, we could check liveness properties such as $\forall\Diamond\mathit{success}$, which justifies that the system can meet our requirements along all paths. By verifying these properties over learning algorithms, we can justify if they will eventually converge at desired situations. In order to understand learning algorithms or adaptive systems in general, we are also interested in whether the models can perform something infinitely often.

Properties can be expressed by nesting different types of operators, such as $\forall\Diamond\forall\Box\,\mathit{success}$. This property describes a complex behaviour that we require the learning process to possess. This behaviour is that the adaptive system starts with no knowledge of the task. At this time, *success* does not hold. During training with examples, it may sometimes show correct answers, i.e. *success* holds, but it may also show incorrect answers. When this happens, the system has found some solutions to the task but these solutions are not stable during further training. However, starting from some states during the training, it is able to show correct answers invariantly. Thereby, the above property is satisfied.

4 Verification

We implemented and verified our model in Uppaal, which is a well-known real-time model checker [Behrmann *et al.*, 2004]. So, our models are converted into Uppaal timed automata notation, which is a real-time extension of communicating automata. We assume that the *deadline* is 5000 units of time, the learning rate is 0.05 and all the weights are randomly distributed around 0.2. Note that the *deadline* and Ω are both arbitrary numbers but they can be specified in real applications. We check the system for deadlock freedom using the following Temporal Logic formula:

$$\forall \Box \neg deadlock$$

The result is that the system has no deadlocks for the XOR training set. We also verify the stability of the network using the following Temporal Logic formula, which contains timing constraints:

$$\forall \Diamond_{\leq deadline} \forall \Box\, success$$

Satisfaction of this formula means the system always reaches a successful state before the *deadline*, and *success* holds invariantly from that state. However, Uppaal does not support this formula. Hence, we introduce a test automaton in Figure 1 (c). It begins at the *Start* location and moves to the next location, *Deadline*, when clock *t* equals the parameter *deadline*. When the test automaton is in this location, the patterns are still presented to the neural network. If learning is successful when *deadline* becomes *true* and remains stable during further training, the next location, *Fail*, is unreachable. With this test automaton, we are able to verify the previous property using the following Temporal Logic formula:

$$\forall \Box \neg Tester.Fail$$

The result of the verification is that the system satisfies the above property and is guaranteed to learn XOR according to the required timing constraints. It also guarantees the learning process is eventually stabilised.

5 Discussion and Future Work

In traditional neural network simulations, semantically interpretable elements are patterns of activation. The states of neural networks are expressed in numerical form, such as a landscape, in a multi-dimensional space. However, the states of the neural networks in our models are represented as the locations in the product automaton, which is automatically generated by the model checker. The locations have the characteristics of symbol systems. Model checking is based on symbolic manipulation of the product automaton, which maps to the landscape in the multi-dimensional space.

In this paper, we have specified a neural network that learns the XOR problem using communicating automata. We then verified the model over a set of properties expressed in Temporal Logic. We believe that this approach can provide insight to the field of neural network based controllers. Our models and properties can be regard as symbolic descriptions of the system behaviour at different levels of abstraction. Verifications may give theoretically well-founded ways to evaluate and justify the learning capacity and determine whether cognitive properties can emerge from neural-level architectures.

We argue that most of the properties, which we have verified, are hard to justify by simulation. This is because simulations can only test that something occurs but are unable to test that something can never occur. Simulations are also not able to test if something is (in)valid forever. Therefore, simulations are limited in their capacity to justify safety and liveness properties without explicit mathematical analysis. Our verification approach on the other hand, explicitly and formally describes the system and properties, so safety and liveness properties can be verified.

The next step of research is to explore different configurations of neural networks, e.g. winner-take-all networks, recurrent networks and more biologically plausible implementations of networks. We are also interested in investigating the potential of this approach in integrating symbolic and sub-symbolic computations, enforcing hierarchical compositional structure over neural networks, and building or justifying brain-level models.

References

[Barnard and Bowman, 2004] P. J. Barnard and H. Bowman. Rendering Information Processing Models of Cognition and Affect Computationally Explicit: Distributed Executive Control and the Deployment of Attention, *Cognitive Science Quarterly,* 3(3):297-328, 2004.

[Bowman and Derrick, 2001] H. Bowman and J. Derrick (editors). *Formal Methods for Distributed Processing: a Survey of Object-oriented Approaches*, Cambridge University Press, 2001.

[Bowman and Gomez, 2005] H. Bowman and R. Gomez. *Concurrency Theory - Calculi and Automata for Modelling Untimed and Timed Concurrent Systems*. Springer-Verlag, To Appear, 2005.

[Behrmann *et al.*, 2004] G. Behrmann, A. David and K. G. Larsen. A Tutorial on Uppaal. *SFM-RT'04, LNCS 3185*, Springer-Verlag, 2004.

[Fodor and Pylyshyn,1988] J. A. Fodor and Z. W. Pylyshyn. Connectionism and Cognitive Architecture: A Critical Analysis, *Cognition: International Journal of Cognitive Science*, Vol. 28, 3-71, 1988.

[Rumelhart *et al*., 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning Internal Representations by Error Propagation. *Paralled Distributed Processing. Explorations in the Microstructure of Cognition*. Vol.1: Foundations, 318-362. MIT Press, 1986.

[Rodrigues *et al*., 2001] P. Rodrigues, J. F. Costa and H. T. Siegelmann. Verifying Properties of Neural Networks. *Artifical and Natural Neural Networks*, *LNCS 2084*, Springer-Verlag, 158-165, 2001.

[Schumann *et al*., 2003] J. Schumann, P. Gupta and S. Nelson. On Verification & Validation of Neural Network Based Controllers, *EANN'03*, 2003.

[Smith, 1992] L. S. Smith. A Framework for Neural Net Specification, *IEEE Transactions on Software Engineering*, 18(7): 601 - 612, 1992.

Rethinking Rule Extraction from Recurrent Neural Networks

Henrik Jacobsson and **Tom Ziemke**
University of Skövde, School of Humanities and Informatics
P.O. Box 408, SE-541 28 Skövde, Sweden
{henrik.jacobsson,tom.ziemke}@his.se

Abstract

We will in this paper identify some of the central problems of current techniques for rule extraction from recurrent neural networks (RNN-RE). Then we will raise the expectations of future RNN-RE techniques considerably and through this, hopefully guide the research towards a common goal. Some preliminary results based on work in line with these goals, will also be presented.

1 Introduction

The problem of extracting rules, or finite state machines from recurrent neural networks (RNN-RE) has occupied several researchers on and off during the last 15 years. The achievements of this research has recently been compiled into a review on the subject [Jacobsson, 2005]. Unlike the field of neural networks in general, the field of RNN-RE has not actually developed much since it was first conceived. The first algorithm published [Giles *et al.*, 1992] is also still the most cited and seems to be the most commonly used algorithm although it is quite simple in its nature. More recent methods seem not to have been built on the findings of earlier approaches and there are no common benchmarks for how to compare these algorithms, nor are there any attempts to compare them in the first place. We would hold that this is due to the lack of a common goal in this field.

In this paper, we will first present some thoughts on why we at all can expect to be able to extract rules from an RNN in the first place, and why these properties should force us to rethink how connectionists should conduct and present their research. Some common constituents of existing RNN-RE techniques will be discussed briefly together with a anlysis of what is lacking in these algorithms. Some preliminary results that have been obtained by an attempt to solve these problems will also be presented briefly. Then we will try to extrapolate from the seeds of existing solutions, a set of goals that may guide research efforts to grow beyond the original intention of just extracting rules from networks.

2 The golden properties of RNNs

If we compare the study of RNNs with the study of physical dynamic systems, there are some quite obvious differences that, comparably, make RNNs perfect subjects for systematic analysis. Let us call these the “golden properties” of RNNs (properties that certainly are shared by a broad range of other systems too). Since RNNs are computer simulated systems, they allow us to (among other things):

- reproduce results with arbitrarily high accuracy,
- create more networks of the same kind without much additional effort after the framework for the creation of the first network is implemented, giving us an easy access to the population of all possible networks,
- duplicate networks (and distribute them among research colleagues),
- study the effect of damage to the network under controlled conditions,
- do nonperturbative studies of internal properties to an arbitrary degree of detail.

In other words, RNNs are almost perfect experimental subjects. Very few scientific communities have the luxury of studying entities with properties so inviting for conducting research on them.

These would be golden properties of a phenomenon for any scientist. And this puts us connectionists, who are scientifically investigating these networks on a ledge: If we are studying a phenomenon with these inviting properties, do we not also have an obligation to utilise these properties as best we can? We would say that we do. Connectionists should not be content with the limited precision associated with the limited resources associated with research on physical systems. Let us exemplify this: If an astronomer wants to verify a theory on, for example, the frequency of earth-like planets in the universe, he or she must rely on secondary or tertiary data from various sources. Gathering new data is expensive and tedious, and astronomers treat their rare data with great care because of this. Often astronomical theories have large variation because of the lack of high-quality and non-contradicting data, e.g. that the estimated age of the universe is between 10 and 20 billion years. A neural network researcher, on the other hand, should not be satisfied with secondary data, since the networks themselves can be duplicated from the original source. In case of quantitative theories, he or she should also not be satisfied to verify these theories on only a few instances of networks since there is no limit to the number of networks that can be generated.

