ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 45-50

Learning Search-Control Heuristics for Automated Deduction Systems with Folding Architecture Networks

Christoph Goller
Institut für Informatik, TU München, Arcisstr. 21,
D-80290 München, Germany, goller@informatik.tu-muenchen.de

Abstract. During the last years, folding architecture networks and the closely related concept of recursive neural networks have been developed for solving supervised learning tasks on data structures. In this paper, these networks are applied to the problem of learning search-control heuristics for automated deduction systems. Experimental results with the automated deduction system SETHEO in an algebraic domain show a considerable performance improvement. Controlled by heuristics which had been learned from simple problems in this domain the system is able to solve several problems from the same domain which had been out of reach for the original system.

1. Introduction

In almost all fields of scientific and technical reasoning, people and systems assisting them have to deal with *structured objects*. Examples are chemical structures, algebraic (mathematical) expressions, software source code, and all kinds of graphs. Informally speaking, structured object are composed of 'smaller' objects, which may be structured too. Though the objects that are usually considered are finite, the size of objects within one domain is often not limited and one normally has objects of very different size within one domain. This contrasts with the *static* kind of data (*fixed-length real vectors*) usually handled by statistical pattern recognition or neural network approaches.

During the last years, *folding architecture networks* (*FAs*) [4, 3] and the closely related concept of *recursive neural networks* [2] have been developed for solving supervised learning tasks (such as classification and prediction) on data structures. Applications in the fields of chemistry and logo recognition are described e.g. in [8, 1]. In this paper, *FAs* are applied in the field of automated deduction. The goal in *automated deduction*[1] (*AD*) is to automatically find formal proofs for conjectures based on sets of axioms. Both, conjectures and axioms are specified in a formal language (e.g. 1st order predicate logic). The central problem in *AD* is the explosive growth of *search spaces* with proof length. Therefore methods of guiding and controlling the search process are

[1] Most of the symbolic AI systems perform some kind of logic-based deductive inference.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 45-50

indispensable. In this paper, *FAs* are applied for learning *search-control heuristics* for *AD*-systems from examples of successful proof searches. In particular, *FAs* are used for learning *heuristic evaluation functions* (*HEFs*) for algebraic expressions, and these *HEFs* are then used for controlling the search process of an *AD*-system. For a more detailed description of the application see also [3].

The paper is organized as follows. In Section 2, *FAs* are briefly reviewed and the kind of structured objects we are interested in is specified. Section 3 shows how *HEFs* for algebraic expressions can be used to represent search-control heuristics for *AD*-systems. Furthermore, a concept for learning such *HEFs* from examples of successful search processes is presented. Experimental results with the *AD*- system SETHEO are presented in Section 4. Section 5 concludes the paper.

2. Folding Architecture Networks

The structured objects that can be handled by *FAs* are *finite labelled ordered trees* in the following simply called labelled trees. The labels are assumed to be real vectors of fixed uniform length. To each node a label is attached and there are no labels for edges. The order concerns the children of a node and we assume that there is a limit (*maximum out-degree*) for the number of children. Furthermore, each tree has exactly one *root-node*. *Logical terms or algebraic expressions* over a finite set of symbols (signature) can easily be represented as labelled trees by choosing the maximum arity of all symbols as maximum out-degree and introducing a function that maps the symbols to real vectors.

FAs are a recursive variant of the standard multi-layer feed-forward network. They can be used to learn approximations to mappings from labelled trees to $\mathbb{R}^n$ based on *samples* for these mappings. A *FA* is composed of two multi-layer feed-forward networks, viz. the recursively applied *encoder network* N_{enc} and the *transformation network* N_{trans}. Usually sigmoid activation functions are considered. N_{enc} is used to compute a *compressed representation* (a fixed length vector) for a given labelled tree while N_{trans} takes this representation as input and computes the final output of the whole network. Figure 1 shows an example of a *FA* which can process labelled trees with a maximum out-degree of two. In this example both N_{enc} and N_{trans} are single-layer feed-forward networks. Note that in case of a maximum out-degree of one and single-layer N_{enc} and N_{trans} we get a simple recurrent network.

