in: Proc. of the International Conference on Artificial Neural Networks (ICANN-91)

OPTIMIZING LOGICAL PROOFS WITH CONNECTIONIST NETWORKS

A. Ultsch, R.Hannuschka, U.Hartmann, M. Mandischer, V.Weber

Department of Computer Science, University of Dortmund
D-4600 Dortmund 50, PO Box 500 500, Germany
e-mail: ultsch@exunido.uucp

We describe a connectionist approach to extract control knowledge from Prolog programs in order to utilize it for new proofs. A metainterpreter encodes successful Prolog proofs as training patterns for neural networks. Trained with these examples, the different connectionist networks store control strategies to select clauses and literals. A second metainterpreter uses the networks to improve the learned as well as new proofs. In this paper we examine the performance of three different types of connectionist networks for this task. The networks showed an interesting trade-off between the exactness of reproduction of a learned proof strategy and the ability to generalize suitably to new proofs.

1. INTRODUCTION

This paper presents the application of connectionist networks to the control of symbolic proofs. By symbolic proofs we understand proofs in first order logic. Prolog interpreters are an implementation of theorem provers for special first order formulas, called horn clauses, based on the resolution principle [11]. Prolog interpreters use a fixed control strategy for symbolic proofs: first, selection of the leftmost partial goal in the resolvent as goal to be solved and second, selection of the clauses in written order for the resolution of a selected literal. In case of failure, the interpreter backtracks to the last choice made without analyzing the cause of failure. Even for simple programs, this implicit control strategy may be not sufficient to obtain efficient computations [10]. Explicit control by using control constructs like ´cut´ is, however, contradictory to the use of Prolog as a declarative programming language. The need for the distinction of logic and control has therefore long been recognized [8, 15].

In our approach we used networks for the control of both, literal, and clause selection. This paper especially focuses on the control of the literal selection. A detailed description of the clause selection is given in [4]. Connectionist models are able to store information in a distributed and compressed way. For a task like this, were a input and a desired goal exists, it seems to be useful to utilize supervised learning models, such as Backpropagation nets, which were therefore also used by the system described in [16]. In addition to this, our aim was to examine how suitably modified unsupervised networks, such as Kohonen‘s Feature Map and ART1 performed for a typical supervised task.

2. CONTROL KNOWLEDGE

Fig. 1a describes a typical situation in writing Prolog programs. A Prolog program can be thought of as consisting of a pure logic program and an encoding of control knowledge (proof strategy). I.e. an encoding of the sequence of successive clause/literal selections.

As we want to improve the proof strategy we need the explicit expression of control knowledge. This is achieved using a certain syntax, called “&-Prolog“, to describe parts of the program where the sequence of literals is not important. It is, however, possible to specify the usage of Prolog's proof strategy for other parts of the program. We propose not to formulate control knowledge explicitly but rather to extract it from the observation of proofs and get an abstract description of specific proof situations (see fig. 1b). The abstract description of control knowledge can be used to solve similar proofs.