The biggest problem is of course that the RNN researcher then “drowns” in all the data from the networks. How can one get an overview over networks when they are all individual entities with potentially quite complex behaviours?

And this is where rule extraction comes in (it comes into play under other circumstances as well, but let us stick to the analysis problem for now). Through rule extraction, the researcher can let some parts of the analysis of the individual networks be automated. If successful, the extracted deterministic rules can be enumerated, enlisted and further analysed in ways virtually impossible to do on the networks directly. And it is the golden properties that are precisely what allows us to do rule extraction from artificial neural networks at all. To do it on physical dynamic systems, e.g. a biological neuron or a star system, is by far much more difficult since it does not have these properties. But, for RNNs, we can build algorithms that utilises these properties and automate parts of the analysis process for the connectionist.

We are not there yet, however. Existing RNN-RE algorithms are still not reliable or efficient enough to be put into use in this manner. Let us have a quick look at why this is so.

3 Previous approaches and their deficiencies

What is an RNN-RE algorithm composed of? In [Jacobsson, 2005], four common ingredients were identified:

1. quantisation of the continuous state space of the RNN, resulting in a discrete set of states,
2. state and output generation (and observation) by feeding the RNN input patterns,
3. rule construction based on the observed state transitions,
4. rule set minimisation.

These four constituents are often quite distinguishable in the algorithms. For example, in [Giles *et al.*, 1992] (1) a simple grid partitioning of the state space quantised it, (2) states were generated by a breadth first search, (3) the rules were constructed by transforming the transitions in the quantised space into a deterministic finite automata and (4) the rules were minimised using a standard minimisation algorithm. Another example is [Tiňo and Vojtek, 1998] where (1) a self organising map was used to quantise, (2) states were generated by observing the network in interaction with its domain and (3) stochastic rules were induced from these observations (no minimisation in this case).

In the corpus we can find up to eight different quantisation algorithms. There are no studies that compare any of these quantisers in the domain of RNN-RE. This means, in effect, that we still do not know which of the existing quantisers should be preferred. But the lack of experimental comparisons is actually not the main problem of these approaches.

The main problem is that none of the eight tested quantisation functions have been tailor-made to comply with the specific demands of quantising the state space of a dynamic system, where the state is recursively enfolded onto itself in interaction with a domain. The eight quantisers all build (roughly) on the assumption that spatial neighbours should be merged and spatial strangers separated. But, two states that are very similar, spatially in the state space, may be very different from each other, functionally [Sharkey and Jackson, 1995].

4 Some preliminary results

When studying the earlier approaches we came to the conclusion that their main problem is the lack of integration of the four ingredients, quantisation, state generation and rule construction and minimisation. Specifically the quantiser should take into account the dynamics of the RNN through closer integration with the other constituents, so that the state space is quantised based on its functional context as set of a states of a dynamic system in interaction with a domain.

This insight was the ground for the development of a novel algorithm named `CrySSMEx` (Crystallizing Substochastic Sequential Machine Extractor) which builds on a novel quantisation algorithm (a Crystalline Vector Quantiser, CVQ) which can merge and split state quanta based on their dynamical properties in the RNN. The main outline of the algorithm is described in Figure 1. By the introduction of `CrySSMEx`, a novel form of state machine, a substochastic sequential machine (SSM) is also introduced. SSMs can, unlike earlier rules, take into account that some information may be missing in the data collected from the RNN in interaction with its domain. `CrySSMEx` is parameter free and gradually eliminates indeterminism in the generated SSMs. Unfortunately, the constituents of the algorithm are quite complex (especially the test of equivalence of states in the SSM and the selection of data for performing splits) and there is no room for these details here[1].

`CrySSMEx` has been successfully applied on various networks in different domains. [Casey, 1996] showed that it is in principle always possible to extract finite state machines from RNNs that can recognise regular languages and this is verified in that using `CrySSMEx` on RNNs adapted to regular language domains requires very few iterations in the algorithm, and poses no real challenge. It is also quite easy to extract from successful networks predicting sequences of augmented strings, in random order, from the truncated context free language $\mathbf{0}^n\mathbf{1}^n$ with $n \leq 10$ in about five to ten iterations. `CrySSMEx` has also been tested on networks with small random weights, a type of network shown to in theory be approximable by definite memory machines [Hammer and Tiňo, 2003]. Figure 2 shows a typical “what excites node X”-machine, i.e. with output symbols depending on the delta value of a specific node's activation, extracted from an RNN with one input node, 10^3 state nodes and one output node, activation function $1/(1+\exp(-net))$ with weights uniformly distributed in $[-0.1, 0.1]$ (Ω based on 10^4 random input symbols). It is also possible to extract nondeterministic stochastic machines, to a precision limited by invested computational resources, from chaotic dynamic systems, e.g. random RNNs

[1] An open source distribution and an article on `CrySSMEx` are in preparation at the time of submission of this paper. The reason we bring up `CrySSMEx` here is not to present the algorithm as such, but to present it as a promising first step towards the Empirical Machine suggested in the last section of this paper.

```
CrySSMEx($\Omega, \Lambda_i, \Lambda_o$)
Input: Time series data from the RNN, $\Omega$, an input
       quantisation function, $\Lambda_i$, and an output
       quantisation function, $\Lambda_o$.
Output: A deterministic machine $M$.
begin
    Let $M$ be the stochastic machine resulting from an
    unquantised state space (i.e. only one state);
    repeat
        Select data for splitting indeterministic states;
        Split quanta according to split data;
        Create $M$ with the new state quantiser, $\Lambda_i$ and
        $\Lambda_o$ as basis for quantisation;
        if $M$ has equivalent states then
            Merge equivalent states;
        end
    until $M$ is deterministic;
    return $M$;
end
```

Figure 1: A simplified description of the main loop of CrySSMEx. The machine M is created from the observed data contained in Ω by quantisation of input, output and state space, of which the latter is optimised in the algorithm.

with larger weights, $\mathbf{0}^n\mathbf{1}^n$-RNNs tested on strings longer than the RNN can predict correctly, or other non-RNN dynamic systems.

5 Suggested goals and ambitions

What should the ambition of our development if future RNN-RE algorithms be? It may sound as a question with one obvious answer: "to generate rules that mimic and explain instantiations of RNNs". But that answer builds on the assumption that it may not succeed, because if we thought that RNN-RE will actually succeed, to a satisfactory degree, the answer would be something like "to generalise RNN-RE algorithms so these algorithms will be applicable to as many problems as possible". We firmly believe that CrySSMEx may be one possible step towards a fully functional RNN-RE technique that may satisfy the first modest ambition. Why then, if CrySSMEx or any other technique could be shown to work satisfactory as an RNN-RE, would we be satisfied with only extracting rules from RNNs, when there are a multitude of phenomena with similar functional structures as RNNs?

There may however be no yellow brick road towards such a goal. Apart from the problem discussed earlier, there are some implicit requirements RNN-RE algorithms have on the underlying RNN. [Craven and Shavlik, 1999] suggested that a rule extraction algorithm should not even be built on the assumption that it is analysing a network at all. But exactly how generic can these algorithms become? We probably need some of the golden properties of the RNNs as research objects and there are also some implicit requirements on the underlying system too since current RNN-RE techniques (including CrySSMEx) are preferably used on RNNs that:

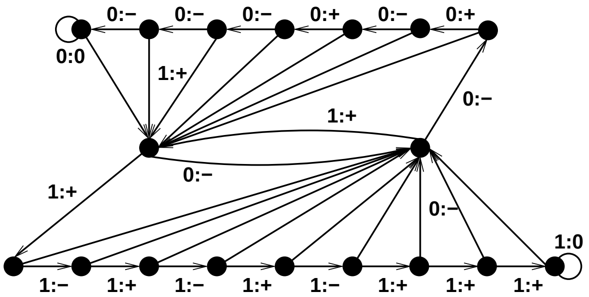


Figure 2: An extracted Mealy machine with two input symbols **0** and **1** and three output symbols **+**, **-** and **0** representing that the value of the output node increases, decreases or remains the same (due to limited machine precision), respectively.

- operate in discrete time, i.e. continuous-time RNN will not work,
- have a clearly defined input, state and output, i.e. less or randomly structured RNNs may be problematic,
- have discrete input and output,
- have a fully observable state, otherwise unobserved state nodes or noise in the observation process would disturb the extraction process since the state space would not be reliably quantised,
- have state nodes that can be set explicitly (for search-based techniques),
- are deterministic, otherwise the same problem as if the state is not fully observable would occur,
- have certain dynamical characteristics, e.g. networks with chaotic behaviour are especially difficult to do rule extraction on,
- are fixed during RE, i.e. no training can be allowed during the RE process.

Apart from the above requirements, computational complexity issues put limits on the number of state nodes and input patterns etc. These requirements are brought up here since the goals of future algorithms are naturally linked to the limitations of current algorithms in that one of the goal must be to eliminate them. If we intend to apply RNN-RE algorithms on a wider range of phenomena than just RNNs, the consequence is that our goals must be centred around creating algorithms that are as generic as possible.