Let $Tree$ represent the domain of labelled trees that have to be processed with a maximum out-degree of $out_{max} \in \mathbb{N}$. Furthermore, let $s_l, s_{rep}, s_{out} \in \mathbb{N}$ denote the dimensions of labels, compressed representations, and the network's final output, respectively. Then N_{enc} has $s_{tot} = s_l + out_{max} \times s_{rep}$ input neurons and s_{rep} output neurons. Thus, N_{enc} computes a function $f_{enc} : \mathbb{R}^{s_l + out_{max} \times s_{rep}} \rightarrow \mathbb{R}^{s_{rep}}$. Assume we have a labelled tree $t = l(t_1, \ldots, t_n)$, where $l \in \mathbb{R}^{s_l}$ is the label of the root-node of t and $t_1, \ldots, t_n \in Tree$ is the ordered list of children of t. Let $\bullet$ denote vector concatenation. Then the function $enc : Tree \rightarrow \mathbb{R}^{s_{rep}}$ which computes the compressed representations

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 45-50

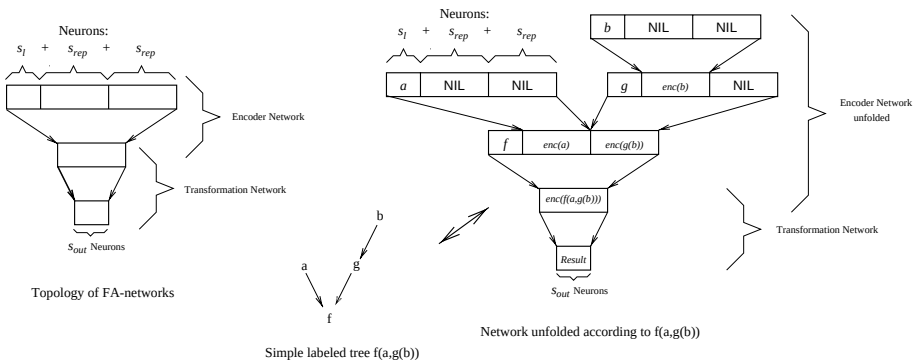

Figure 1: Example of a *FA* processing the logical term $f(a, g(b))$.

is defined by $enc(l(t_1, \ldots, t_n)) := f_{enc}(l \bullet enc(t_1) \bullet \ldots \bullet enc(t_n) \bullet \mathsf{NIL}^{(a_{max}-n)})$. We say the network is *virtually unfolded* to encode t. Leaf-nodes are encoded first and composite trees are encoded using the representations of their children. $\mathsf{NIL} \in \mathbb{R}^{s_{rep}}$ is a fixed representation indicating the absence of a child. The transformation network N_{trans} has s_{rep} input neurons and s_{out} output neurons. Thus, it computes a function $f_{trans} : \mathbb{R}^{s_{rep}} \to \mathbb{R}^{s_{out}}$. By using the output of the encoder network N_{enc} as the input for N_{trans}, the total network computes a function $f_{trans} \circ enc : Tree \to \mathbb{R}^{s_{out}}$. See also Figure 1.

It has been shown that *FAs* are universal approximators for functions from labelled trees to real vector spaces [5] and that any bottom-up tree-automaton can be simulated by a *FA* [6]. *FAs* can be trained by *back-propagation through structure* (*BPTS*), a recursive variant of standard back-propagation. *BPTS* means that the error at the output layer of N_{trans} is propagated back through the entire unfolded network. In this way the exact gradient with respect to the weights in N_{trans} and N_{enc} is computed and the training is completely supervised. By representing identical subtrees which occur at different positions only once, the labelled trees of a training set can be represented very efficiently as one minimal directed acyclic graph, and this can speed up gradient computation with *BPTS* considerably.

3. Learning Heuristic Evaluation Functions for Controlling Search in Automated Deduction

Figure 2 shows a *finite initial part* of a *search-tree* generated by an *AD*-system for one proof problem. Nodes and edges of such a search-tree represent *states* of the *AD*-system and *inference steps*, respectively. Both, inference steps and states are logical expressions or sets of such expressions. Proofs are represented by states with specific properties which can be tested easily. Search-control heuristics can be represented as *HEFs* which compute numeric ratings for states or inference steps. For the following we restrict ourselves to *HEFs* for states and

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 45-50

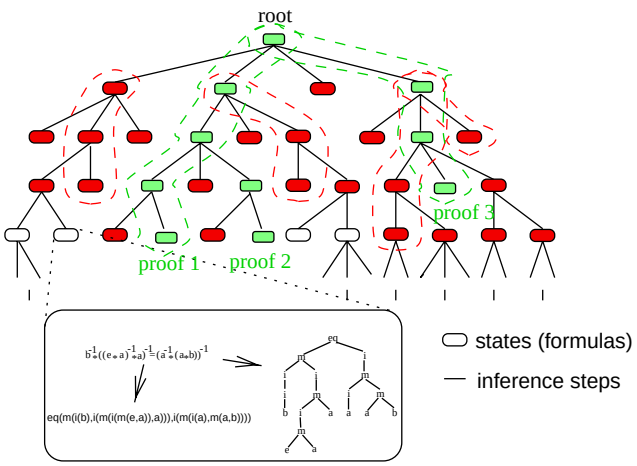

Figure 2: Search-tree of an automated deduction system for one proof problem.