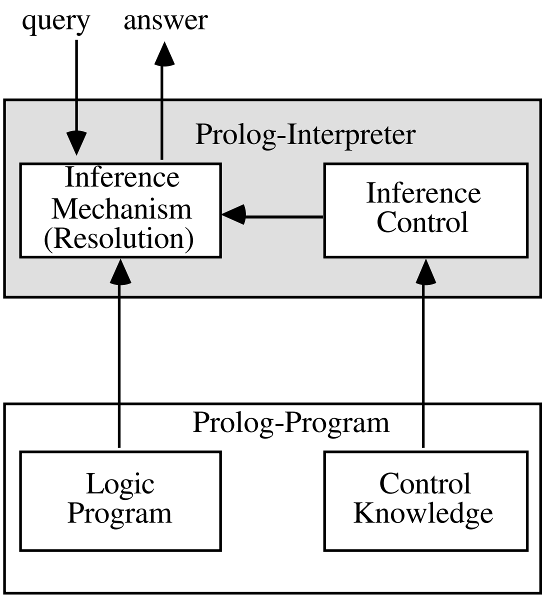

Fig. 1a: Relation of logic and control in Prolog

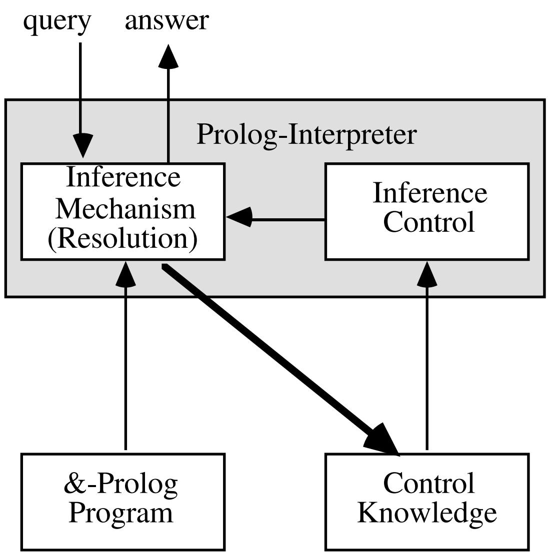

Fig. 1b: Relation of logic and control in CRECK

Basically, control knowledge is formulated as a pair of descriptions: one concerning the selection situation (description of goal) and a second representing the chosen selection. In case of clause selection this means the description of the goal to be solved and the description of the chosen clause. For literal selection we describe the structure of the resolvent and the ordering of the literals. Each of these descriptions consists of the types of the terms each argument is instantiated to. In case of composed terms, there is also a description of each subterm. Furthermore the identity of the terms and/or subterms at each argument position is determined. The examination of the instantiations and identities provides an implicit analysis of the arguments with respect to inclusions of terms. Table1 gives an example of a goal and shows the corresponding description of its structure.

Goal: `tree_del`	`(tree(`	`void,`	`3,`	`void),`	`3,`	`X).`
Term	`tree`	`void`	`3`	`void`	`3`	`X`
Type-description:	n-ary op.	atom	integer	atom	integer	variable
Position	1	2	3	4	5	6

Table 1. example for a structure description

In the example before the terms on position (2,4) and (3,5) are identical terms. This description includes also the information that the second argument of tree_del is a subterm of the first argument. This way of describing control knowledge is designed for possible use in new proofs. The description is not restricted to a given proof, since the control knowledge forms an abstraction of the executed proof. The system can take advantage of the ability of connectionist networks to generalize, when the knowledge is transfered to new proofs. The connectionist networks store this control knowledge in a compressed and distributed way in their weights.

3. THE GENERATION AND USAGE OF CONTROL KNOWLEDGE

In the following section we describe a system for generating and using control knowledge for proofs in Prolog. To derive and use control knowledge we need a system which gives us access to the inference process. This is possible using a metainterpreter which allows us to change the proof strategy.

For practical purposes we distinguish two different phases. During the training phase Prolog proofs are observed, and control knowledge is derived. Successful proofs are encoded and the networks are trained with these descriptions. After training, the networks are used to optimize new proofs in a second phase. Fig. 3 provides an overview of the system.