Exactly what class of phenomena descendants of RNN-RE-algorithms can be used on remains an open issue, but at least we should be able to make a "wish-list" of what the algorithms should be able to do (we will still speak of RNNs, but feel free to substitute RNN by "dynamic system", or, if you are optimistic, "physical system"):

- User freedom: The user should have the freedom to select, as parameters in one and the same algorithm, to prioritise between the following four, possibly opposite goals [Jacobsson, 2005]:
    - Comprehensibility, i.e. readability and rule set size.
    - Fidelity, i.e. the degree to which the rules actually mimic the underlying RNN.

  - Accuracy, i.e. how well the rules generalise correctly on the domain of the RNN (this of course presupposes the existence of a domain in which "correctness" can be defined).
  - Efficiency, i.e. how fast the rules are generated.

- Consistency over parameters: The results should not vary with parameters that are not of the kind described above. Preferably there are no other parameters than those that are of relevance for user preferences.
- Any-time extraction [Craven and Shavlik, 1999]: The RNN-RE algorithm should provide be able to provide a quick and dirty model which is then refined to a degree (towards one of the three first goals above) proportional to the computational power invested.
- Distance measure: The RNN-RE should provide means for testing similarity of RNNs. The similarity of two RNNs may not be connected to their weights or topology in any obvious way. A distance measurement also subsumes an equivalence tester, i.e. if $d(RNN_1, RNN_2) = 0$ then RNN_1 is equivalent with RNN_2.
- Automatic subsystem identification: If a single RNN can be more efficiently described as several loosely connected subsystems, the RNN-RE should be able to discover this. This may be the case if the RNN uses separate subparts of itself to solve different subproblems in the domain.
- Queryable rules: Many times the comprehensibility of the rules overshadows all other ambitions with RNN-RE algorithms. But one can argue that a large, incomprehensible rule set, from which meaningful answers can be derived through queries, can be very useful. For example, if the network operates in a domain, it could be interesting to "ask the rules" to describe the circumstances under which the RNN makes mistakes. The answer to such queries should, for example, be helpful in designing the training conditions for the RNN.
- Automated empirical process: The RNN-RE should be able to discover when rules are not supported with enough data and then interact with the RNN such that the lacking data is acquired from the RNN. If this requirement is satisfied we propose to call the RNN-RE an *Empirical Machine* (not to be confused with the extracted machines themselves).
- Automated theory building: The RNN-RE should be able to create "theories" of one or more RNNs and strive for as universal and precise, and thereby falsifiable, theories [Popper, 1959], as possible. At this level we propose the RNN-RE should be called a *Popperian Machine*[2].

The above list is not sorted, but clearly the three last points build on each other. For the Empirical Machine to work, it has to be able to itself query the rules about how to interact with the RNN to collect missing data. The Popperian Machine may require a whole set of cooperating empirical machines generating, confirming and falsifying theories (which are statements that must in turn be translated into queries) through "experiments" on the underlying systems.

The Empirical and Popperian machines are the ultimate goals we suggest for the field of RNN-RE. If we can achieve them it would mean a great deal for the research on RNNs in general, since many portions of the research can then be automated. If we at the same time can be able to apply descendants of RNN-RE algorithms on real physical systems, that would be a significant step towards an "artificial scientist". The reason we see the Empirical and Popperian Machines as natural goals for future RNN-RE algorithms is that current techniques already contain fragments of these aspects; RNN-RE techniques, efficient or not, build detailed models (or theories) grounded in data, using a semi-empirical observational process. The preliminary promising results of `CrySSMEx`, in which the empirical process has been made explicit through a model-based selection of observed data, suggests that this may be a reasonable way towards success.

References

[Casey, 1996] M. Casey. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8(6):1135–1178, 1996.

[Craven and Shavlik, 1999] M. W. Craven and J. W. Shavlik. Rule extraction: Where do we go from here? Technical Report Machine Learning Research Group Working Paper 99-1, Department of Computer Sciences, University of Wisconsin, 1999.

[Dennett, 1996] D.C. Dennett. *Kinds of Mind*. Basic Books, New York, 1996.

[Giles *et al.*, 1992] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, and G. Z. Sun. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[Hammer and Tiňo, 2003] B. Hammer and P. Tiňo. Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, 15(8):1897–1929, 2003.

[Jacobsson, 2005] H. Jacobsson. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6):1223–1263, 2005.

[Popper, 1959] Karl R. Popper. *The Logic of Scientific Discovery*. Hutchinson Education, London, 1959.

[Sharkey and Jackson, 1995] N. E. Sharkey and S. A. Jackson. An internal report for connectionists. In R. Sun and L. A. Bookman, editors, *Computational Architectures integrating Neural and Symbolic Processes*, pages 223–244. Kluwer, Boston, 1995.

[Tiňo and Vojtek, 1998] P. Tiňo and V. Vojtek. Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences. *Neural Network World*, 8(5):517–530, 1998.

[2] Not to be confused with Dennett's Popperian Minds [Dennett, 1996].

Extracting Reduced Logic Programs from Artificial Neural Networks

Jens Lehmann[1], **Sebastian Bader**[2*], **Pascal Hitzler**[3†]
[1]Department of Computer Science, Technische Universität Dresden, Germany
[2]International Center for Computational Logic, Technische Universität Dresden, Germany
[3]AIFB, Universität Karlsruhe, Germany

Abstract

Artificial neural networks can be trained to perform excellently in many application areas. While they can learn from raw data to solve sophisticated recognition and analysis problems, the acquired knowledge remains hidden within the network architecture and is not readily accessible for analysis or further use: Trained networks are *black boxes*. Recent research efforts therefore investigate the possibility to extract symbolic knowledge from trained networks, in order to analyze, validate, and reuse the structural insights gained implicitly during the training process. In this paper, we will study how knowledge in form of propositional logic programs can be obtained in such a way that the programs are as *simple* as possible — where *simple* is being understood in some clearly defined and meaningful way.

1 Introduction and Motivation

The success of the neural networks machine learning technology for academic and industrial use is undeniable. There are countless real uses spanning over many application areas such as image analysis, speech and pattern recognition, investment analysis, engine monitoring, fault diagnosis, etc. During a training process from raw data, artificial neural networks acquire expert knowledge about the problem domain, and the ability to generalize this knowledge to similar but previously unencountered situations in a way which often surpasses the abilities of human experts.

The knowledge obtained during the training process, however, is hidden within the acquired network architecture and connection weights, and not directly accessible for analysis, reuse, or improvement, thus limiting the range of applicability of the neural networks technology. For these purposes, the knowledge would be required to be available in structured symbolic form, most preferably expressed using some logical framework.

Suitable methods for the extraction of knowledge from neural networks are therefore being sought within many ongoing research projects worldwide, see [1; 2; 8; 14; 18; 19] to mention a few recent publications. One of the prominent approaches seeks to extract knowledge in the form of logic programs, i.e. by describing the input-output behaviour of a network in terms of material implication or *rules*. More precisely, activation ranges of input and output nodes are identified with truth values for propositional variables, leading directly to the description of the input-output behaviour of the network in terms of a set of logic program rules.

This naive approach is fundamental to the rule extraction task. However, the set of rules thus obtained is usually highly redundant and turns out to be as hard to understand as the trained network itself. One of the main issues in propositional rule extraction is therefore to alter the naive approach in order to obtain a *simpler* set of rules, i.e. one which appears to be more meaningful and intelligible.

Within the context of our own broader research efforts described e.g. in [3; 4; 5; 6; 11; 12], we seek to understand rule extraction within a learning cycle of (1) initializing an untrained network with background knowledge, (2) training of the network taking background knowledge into account, (3) extraction of knowledge from the trained network, see Figure 1, as described for example in [10]. While our broader research efforts mainly concern first-order neural-symbolic integration, we consider the propositional case to be fundamental for our studies.

We were surprised, however, that the following basic question apparently had not been answered yet within the avail-

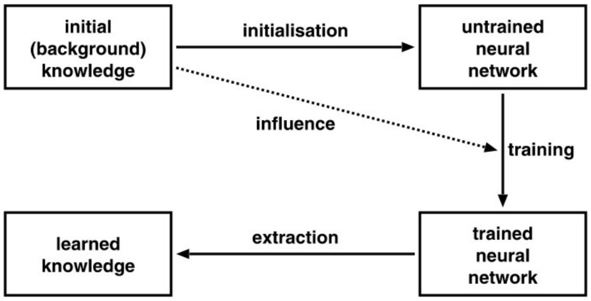


Figure 1: Neural-symbolic learning cycle

[*] Sebastian Bader is supported by the GK334 of the German Research Foundation (DFG).

[†] Pascal Hitzler is supported by the German Federal Ministry of Education and Research under the SmartWeb project, and by the European Commission under contract IST-2003-506826 SEKT.

able literature: *Using the data obtained from the naive rule extraction approach described above — when is it possible to obtain a unique irredundant representation of the extracted data?* While we believe that applicable extraction methods will have to deviate from the exact approach implicitly assumed in the question, we consider an answer important for providing a fundamental understanding of the issue. This paper is meant to settle the question to a satisfactory extent.

More precisely, we will show that a unique irredundant representation can be obtained if the use of negation within the knowledge base is forbidden, i.e. when considering *definite* logic programs — and we will also clarify formally what we mean by *redundancy* in this case. In the presence of negation, i.e. for *normal* logic programs, unique representations cannot be obtained in general, but we will investigate methods and present algorithms for removing redundancies.

The structure of the paper is as follows. After some preliminaries reviewed in Sections 2 and 3, we will present our main result on the extraction of a unique irredundant definite logic program in Section 4. How to remove redundancies in normal logic programs is discussed in Section 5, while a corresponding algorithm is presented in Section 6.

2 Logic Programs

We first introduce some standard notation for logic programs, roughly following [16]. A predicate in propositional logic is also called an *atom*. A *literal* is an atom or a negated atom. A *(Horn) clause* in propositional logic is of the form $q \leftarrow l_1, \ldots, l_n$ with $n \geq 0$, where q is an atom and all l_i with $1 \leq i \leq n$ are literals, and q is called the *head* and $l_1, \ldots, l_n$ the *body* of the clause. Clause bodies are understood to be conjunctions. If all l_i are atoms a clause is called *definite*. The number of literals in the body of a clause is called the *length* of the clause. A *(normal propositional) logic program* is a finite set of clauses, a *definite (propositional) logic program* is a finite set of definite clauses.

An *interpretation* maps predicates to $true$ or $false$. We will usually identify an interpretation with the set of predicates which it maps to $true$. An interpretation is extended to literals, clauses and programs in the usual way. A *model* of a clause C is an interpretation I which maps C to $true$ (in symbols: $I \models C$). A model of a program $\mathcal{P}$ is an interpretation which maps every clause in $\mathcal{P}$ to $true$.

Given a logic program $\mathcal{P}$, we denote the (finite) set of all atoms occurring in it by $B_{\mathcal{P}}$, and the set of all interpretations of $\mathcal{P}$ by $I_{\mathcal{P}}$; note that $I_{\mathcal{P}}$ is the powerset of the (finite) set $B_{\mathcal{P}}$ of all atoms occurring in $\mathcal{P}$.

As a neural network can be understood as a function between its input and output layer, we require a similar perspective on logic programs. This is provided by the standard notion of a semantic operator, which is used to describe the meaning of a program in terms of operator properties [16]. We will elaborate on the relation to neural networks in Section 3. The *immediate consequence operator* $T_{\mathcal{P}}$ associated with a given logic program $\mathcal{P}$ is defined as follows:

Definition 2.1. *$T_{\mathcal{P}}$ is a mapping from interpretations to interpretations defined in the following way for an interpretation I and a program $\mathcal{P}$:*

$$T_{\mathcal{P}}(I) := \{q \mid q \leftarrow \mathrm{B} \in \mathcal{P} \text{ and } I \models \mathrm{B}\}.$$

If the underlying program is definite we will call $T_{\mathcal{P}}$ *definite*. An important property of definite $T_{\mathcal{P}}$-operators is monotonicity, i.e. $I \subseteq J$ implies $T_{\mathcal{P}}(I) \subseteq T_{\mathcal{P}}(J)$. The operators $T_{\mathcal{P}}$ for a program $\mathcal{P}$ and $T_{\mathcal{Q}}$ for a program $\mathcal{Q}$ are *equal* if they are pointwise equal, i.e. if we have $T_{\mathcal{P}}(I) = T_{\mathcal{Q}}(I)$ for all interpretations I. In this case, we call the programs $\mathcal{P}$ and $\mathcal{Q}$ *equivalent*.

As mentioned in the introduction, we are interested in extracting *small* programs from networks. We will use the obvious measure of *size* of a program $\mathcal{P}$, which is defined as the sum of the number of all (not necessarily distinct) literals in all clauses in $\mathcal{P}$. A program $\mathcal{P}$ is called (strictly) *smaller* than a program $\mathcal{Q}$, if its size is (strictly) less than the size of $\mathcal{Q}$.

As already noted, the immediate consequence operator will serve as a link between programs and networks, i.e. we will be interested in logic programs *up to equivalence*. Consequently, a program will be called *minimal*, if there is no strictly smaller equivalent program.

The notion of minimality just introduced is difficult to operationalize. We thus introduce the notion of *reduced* program; the relationship between reduction and minimality will become clear later on in Corollary 4.4. Reduction is described in terms of *subsumption*, which conveys the idea of redundancy of a certain clause C_2 in presence of another clause C_1. If in a given program $\mathcal{P}$, we have that C_1 subsumes C_2, we find that the $T_{\mathcal{P}}$-operator of the program does not change after removing C_2.

Definition 2.2. *A clause $C_1 : h \leftarrow p_1, \ldots, p_a, \neg q_1, \ldots, \neg q_b$ is said to* subsume *$C_2 : h \leftarrow r_1, \ldots, r_c, \neg s_1, \ldots, \neg s_d$, iff we have $\{p_1, \ldots, p_a\} \subseteq \{r_1, \ldots, r_c\}$ and $\{q_1, \ldots, q_b\} \subseteq \{s_1, \ldots, s_d\}$.*

A program $\mathcal{P}$ is called reduced *if the following properties hold:*

1. *There are no clauses C_1 and C_2 with $C_1 \neq C_2$ in $\mathcal{P}$, such that C_1 subsumes C_2.*
2. *A predicate symbol does not appear more than once in any clause body.*
3. *No clause body contains a predicate and its negation.*

Condition 3 is actually redundant, as it is covered by condition 2. Nevertheless, we have chosen to state it seperately as this form of presentation appears to be more intuitive. Humans usually write reduced logic programs.

Using Definition 2.2, we can define a naive algorithm for reducing logic programs: Simply check every condition separately on every clause, and remove the subsumed, respectively irrelevant, symbols or clauses. Performing steps of this algorithm is called *reducing* a program. The following result is obvious.

Proposition 2.3. *If Q is a reduced version of the propositional logic program $\mathcal{P}$, then $T_{\mathcal{P}} = T_{\mathcal{Q}}$.*

3 Neural-Symbolic Integration

An *artificial neural network*, also called *connectionist system*, consists of (a finite set of) *nodes* or *units* and weighted directed connections between them. The weights are understood to be real numbers. The network updates itself in discrete time steps. At every point in time, each unit carries a real-numbered *activation*. The activation is computed based on the current input of the unit from the incoming weighted connections from the previous time step, as follows. Let $v_1, \ldots, v_n$ be the activation of the predecessor units for a unit k at time step t, and let $w_1, \ldots, w_n$ be the weights of the connections between those units and unit k, then the *input* of unit k is computed as $i_k = \sum_i w_i \cdot v_i$. The activation of the unit at time step $t + 1$ is obtained by applying a simple function to its input, e.g. a threshold or a sigmoidal function. We refer to [7] for background on artificial neural networks.

For our purposes, we consider so-called 3-layer feed forward networks with threshold activation functions, as depicted in Figure 2. The nodes in the leftmost layer are called the *input nodes* and the nodes in the rightmost layer are called the *output nodes* of the network. A network can be understood as computing the function determined by propagating some input activation to the output layer.

In order to connect the input-output behaviour of a neural network with the immediate consequence operator of a logic program, we interpret the input and output nodes to be propositional variables. Activations above a certain threshold are interpreted as *true*, others as *false*. In [12; 13], an algorithm was presented for constructing a neural network for a given $T_\mathcal{P}$-operator, thus providing the initialization step depicted in Figure 1. Without going into the details, we will give the basic principles here. For each atom in the program there is one unit in the input and output layer of the network, and for each clause there is a unit in the hidden layer. The connections between the layers are set up such that the input-output behaviour of the network matches the $T_\mathcal{P}$-operator. The basic idea is depicted in Figure 3, and an example-run of the network is shown in Figure 4. The algorithm was generalized to sigmoidal activation functions in [10], thus enabling the use of powerful learning algorithms based on backpropagation [7].

In this paper, however, we are concerned with the extraction of logic programs from neural networks. The naive, sometimes called *global* or *pedagogical* approach is to activate the input layer of the given network with all possible interpretations, and to read off the corresponding interpretations in the output layer. We thus obtain a mapping $f : I_\mathcal{P} \to I_\mathcal{P}$ as *target function* for the knowledge extraction by interpreting it as an immediate consequence operator. The task which remains is to find a logic program $\mathcal{P}$ such that $T_\mathcal{P} = f$, and furthermore, to do this in a way such that $\mathcal{P}$ is as simple as possible, i.e. minimal respectively reduced.

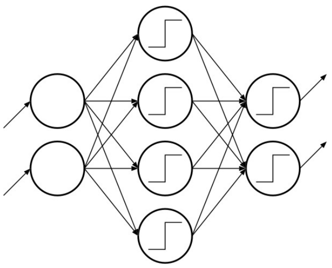


Figure 2: A simple 3-layer feed forward neural network with threshold activation function.

$$\begin{array}{rl} \mathcal{P} = \{ & p \leftarrow \neg p, \neg q; \\ & p \leftarrow p, q; \\ & q \leftarrow p, \neg q; \\ & q \leftarrow p, q \qquad \} \end{array}$$

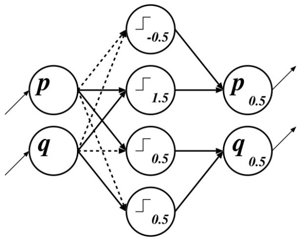


Figure 3: The 3-layer network constructed to implement the $T_\mathcal{P}$-operator of the given program $\mathcal{P}$. Connections with weight 1 are depicted solid, those with weight -1 are dashed. The numbers denote the thresholds of the units.