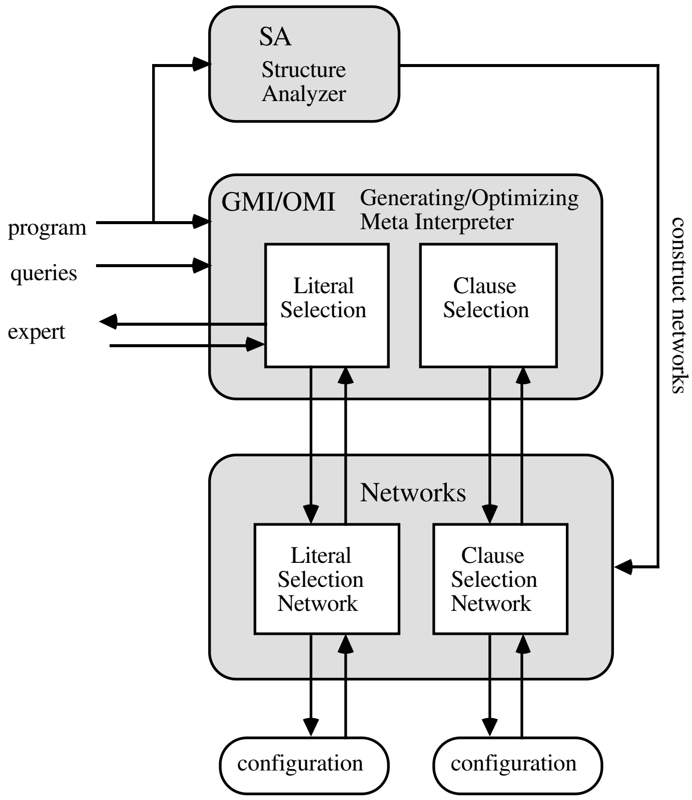

Fig. 2. System for generating and using connectionist control knowledge

To generate control knowledge, we use a Prolog program and a set of queries. The program is examined by a structure analyzer (SA) to determine the size of the networks. A generating metainterpreter (GMI) observes the proof of each query. The sequences of literals to be proved are specified by a certain heuristic or a human expert. Each occurring selection situation is described. The abstract description of each partial goal and the chosen selection is derived. The networks are trained with the encoded descriptions of goals and selections.

In the working phase the control knowledge represented in the trained connectionist networks is used by an optimizing metainterpreter (OMI). A program and a query are input to the metainterpreter. Each literal and/or clause selection situation appearing in the proof of a query is encoded and passed to the networks. The corresponding network determines the next selection to be made.

4. CODING AND USED NETWORKS

A simple binary coding of the control knowledge for input and output is not appropriate, because similarities between structures can not be encoded this way (see [14]). To represent the similarities between types in the structure description we utilize a distributed representation by microfeatures [5, 9]. For coding identities and inclusions of arguments we use a triangle matrix as shown below.

■ is type
□ is not type

Microfeatures \ Types	argument	nonvar	constant	compound	list	nlst compoun	arity 1	arity 2
nul								
variable	■							
integer	■	■	■				■	
atom	■	■	■					■
nil	■	■		■	■			■
list	■	■		■	■		■	
operator 1	■	■		■		■	■	
operator 2	■	■		■		■		■
operator n	■	■		■		■	■	■

Fig. 3a. Microfeatures

■ identity
□ non identity

	argument 1	argument 2	argument 3	argument 4	argument 5	argument 6
argument 1		□	□	□	□	□
argument 2			□	■	□	□
argument 3				□	■	□
argument 4					□	□
argument 5						□
argument 6						

Fig. 3b. Triangle matrix for the example in table 1

We experimented with three different types of connectionist networks to learn control knowle-dge: a Backpropagation network described by [12], a modified ART1-model [1], and a modified Selforganizing Feature Map [7].

Fig. 4 shows the architecture of the Backpropagation network. We use separated hidden layers of 150 to 300 units each to cope with the different control knowledge features like instantiations and identities of arguments. The sizes of the in- and output layer are determined by the size of the Prolog program under examination. For our examples, we found a typical size of 300 to 600 units in the input layer and up to 20 units in the output layer.