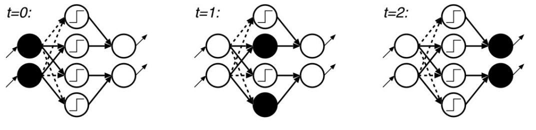


Figure 4: A run of the network depicted in Figure 3 for the interpretation $I = \{p, q\}$. A unit is depicted in black, if its activation is 1. At time $t = 0$ the corresponding units in the input layer are activated to 1. This activation is propagated to the hidden layer and results in two active units there. Finally, it reaches the output layer, i.e. $T_\mathcal{P}(I) = \{p, q\}$.

We start with naive extraction by *"Full Exploration"*, detailed in Algorithms 1 and 2, for definite respectively normal logic program extraction. We will see later that the extraction of definite programs is easier and theoretically more satisfactory. However, negation is perceived as a highly desirable feature because in general it allows to express knowledge more naturally. The target function itself does not limit the choice, so which approach will be chosen for a problem at hand will depend on the application domain. We give an example for full exploration in the normal case.

Example 1. *Let $I_\mathcal{P} = \{p, q\}$ and the mapping f be obtained by full exploration of the network shown in Figure 3. Using Algorithm 2, we obtain program $\mathcal{P}$ again, and $T_\mathcal{P} = f$ holds.*

$$\begin{array}{rrclcrl} f = \{ & \emptyset & \mapsto & \{p\} & \qquad & \mathcal{P} = \{ & p \leftarrow \neg p, \neg q; \\ & \{p\} & \mapsto & \{q\} & & & p \leftarrow p, q; \\ & \{q\} & \mapsto & \emptyset & & & q \leftarrow p, \neg q; \\ & \{p, q\} & \mapsto & \{p, q\}\} & & & q \leftarrow p, q \qquad \} \end{array}$$

Using Algorithm 2, the following result is easily obtained.

Proposition 3.1. *For every mapping $f : I_\mathcal{P} \to I_\mathcal{P}$, we can construct a propositional logic program $\mathcal{P}$ with $T_\mathcal{P} = f$.*

Note that programs obtained using Algorithms 1 or 2 are in general neither reduced nor minimal. In order to obtain simpler programs, there are basically two possibilities. On the one hand we can extract a large program using e.g. Algorithms 1 or 2 and refine it. This general idea was first described in [13], but not spelled out using an algorithm. On

Algorithm 1 Full Exploration — Definite

Let f be a mapping from $I_\mathcal{P}$ to $I_\mathcal{P}$. Initialize $\mathcal{P} = \emptyset$. For every interpretation $I = \{r_1, \dots, r_a\} \in I_\mathcal{P}$ and each element $h \in f(I)$ add a clause $h \leftarrow r_1, \dots, r_a$ to $\mathcal{P}$. Return $\mathcal{P}$ as result.

Algorithm 2 Full Exploration — Normal

Let f be a mapping from $I_\mathcal{P}$ to $I_\mathcal{P}$. Initialize $\mathcal{P} = \emptyset$. For every interpretation $I = \{r_1, \dots, r_a\} \in I_\mathcal{P}$, we have $B_\mathcal{P} \setminus I = \{s_1, \dots, s_b\}$. For each element $h \in f(I)$ add a clause $h \leftarrow r_1, \dots, r_a, \neg s_1, \dots, \neg s_b$ to $\mathcal{P}$. Return $\mathcal{P}$ as result.

Algorithm 3 Extracting a Reduced Definite Program

Let $f : I_\mathcal{P} \to I_\mathcal{P}$ be a given mapping, as obtained e.g. from a neural network, and consider $I_\mathcal{P}$ to be totally ordered in some way such that I is before K in the ordering if $|I| < |K|$. Let $\mathcal{Q}$ be an initially empty program.
For all interpretations $I \in I_\mathcal{P}$, in sequence of the assumed ordering, do the following:

- Let $I = \{p_1, \dots, p_n\}$. For every $q \in f(I)$, check whether a clause $q \leftarrow q_1, \dots, q_m$ with $\{q_1, \dots, q_m\} \subseteq I$ is already in $\mathcal{Q}$. If not, then add the clause $q \leftarrow p_1, \dots, p_n$ to $\mathcal{Q}$.

Return $\mathcal{Q}$ as the result.

the other hand, we can build a program from scratch. Both possibilities will be pursued in the sequel.

4 Extracting Reduced Definite Programs

First, we will discuss the simpler case of definite logic programs. We will derive an algorithm which returns only minimal programs, and we will also show that the notion of minimal program coincides with that of reduced program, thus serving both intuitions at the same time. Algorithm 3 satisfies our requirements, as we will see shortly.

Proposition 4.1. *Let $T_\mathcal{P}$ be a definite consequence operator and $\mathcal{Q}$ be the result of Algorithm 3, obtained for $f = T_\mathcal{P}$. Then $T_\mathcal{P} = T_\mathcal{Q}$.*

Proof. We have to show $T_\mathcal{P}(I) = T_\mathcal{Q}(I)$ for an arbitrary interpretation $I = \{p_1, \dots, p_n\}$.

For $T_\mathcal{P}(I) \subseteq T_\mathcal{Q}(I)$ we have to show that $q \in T_\mathcal{Q}(I)$ holds if $q \in T_\mathcal{P}(I)$. Assume we have a predicate q in $T_\mathcal{P}(I)$. We know that the algorithm will treat I and q (because for every interpretation I every element in $T_\mathcal{P}(I)$ is investigated). Then we have to distinguish two cases.

1. There already exists a clause $q \leftarrow q_1, \dots, q_m$ with $\{q_1, \dots, q_m\} \subseteq I$ in $\mathcal{Q}$. Then by definition $q \in T_\mathcal{Q}(I)$.
2. If there is no such clause $q \leftarrow p_1, \dots, p_n$ yet, it is added to $\mathcal{Q}$, hence we have $q \in T_\mathcal{Q}(I)$.

Conversely, we show $T_\mathcal{Q}(I) \subseteq T_\mathcal{P}(I)$. As in the other direction, we now have a predicate q in $T_\mathcal{Q}(I)$ and have to show that it is also in $T_\mathcal{P}(I)$. If $q \in T_\mathcal{Q}(I)$ we have by definition of $T_\mathcal{Q}$ a clause $q \leftarrow q_1, \dots, q_m$ with $\{q_1, \dots, q_m\} \subseteq I$. This means that the extraction algorithm must have treated the case $q \in T_\mathcal{P}(J)$ with $J = \{q_1, \dots, q_m\}$. Since $T_\mathcal{P}$ is monotonic (it is the operator of a definite program) and $J \subseteq I$ we have $T_\mathcal{P}(J) \subseteq T_\mathcal{P}(I)$, hence q is also an element of $T_\mathcal{P}(I)$. □

Proposition 4.2. *The output of Algorithm 3 is a reduced definite propositional logic program.*

Proof. Obviously the output of the algorithm is a definite program $\mathcal{Q}$, because it generates only definite clauses. We have to show that the resulting program is reduced. For a proof by contradiction we assume that $\mathcal{Q}$ is not reduced. According to Definition 2.2 there are two possible reasons for this:

Case 1: A predicate symbol appears more than once in the body of a clause. This is impossible, because the algorithm does not generate such clauses (sets do not contain elements twice).

Case 2: There are two different clauses C_1 and C_2 in $\mathcal{Q}$, such that C_1 subsumes C_2. Let C_1 be $h \leftarrow p_1, \dots, p_a$ and C_2 be $h \leftarrow q_1, \dots, q_b$ with $\{p_1, \dots, p_a\} \subseteq \{q_1, \dots, q_b\}$. As abbreviations we use $I = \{p_1, \dots, p_a\}$ and $J = \{q_1, \dots, q_b\}$. Because of case 1 we know $|I| = a$ and $|J| = b$ (all elements in the body of a clause are different). Thus we have $|I| < |J|$, because C_1 and C_2 are not equal. This means the algorithm has treated I (and $h \in f(I)$) before J (and $h \in f(J)$). C_1 was generated by treating I and h, because C_1 exists and can only be generated through I and h (otherwise the body respectively head of the clause would be different). Later the case J and h was treated. The algorithm checks for clauses $h \leftarrow r_1, \dots, r_m$ with $\{r_1, \dots, r_m\} \subseteq J$. C_1 is such a clause, because $I \subseteq J$, so C_2 is not added to $\mathcal{Q}$. Because (by the same argument as above) C_2 can only be generated through J and h, C_2 cannot be a clause in $\mathcal{Q}$, which is a contradiction and completes the proof. □

Propositions 4.1 and 4.2 have shown that the output of the extraction algorithm is in fact a reduced definite program, which has the desired operator. We proceed to show that the obtained reduced program is unique. The following, together with Corollary 4.4, is the main theoretical result in this paper.

Theorem 4.3. *For any operator $T_\mathcal{P}$ of a definite propositional logic program $\mathcal{P}$ there is exactly one reduced definite propositional logic program $\mathcal{Q}$ with $T_\mathcal{P} = T_\mathcal{Q}$.*

Proof. Assume we have an operator $T_\mathcal{P}$ of a definite program $\mathcal{P}$. With Algorithm 3 applied to $f = T_\mathcal{P}$ and Propositions 4.1 and 4.2 it follows that there is a reduced definite program $\mathcal{Q}$ with $T_\mathcal{P} = T_\mathcal{Q}$. We have to show that there cannot be more than one program with this property.

To prove this we assume (by contradiction) that we have two different reduced definite programs P_1 and P_2 with $T_\mathcal{P} = T_{P_1} = T_{P_2}$. Two programs being different means that there is at least one clause existing in one of the programs which does not exist in the other program, say a clause C_1 in P_1 which is not in P_2. C_1 is some definite clause of the form $h \leftarrow p_1, \dots, p_m$. By definition of $T_\mathcal{P}$ we have $h \in T_{P_1}(\{p_1, \dots, p_m\})$. Because T_{P_1} and T_{P_2} are equal we also have $h \in T_{P_2}(\{p_1, \dots, p_m\})$. This means that there is a clause C_2 of the form $h \leftarrow q_1, \dots, q_n$ with

$\{q_1, \dots, q_n\} \subseteq \{p_1, \dots, p_n\}$ in P_2. Applying the definition of $T_{\mathcal{P}}$ again this means that $h \in T_{P_2}(\{q_1, \dots, q_n\})$ and $h \in T_{P_1}(\{q_1, \dots, q_n\})$. Thus we know that there must be a clause C_3 of the form $h \leftarrow r_1, \dots, r_o$ with $\{r_1, \dots, r_o\} \subseteq \{q_1, \dots, q_n\}$ in P_1.

C_3 subsumes C_1, because it has the same head and $\{r_1, \dots, r_o\} \subseteq \{q_1, \dots, q_n\} \subseteq \{p_1, \dots, p_m\}$. We know that by our assumption C_1 is not equal to C_2, because C_1 is not equal to any clause in P_2. Additionally, we know that $|\{p_1, \dots, q_m\}| = m$ and $|\{q_1, \dots, q_n\}| = n$, because P_1 and P_2 are reduced, i.e. no predicate appears more than once in any clause body. So we have $\{q_1, \dots, q_n\} \subset \{p_1, \dots, p_m\}$. Because C_3 has at most as many elements in its body as C_2, we know that C_1 is not equal to C_3. That means that P_1 contains two different clauses C_1 and C_3, where C_3 subsumes C_1. This is a contradiction to P_1 being reduced. □

This shows that each algorithm extracting reduced definite programs from a neural network must return the same result as Algorithm 3. We can now also obtain that the notion of reduced program coincides with that of minimal program, which shows that Algorithm 3 also extracts the *least* program in terms of size.

Corollary 4.4. *If $\mathcal{P}$ is a reduced definite propositional logic program, then it is least in terms of size.*

Proof. Let $\mathcal{Q}$ be a program with $T_{\mathcal{Q}} = T_{\mathcal{P}}$. If $\mathcal{Q}$ is reduced, then it must be equal to $\mathcal{P}$ by Theorem 4.3, so assume it is not, i.e $\mathcal{Q}$ can be reduced. The resulting program $\mathcal{Q}_{red}$ is definite, by Definition 2.2 obviously smaller than before the reduction, and has operator $T_{\mathcal{P}} = T_{\mathcal{Q}}$. From Theorem 4.3 we know that there is only one reduced definite program with operator $T_{\mathcal{P}}$, so we have $\mathcal{P} = \mathcal{Q}_{red}$. Because $\mathcal{Q}_{red}$ is smaller than $\mathcal{Q}$, $\mathcal{P}$ is also smaller than $\mathcal{Q}$. □

5 Reducing Normal Logic Programs

As discussed in Section 3, it is possible to extract a normal logic program $\mathcal{P}$ from a neural network, such that the behaviour of the associated $T_{\mathcal{P}}$-operator and the input-output-mapping of the network are identical. But the program obtained from the naive Algorithm 2 in general yields an unwieldy program. In this section, we will show how to refine this logic program.

The first question to be asked is: Will we be able to obtain a result as strong as Theorem 4.3? The following example indicates a negative answer.

Example 2. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be defined as follows:*

$$\begin{array}{ll} \mathcal{P}_1 = \{p \leftarrow q; & \mathcal{P}_2 = \{p \leftarrow\} \\ \qquad p \leftarrow \neg q\} & \end{array}$$

Obviously, in program $\mathcal{P}_1$, p does not depend on q. Hence, the two programs are equivalent but $\mathcal{P}_2$ is smaller than $\mathcal{P}_1$. We note, however, that $\mathcal{P}_2$ cannot be obtained from $\mathcal{P}_1$ by reduction in the sense of Definition 2.2.

Example 2 shows that the notion of reduction in terms of Definition 2.2 is insufficient for normal logic programs. *Size* obviously is a meaningful notion. A naive algorithm for obtaining minimal normal programs is easily constructed: As $B_{\mathcal{P}}$ is finite, so is the set of all possible normal programs over $B_{\mathcal{P}}$ (assuming we avoid multiple occurrences of atoms in the same clause body). We can now search this set and extract from it all programs whose immediate consequence operator coincides with the target function, and subsequently we can extract all minimal programs by doing a complete search. This algorithm is obviously too naive to be practical. But it raises the question: *Is there always a unique minimal (i.e. least) program for any given target function?* The answer is negative, as the following example shows.

Example 3. *The following programs are equivalent.*

$$\begin{array}{ll} \mathcal{P}_1 = \{p \leftarrow \neg p, \neg r; & \mathcal{P}_2 = \{p \leftarrow \neg p, \neg r; \\ \qquad p \leftarrow p, r; & \qquad p \leftarrow p, r; \\ \qquad p \leftarrow \neg p, q \quad \} & \qquad p \leftarrow q, r \quad \} \end{array}$$

A full search easily reveals that the given two programs are minimal. We skip the details, which can be found in [15].

Example 3 shows that an analogy to Corollary 4.4 does not hold for normal programs. This means that we can at best hope to extract minimal normal programs from neural networks, but in general not a least program. The complexity of this task is as yet unknown, as is an optimal extraction algorithm, but we will later be able to discuss a refinement of the naive algorithm given earlier.

For the moment, we will shortly discuss possibilities for refining the set obtained by Algorithm 2. We start with two examples.

Example 4. *Let $\mathcal{P}_1$ be defined as introduced in Example 1:*

$$\begin{array}{ll} \mathcal{P}_1 = \{p \leftarrow \neg p, \neg q; & \mathcal{P}_2 = \{p \leftarrow \neg p, \neg q; \\ \qquad p \leftarrow p, q; & \qquad p \leftarrow p, q; \\ \qquad q \leftarrow p, \neg q; & \qquad q \leftarrow p \qquad \} \\ \qquad q \leftarrow p, q \quad \} & \end{array}$$

A closer look at the clauses 3 and 4 of $\mathcal{P}_1$ yields that q does not depend on q, hence we could replace both by $q \leftarrow p$. The resulting program is shown as $\mathcal{P}_2$.

Another case is given by the setting in Example 2, where a similar situation occurs. By generalizing from these examples, we arrive at the following notion.

Definition 5.1. *An α-reduced program $\mathcal{P}$ is a program with the following properties.*

1. *$\mathcal{P}$ is reduced.*
2. *There are no clauses C_1 and C_2 with $C_1 \neq C_2$ in $\mathcal{P}$, where C_1 is of the form $p \leftarrow q, r_1, \dots, r_a, \neg s_1, \dots, \neg s_b$ and C_2 is of the form $p \leftarrow \neg q, t_1, \dots, t_c, \neg u_1, \dots, \neg u_d$, where $\{r_1, \dots, r_a\} \subseteq \{t_1, \dots, t_c\}$ and $\{s_1, \dots, s_b\} \subseteq \{u_1, \dots, u_d\}$.*
3. *There are no clauses C_1 and C_2 with $C_1 \neq C_2$ in $\mathcal{P}$, where C_1 is of the form $p \leftarrow \neg q, r_1, \dots, r_a, \neg s_1, \dots, \neg s_b$ and C_2 is of the form $p \leftarrow q, t_1, \dots, t_c, \neg u_1, \dots, \neg u_d$, where $\{r_1, \dots, r_a\} \subseteq \{t_1, \dots, t_c\}$ and $\{s_1, \dots, s_b\} \subseteq \{u_1, \dots, u_d\}$.*

Both Example 2 and 4 show logic programs and their α-reduced versions. The following result and corresponding Algorithm 4 can be obtained, for details we refer to [15].

Algorithm 4 Constructing an α-reduced program

For an arbitrary propositional logic program $\mathcal{P}$ perform the following reduction steps as long as possible:

1. If there are two clauses C_1 and C_2 such that point 2 of Definition 5.1 is fulfilled, then remove $\neg q$ in the body of C_2.
2. If there are two clauses C_1 and C_2 such that point 3 of Definition 5.1 is fulfilled, then remove q in the body of C_2.
3. If there are clauses C_1 and C_2 with $C_1 \neq C_2$ in $\mathcal{P}$ and C_1 subsumes C_2, then remove C_2.
4. If a literal appears twice in the body of a clause, then remove one occurrence.
5. If a predicate and its negation appear in the body of a clause, then remove this clause.