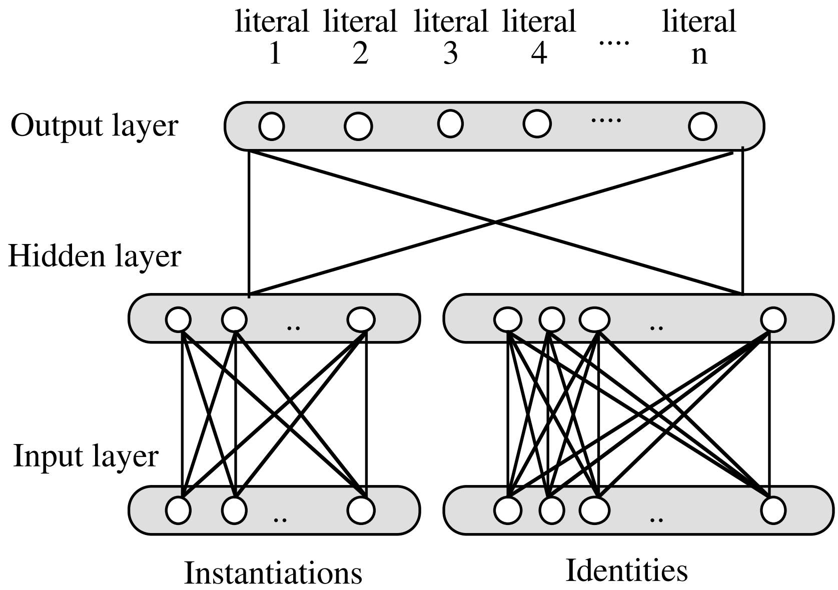

Fig. 4. Backpropagation architecture for literal selection

The ART1-Model provides an unsupervised learning procedure [1]. Similar input patterns are represented by one output unit in this model. The ART1-Model uses a vigilance-parameter to determine the maximum difference between patterns represented by the same output unit. In our approach an ART1-Network is modified to perform supervised learning. The main ideas to achieve this aim are as follows:

- Connecting the output layer with the representation of features of a program by a link
- Handling symbolic mismatch by increasing the vigilance parameter of the ART1-Model
- Using the bottom-up weights of the network as a Competitive Learning Network if the ART1-Network does not compute a convenient output pattern

Kohonen's Selforganizing Feature Maps are used mostly in the context of unsupervised learning [7]. In order to modify the map for supervised learning we changed the input to the net as follows:A training vector consists of two parts:a description of the proof selection situation and the selection that was effective for that situation. A proof selection situation is described by a binary vector of features. The selection is encoded as a sequence of integer numbers indicating the qualification of literals. To determine a selection for a selection situation, only the description of the selection situation is considered for matching. The vector is completed using the weights of the matched unit. We use a grid with 30x30 units, which are trained for 350 epoches.

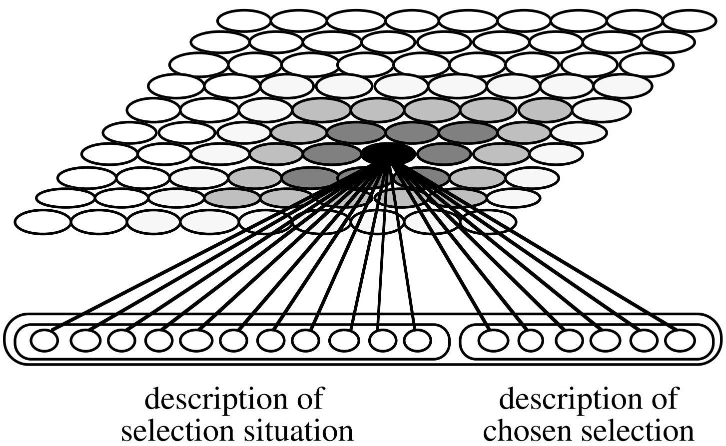

Fig. 5: Kohonen‘s self organizing feature maps

5. RESULTS

Using different networks to learn control knowledge we found that self-organizing feature maps (Kohonen maps) perform this task best. A surprising result is that this network and also the appropriately modified unsupervised network for ART1 can perform typical supervised learning tasks better than the backpropagation algorithm.
The following results are derived by example programs performing map-coloring problems. Networks have been trained with proofs of 500 randomly generated queries.

The number of resolutions for proofs with Prologs, the networks, and an optimal strategy have been counted for the following sets of queries:

- 500 queries used to generate training patterns for learning
- 500 additional queries generated by random
- 500 queries for a new, but similar program

In our approach we trained a backpropagation network with 150,000 epoches. Using back-propagation the performance is hardly improved and the ability of the network to generalize cannot be used advantageously for the proof of unknown queries. The trained ART1 network is able to reproduce the control knowledge for each training query. The cause of this behavior is that patterns representing control knowledge for a proof are stored separately. The network is also able to generalize to unknown proofs. Results for unknown queries are better than Prolog's but worse than those achieved by the feature maps. As the results obtained using self-organizing feature maps in fig. 6 show, these maps prove unobserved queries nearly as good as trained ones. Even for a program with a similar structure which was not used to generate control knowledge, the improvement of resolution steps is quite satisfying.