Proposition 5.2. *Let $\mathcal{P}$ be a logic program. If $\mathcal{Q}$ is the result of Algorithm 4 on input $\mathcal{P}$, then $\mathcal{Q}$ is an α-reduced logic program and $T_{\mathcal{P}} = T_{\mathcal{Q}}$.*

Unfortunately, α-reduced programs are not necessarily minimal, as the next example shows.

Example 5. *The following two programs are equivalent. $\mathcal{P}_2$ is as in Example 3.*

$$\begin{array}{rlcrl} \mathcal{P}_2 = \{ & p \leftarrow \neg p, \neg r; & & \mathcal{P}_3 = \{ & p \leftarrow \neg p, \neg r; \\ & p \leftarrow p, r; & & & p \leftarrow p, r; \\ & p \leftarrow q, r \quad \} & & & p \leftarrow q, r; \\ & & & & p \leftarrow \neg p, q \quad \} \end{array}$$

Even though both programs are α-reduced, P_3 is larger than P_2. Note also that P_3 can be transformed to P_2 by removing a redundant clause. However, this cannot be done by α-reduction.

In a similar manner, we can refine α-reduction by introducing further refinement conditions. Refinement conditions can for example be obtained by recurring to insights from inverse resolution operators as used in Inductive Logic Programming [17]. Such a line of action was spelled out in [15]. The resulting algorithms did yield further refined programs at the cost of lower efficiency, but no satisfactory algorithms for obtaining minimal programs.

6 A Greedy Extraction Algorithm

We present another extraction algorithm for normal programs, which is closer in spirit to Algorithm 3 in that it incrementally builds a program. For this purpose we introduce the notion of *allowed clause bodies*, where the idea is that we do not want to allow clauses which clearly lead to an incorrect $T_{\mathcal{P}}$ operator, and we do not want to allow clauses, for which a shorter allowed clause exists.

The following example illustrates the intuition.

Example 6. *We will use the operator of the programs given in Example 3:*

$$\begin{array}{rrclrcl} T_{\mathcal{P}} = \{ & \emptyset & \mapsto & \{p\} & \{p,q\} & \mapsto & \emptyset \\ & \{p\} & \mapsto & \emptyset & \{p,r\} & \mapsto & \{p\} \\ & \{q\} & \mapsto & \{p\} & \{q,r\} & \mapsto & \{p\} \\ & \{r\} & \mapsto & \emptyset & \{p,q,r\} & \mapsto & \{p\}\} \end{array}$$

The 3 atoms p, q, r are being used, so there would be 27 different possible clauses bodies, as shown in Table 1. The clause $p \leftarrow p$, for example, is not correct, since we have $p \notin T_{\mathcal{P}}(\{p\})$. Hence the body p is not allowed.

We will give a formal definition of allowed clauses, before continuing with the example. Please note that in the following definition B is not necessarily a clause in $\mathcal{P}$.

Definition 6.1. *Let $T_{\mathcal{P}}$ be an immediate consequence operator, and h be a predicate. We call $B = p_1, \ldots, p_a, \neg q_1, \ldots, \neg q_b$* allowed with respect to h and $T_{\mathcal{P}}$ *if the following two properties hold:*

- *For every interpretation $I \subseteq B_{\mathcal{P}}$ with $I \models B$ we have $h \in T_{\mathcal{P}}(I)$.*
- *There is no allowed body $B' = r_1, \ldots, r_c, \neg t_1, \ldots, \neg t_d$ for h and $T_{\mathcal{P}}$ with $B' \neq B$ such that $\{r_1, \ldots, r_c\} \subseteq \{p_1, \ldots, p_a\}$ and $\{t_1, \ldots, t_d\} \subseteq \{q_1, \ldots, q_b\}$.*

As given in Definition 6.1, there are two reason for a clause body B not to be allowed. First, the resulting clause could be wrong, as discussed in Example 6. Secondly, there could be a smaller allowed body B', such that $h \leftarrow B'$ subsumes $h \leftarrow B$.

Example 6 (continued). *Table 1 shows all possible clause bodies for $B_{\mathcal{P}} = \{p, q, r\}$ on the left side. The right side shows either "OK", if the body is allowed, or gives the reason why it is not allowed.*

We use the notion of allowed clause bodies to present a greedy algorithm that constructs a logic program for a given target function. The algorithm will incrementally add clauses to an initially empty program. The clause to add is chosen from the set of allowed clauses with respect to some *score*-function, which is a heuristics for the importance of a clause. This function computes the number of interpretations for which the program does not yet behave correctly, but for which it would after adding the clause.

Definition 6.2. *Let $B_{\mathcal{P}}$ be a set of predicates. The* score *of a clause $C : h \leftarrow \mathrm{B}$ with respect to a program $\mathcal{P}$ is defined as*

$$\mathrm{score}(C, \mathcal{P}) := \big|\{I \mid I \subseteq B_{\mathcal{P}} \textit{ and } h \notin T_{\mathcal{P}}(I) \textit{ and } I \models \mathrm{B}\}\big|.$$

To keep things simple, we will consider one predicate at a time only, since after treating every predicate symbol, we can put the resulting sub-programs together. Let $q \in B_{\mathcal{P}}$ be an atom, then we call $T^q_{\mathcal{P}}$ the *restricted* consequence operator for q and set $T^q_{\mathcal{P}}(I) = \{q\}$ if $q \in T_{\mathcal{P}}(I)$, and $T^q_{\mathcal{P}}(I) = \emptyset$ otherwise. Algorithm 5 gives the details of the resulting procedure and is illustrated in Example 7.

clause body	evaluation
empty	False, $p \notin T_\mathcal{P}(\{p\})$.
p	False, because $p \notin T_\mathcal{P}(\{p\})$.
q	False, because $p \notin T_\mathcal{P}(\{p, q\})$.
r	False, because $p \notin T_\mathcal{P}(\{r\})$.
$\neg p$	False, because $p \notin T_\mathcal{P}(\{r\})$.
$\neg q$	False, because $p \notin T_\mathcal{P}(\{p\})$.
$\neg r$	False, because $p \notin T_\mathcal{P}(\{p\})$.
p, q	False, because $p \notin T_\mathcal{P}(\{p, q\})$.
p, r	OK.
q, r	OK.
$p, \neg q$	False, because $p \notin T_\mathcal{P}(\{p\})$.
$p, \neg r$	False, because $p \notin T_\mathcal{P}(\{p\})$.
$q, \neg p$	OK.
$q, \neg r$	False, because $p \notin T_\mathcal{P}(\{p, q\})$.
$r, \neg p$	False, because $p \notin T_\mathcal{P}(\{r\})$.
$r, \neg q$	False, because $p \notin T_\mathcal{P}(\{r\})$.
$\neg p, \neg q$	False, because $p \notin T_\mathcal{P}\{r\})$.
$\neg p, \neg r$	OK.
$\neg q, \neg r$	False, because $p \notin T_\mathcal{P}(\{p\})$.
p, q, r	Not considered, because p, r is smaller.
$p, q, \neg r$	False, because $p \notin T_\mathcal{P}(\{p, q\})$.
$p, \neg q, r$	Not considered, because p, r is smaller.
$\neg p, q, r$	Not considered, because q, r is smaller.
$p, \neg q, \neg r$	False, because $p \notin T_\mathcal{P}(\{p\})$.
$\neg p, q, \neg r$	Not considered, because $\neg p, q$ is smaller.
$\neg p, \neg q, r$	False, because $p \notin T_\mathcal{P}(\{r\})$.
$\neg p, \neg q, \neg r$	Not considered, because $\neg p, \neg r$ is smaller.

Table 1: Allowed clause bodies for $T_\mathcal{P}$ from Example 6.

Example 7. *Let $T_\mathcal{P}$ be given as follows:*

$$
T_\mathcal{P} = \{ \quad
\begin{array}{rlrl}
\emptyset & \mapsto \{p\} & \{q, r\} & \mapsto \emptyset \\
\{p\} & \mapsto \emptyset & \{q, s\} & \mapsto \{p\} \\
\{q\} & \mapsto \{p\} & \{r, s\} & \mapsto \emptyset \\
\{r\} & \mapsto \emptyset & \{p, q, r\} & \mapsto \{p\} \\
\{r\} & \mapsto \emptyset & \{p, q, s\} & \mapsto \emptyset \\
\{p, q\} & \mapsto \{p\} & \{p, r, s\} & \mapsto \{p\} \\
\{p, r\} & \mapsto \{p\} & \{q, r, s\} & \mapsto \{p\} \\
\{p, s\} & \mapsto \{p\} & \{p, q, r, s\} & \mapsto \{p\}\}
\end{array}
$$

Obviously, we can concentrate on the predicate p, since there are no other predicates occurring as a consequence. The resulting set of allowed clause bodies is

$$
\begin{aligned}
S = \{&p, r; \neg p, \neg r, \neg s; q, \neg p, \neg r; q, \neg r, \neg s; \\
&p, q, \neg s; p, s, \neg q; q, s, \neg p; q, r, s\}
\end{aligned}
$$

Tables 2 and 3 show two example runs of the algorithm. In each step the score for the allowed clauses which are not yet in the program, is indicated. (The score of the clause which is added to the constructed program $\mathcal{Q}$ is given in boldface.) As an example the score for $p, q, \neg s$ in the first step of the first run is 2, because $p \in T_\mathcal{P}(p, q)$ and $p \in T_\mathcal{P}(p, q, r)$. It goes down to 1 in the second step, because we have $\mathcal{Q} = \{p, r\}$ and therefore $p \in T_\mathcal{Q}(p, q, r)$ at this point. Intuitively this means that we would only gain one additional interpretation by adding $p \leftarrow p, q, \neg s$.