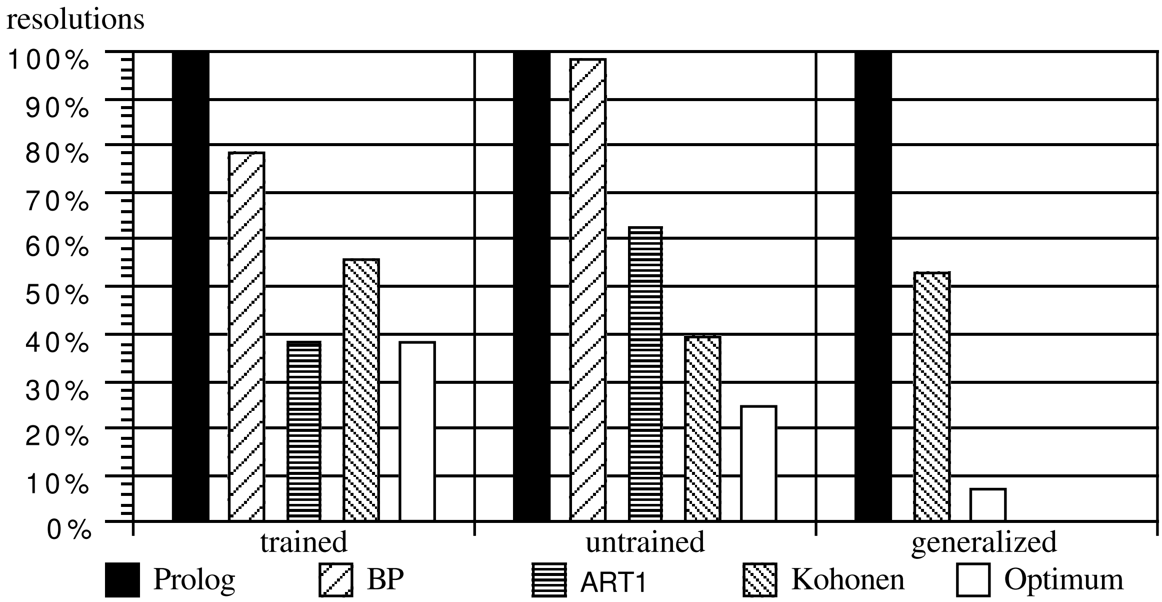

Fig. 6. Results for networks

The modified ART1 network is able to recall the trained control knowledge exactly. The results in generalizing to unknown proofs are worse than the performance of the self organizing feature maps. On the other hand the feature maps are not able to reproduce all the trained proofs correctly; however, their performance in generalizing to unknown proofs is quite impressive. These observations suggest a relationship between the ability to generalize and to store knowledge in connectionist networks.

6. DISCUSSION

The representation of control knowledge for Prolog programs has been proposed before. For example Kasif suggests representing control knowledge as an expression in a procedural language [6]. Others propose a declarative representation of control knowledge [2; 3; 15]. This means conditions are formulated on which certain clauses may be used to prove a partial goal. All these approaches require a programmer to formulate the control knowledge. Unlike the declarative approaches mentioned above, our description consists of both the instantiation status and identities and/or inclusions of arguments. As far as clause selection is concerned we use a much more detailed description of instantiated arguments [4, 16]. To improve efficiency at the selection of clauses it is important not only to consider the instantiation status of arguments but also the types of the arguments. Suttner introduces control to the selection of axioms in theorem provers [13]. A proof is observed and certain features of the proof are counted, e.g. the number of variables in the theorem. In contrast to our approach this encoding loses information about the positions of the variables.