Algorithm 5 Greedy Extraction Algorithm

```
Let $T_\mathcal{P}$ and $B_P = \{q_1, \ldots, q_m\}$ be the input of the algorithm.
Initialize $\mathcal{Q} = \emptyset$.
For each predicate $q_i \in B_\mathcal{P}$:
  1. construct the set $S_i$ of allowed clause bodies for $q_i$
  2. initialize: $\mathcal{Q}_i = \emptyset$
  3. repeat until $T_{\mathcal{Q}_i} = T_\mathcal{P}^{q_i}$:
     (a) Determine a clause $C$ of the form $h \leftarrow B$ with $B \in S_i$ with the highest score with respect to $\mathcal{Q}_i$.
     (b) If several clauses have the highest score, then choose one with the smallest number of literals.
     (c) $\mathcal{Q}_i = \mathcal{Q}_i \cup \{C\}$
  4. $\mathcal{Q} = \mathcal{Q} \cup \mathcal{Q}_i$
Return $\mathcal{Q}$ as the result.
```

clause body	1.	2.	3.	4.	5.	6.
p, r	$\boxed{\mathbf{4}}$					
$\neg p, \neg r, \neg s$	2	2	$\boxed{\mathbf{1}}$			
$q, \neg p, \neg r$	2	$\boxed{\mathbf{2}}$				
$q, \neg r, \neg s$	2	2	1	$\boxed{\mathbf{1}}$		
$p, q, \neg s$	2	1	1	1	0	0
$p, s, \neg q$	2	1	1	1	$\boxed{\mathbf{1}}$	
$q, s, \neg p$	2	2	1	1	1	$\boxed{\mathbf{1}}$
q, r, s	2	1	1	1	1	1

$$
\begin{aligned}
\mathcal{P}_1 = \{&p \leftarrow p, r; \\
&p \leftarrow q, \neg p, \neg r; \\
&p \leftarrow \neg p, \neg r, \neg s; \\
&p \leftarrow q, \neg r, \neg s; \\
&p \leftarrow p, s, \neg q; \\
&p \leftarrow q, s, \neg p \qquad \}
\end{aligned}
$$

Table 2: Example run 1 and the resulting program.

Example 7 is constructed in such a way that there are two different possible runs of the algorithm, which return programs of different size for the same operator. The first run produces a program with six clauses and 17 literals. The second run produces a program with five clauses and 14 literals. This shows that the algorithm does not always return a minimal program, which was expected as the algorithm is greedy, i.e. it chooses the clause with respect to some heuristics and without forecasting the effects of this decision. We also see that the algorithm is not deterministic, because there may be several clauses with the highest score and the lowest number of literals (e.g. in step 3 of run 1). As for performance, the use of allowed clause bodies in this case made it possible to reduce checking from 27 to 4 clauses.

Let us finally mention how to modify Algorithm 5 in order

clause body	1.	2.	3.	4.	5.
p, r	$\boxed{\mathbf{4}}$				
$\neg p, \neg r, \neg s$	2	$\boxed{\mathbf{2}}$			
$q, \neg p, \neg r$	2	2	1	0	0
$q, \neg r, \neg s$	2	2	1	$\boxed{\mathbf{1}}$	
$p, q, \neg s$	2	1	1	1	0
$p, s, \neg q$	2	1	1	1	$\boxed{\mathbf{1}}$
$q, s, \neg p$	2	2	$\boxed{\mathbf{2}}$		
q, r, s	2	1	1	0	0

$$
\begin{aligned}
\mathcal{P}_2 = \{&p \leftarrow p, r; \\
&p \leftarrow \neg p, \neg r, \neg s; \\
&p \leftarrow q, s, \neg p; \\
&p \leftarrow q, \neg r, \neg s; \\
&p \leftarrow p, s, \neg q \qquad \}
\end{aligned}
$$

Table 3: Example run 2 and the resulting program.

Algorithm 6 Intelligent Program Search

```
Let $T_{\mathcal{P}}$ and $B_{\mathcal{P}} = \{q_1, \dots, q_m\}$ be the input of the algorithm.
Initialize: $Q = \emptyset$.
For each predicate $q_i \in B_{\mathcal{P}}$:
    1. construct the set $S_i$ of allowed clause bodies for $q_i$
    2. initialize: $n_i = 0$
    3. Search all programs with size equal to $n_i$ until a program
       $\mathcal{Q}_i$ with $T_{\mathcal{P}}^{q_i} = T_{\mathcal{Q}_i}$ is found. if no such program is found
       then increment $n_i$ and repeat step 3.
    4. $\mathcal{Q} = \mathcal{Q} \cup \mathcal{Q}_i$
```

to obtain minimal programs. We do this by performing a full program search instead of using a heuristics, i.e. the score function, to add clauses to subprograms. See Algorithm 6.

7 Conclusions

We presented algorithms to extract definite and normal propositional logic programs from neural networks. For the case of definite programs, we have shown that our algorithm is optimal in the sense that it yields the minimal program with the desired operator; and it was formally shown that such a minimal program always exists. For normal logic programs we presented algorithms for obtaining minimal programs, and more efficient algorithms which do produce small but not necessarily minimal programs.

The main contribution of this paper is the automatic refinement of logic programs, obtained by global extraction methods as in [9; 13]. We have thus addressed and answered fundamental (and obvious) open questions. We consider the results as a base for investigating the extraction of first-order logic programs, and thus for the development of the neural-symbolic learning cycle as laid out in Figure 1, which has high potential for impact in application areas.

References

[1] R. Alexandre, J. Diederich, and A. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, pages 373–389, 1995.

[2] R. Andrews and S. Geva. Rule extraction from local cluster neural nets. *Neurocomputing*, 47(1–4):1–20, 2002.

[3] S. Bader and P. Hitzler. Logic programs, iterated function systems, and recurrent radial basis function networks. *Journal of Applied Logic*, 2(3):273–300, 2004.

[4] S. Bader, P. Hitzler, and A. S. d'Avila Garcez. Computing first-order logic programs by fibring artificial neural network. In *Proceedings of the 18th International FLAIRS Conference, Clearwater Beach, Florida, May 2005*, 2005. To appear.

[5] S. Bader, P. Hitzler, and S. Hölldobler. The integration of connectionism and first-order knowledge representation and reasoning as a challenge for artificial intelligence. In L. Li and K.K. Yen, editors, *Proceedings of the Third International Conference on Information*, pages 22–33, Tokyo, Japan, November/December 2004. International Information Institute.

[6] S. Bader, P. Hitzler, and A. Witzel. Integrating first-order logic programs and connectionist systems — a constructive approach. In *Proceedings of the IJCAI-05 Workshop on Neural-Symbolic Learning and Reasoning, NeSy'05, Edinburgh, UK*, 2005. To appear.

[7] Ch. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[8] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 126(1–2):155–207, 2001.

[9] A. S. d'Avila Garcez, K. B. Broda, and D. M. Gabbay. *Neural-Symbolic Learning Systems — Foundations and Applications*. Perspectives in Neural Computing. Springer, Berlin, 2002.

[10] A. S. d'Avila Garcez and G. Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence, Special Issue on Neural networks and Structured Knowledge*, 11(1):59–77, 1999.

[11] P. Hitzler, S. Bader, and A. Garcez. Ontology learning as a use-case for neural-symbolic intergration. In *Proceedings of the IJCAI-05 Workshop on Neural-Symbolic Learning and Reasoning, NeSy'05, Edinburgh, UK*, 2005. To appear.

[12] P. Hitzler, S. Hölldobler, and A. K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3):245–272, 2004.

[13] S. Hölldobler and Y. Kalinke. Towards a new massively parallel computational model for logic programming. In *Proceedings of the ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.

[14] F. J. Kurfess. Neural networks and structured knowledge: Rule extraction and applications. *Applied Intelligence*, 12(1–2):7–13, 2000.

[15] J. Lehmann. Extracting logic programs from artificial neural networks. Belegarbeit, Fakultät Informatik, Technische Universität Dresden, February 2005.

[16] J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.

[17] S.H. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[18] J.-F. Remm and F. Alexandre. Knowledge extraction using artificial neural networks: application to radar target identification. *Signal Processing*, 82(1):117–120, 2002.

[19] A. B. Tickle, F. Maire, G. Bologna, R. Andrews, and J. Diederich. Lessons from past, current issues, and future research directions in extracting the knowledge embedded in artificial neural networks. *Hybrid Neural Systems*, pages 226–239, 1998.