7. CONCLUSION

In this paper we presented a new approach to learn and memorize control knowledge for Prolog programs. Trained with examples of successful proofs, different networks learn a control strategy to a different degree. As supervised learning networks, we examined backpropagation networks. As unsupervised models, we tested modified ART-type networks and Kohonen's feature maps. Proof strategies are declaratively described by certain features as the instantiation and the identities of the literals' arguments. Efficient proof strategies described by these features are used to train the network in order to generalize to similar selection-situations occurring in other proofs. The results of the experiments show the different abilities of the three network types to learn the encoded description of proofs. Embedded in our system, the networks are able to generalize a strategy for clause and literal selections to new but similar proofs. In particular, we found that appropriately modified unsupervised learning paradigms can perform typical supervised learning tasks better than the standard backpropagation algorithm. Moreover, our results suggest a trade off between the ability of the different networks to generalize and to reproduce knowledge. An important advantage of this approach is the ability to learn proof heuristics from relatively few examples. Proof strategies that have been effectively used in the past may be of use in new proofs.

ACKNOWLEDGEMENTS

This work has been supported in parts by the Forschungspreis Nordrhein-Westfalen. We also would like to thank all members of the PANDA (Prolog and Neural Distributed Architectures, Department of Computer Science, University of Dortmund) for their encourage.

REFERENCES

[1] G.A. Carpenter, S. Grossberg: *A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine*; 'Computer Vision, Graphics, and Image Processing', Vol. 37, 1987, pp. 54-115

[2] H. Gallaire, C. Lasserre: *Metalevel Control for Logic Programs,* in: Clark/Tärlund (Eds.): Logic Programs, Academic Press, London, 1982, pp.173-188

[3] M.R. Genesereth, M.L. Ginsberg: *Logic Programming* in: Communications of the ACM, Vol. 28 Nº 9, 1985, pp.933-941

[4] R. Hannuschka, U. Hartmann, M. Mandischer, V. Weber:*Control of Symbolic Proofs with Connectionist Networks*, in: [18], in German

[5] G.E.Hinton, J.L.McClelland, D.E.Rumelhart: *Distributed Representations*. In: [12], pp.77-109

[6] S. Kasif, M. Kohli, J. Minker: *Prism: A Parallel Inference System for Problem Solving,* in: Proceedings of the 8th IJCAI, Karlsruhe 1983, pp.544-546

[7] T. Kohonen: *Self-Organisation and Associative Memory*, Springer Verlag, Berlin, 1984

[8] R. Kowalski: *Algorithm = Logic + Control,* in: Communications of the ACM, Vol. 33 Nº 7, July 1979, pp.424-436

[9] J.L. McClelland, A.H. Kawamoto: *Mechanisms of Sentence Processing: Assigning Roles to Constituents*. In: [12], pp.272-326

[10] L. Naish: *Prolog Control Rules,* Technical Report, No.13, University of Melbourne, Australia, 1984

[11] J.A. Robinson: *A machine-oriented Logic based on the Resolution Principle,* in: Journal of the ACM, Vol. 12 Nº 1, Jan. 1965, pp.23-44

[12] D.E. Rumelhart, J.L. McClelland: *Parallel Distributed Processing : Explorations in the Microstructure of Cognition,* MIT Press, 1986

[13] C.B. Suttner: *A feature-based heuristic module for the SETTHEO theorem prover,* Forschungsgruppe Künstliche Intelligenz, TU München, 1989

[14] M. Takeda, J.W. Goodman: *Neural networks for computation*: *number representations and programming complexity*. In: Applied Optics 25 (1986), pp. 3033-3046

[15] A. Ultsch: *Control for Knowledge-based Information Retrieval,* Dissertation, Swiss Federal Institute of Technology, Zurich, 1987

[16] A. Ultsch, R. Hannuschka, U. Hartmann, V. Weber: *Learning of Control Knowledge for Symbolic Proofs with Backpropagation networks*, in: R. Eckmiller, G. Hartmann, G. Hauske (eds.); Parallel Processing in Neural Systems and Computers, Elsevier Science Publishers (North Holland), Amsterdam, 1990, pp.499-503

[17] A. Ultsch, R. Hannuschka, U. Hartmann, M. Mandischer, V. Weber: *Connectionist Represented Control Knowledge*, in: Proceedings of the 3rd Neuro-Nimes 1990

[18] A. Ultsch (ed.), *Coupling of declarative and connectionist knowledge representation*, Res.-Rep.352, Department of Computer Science, University of Dortmund, April 1990, in German