assume that these states are logical expressions[2]. Without loss of generality we further assume that high ratings mean good ratings and low ratings mean bad ones. Furthermore, a search strategy that globally compares ratings is used. This means that state s_i is explored before state s_j if $\exists_{s'_j \in pred(s_j)} \forall_{s'_i \in pred(s_i)} : eval(s'_i) > eval(s'_j)$. Here *eval* denotes a *HEF* and *pred* the reflexive and transitive closure of the immediate predecessor relation in a search-tree.

Normally, there is an infinite number of possible proofs for a provable conjecture. The user of an *AD*-system is only interested in finding at least one proof for every problem and usually does not care which proof is found. Experience with hand-crafted heuristics for *AD*-systems shows two properties of good search-control heuristics. Firstly, they are usually *domain-specific*. Secondly, they not necessarily allow all possible proofs for provable problems of the domain. Instead they focus on proofs with specific properties. Therefore, heuristics should be learned for specific domains and a method for learning heuristics should have the freedom to decide by itself on which proofs to focus.

When a finite initial part of a search-tree for a proof problem has been explored using brute-force search or a standard heuristic and when at least one proof has been found, a *training sample* for learning a *HEF* can be extracted as follows. All states lying on a path to a proof (*solution-path*) can be used as examples with positive target rating. All states not lying on a solution-path could be used as examples with negative rating. However, the number of such negative examples is usually too high. It is reasonable to restrict the negative examples to states lying close (in the search-tree) to positive examples.

Unfortunately, standard error measures such as the quadratic error measure are not adequate for learning a *HEF* based on a training sample because individual training examples are not independent of each other. Informally

[2]The concept for learning *HEFs* can easily be extended to *AD*-systems that have states consisting of sets of expressions or to learning *HEFs* for inference steps [3].

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 45-50

speaking, a high rating for a state on a solution-path is only useful for finding the respective proof if all states on this solution-path have a high rating. However, if all states on one solution-path have a high rating it is not harmful if some positive examples lying on other solution-paths have low ratings, since one is only interested in finding one proof for each proof problem. Furthermore, the number of examples in the training sample is usually small compared to the number of states that can be explored by the *AD*-system in a reasonable amount of time. Therefore, a *HEF* is regarded as perfect w.r.t. a sample if it keeps search within the sample. A detailed analysis [3] leads to the following error measure (minimization) for learning *HEFs*

$$E[eval] = \begin{cases} 0 & \text{if } \max\{\min\{eval(s)\}_{s\in l}\}_{l\in L_{pos}} > \max\{\min\{eval(s)\}_{s\in l}\}_{l\in L_{neg}} \\ 1 & \text{otherwise} \end{cases}$$

where L_{pos} is the set of all solution-paths and L_{neg} the set of *failure-paths*. A solution-path is the set of all nodes lying on a path from the root of the search-tree to a proof. Each negative example in the sample which does not have a successor state that is a member of the sample defines a failure-path. This failure-path contains the negative example itself and all predecessor states that are not members of all solution-paths (see Figure 2). Since a gradient-based method is used to learn *HEFs* (Section 2), we need a continuous differentiable error measure. This can be achieved by using the following approximations $\max_n^\beta\{x_i\}_{i\in\{1,\ldots,n\}} := \frac{1}{\beta}\ln\left(\frac{1}{n}\sum_{i=1}^n e^{\beta x_i}\right)$ and $\min_n^\beta\{x_i\}_{i\in\{1,\ldots,n\}} := -\frac{1}{\beta}\ln\left(\frac{1}{n}\sum_{i=1}^n e^{-\beta x_i}\right)$ in $E[eval]$. They can be derived by integrating the generalized sigmoid function [3]. So far $E[eval]$ is only defined w.r.t. a training sample coming from the search-tree of one proof problem. It is straight-forward to combine samples from several proof problems by using the maximum (again the approximation) of their error contributions as overall error measure.

4. Experimental Results with the Theorem Prover SETHEO

The error measure described in Section 3 and *FAs* have been used for learning *HEFs* for the *AD*-system SETHEO. The standard version of SETHEO is one of the best existing general purpose *AD*-systems [7]. The experiments on learning *HEFs* have been carried out in the domain of word problems in group theory (*WPGT*). It is of course well-known that *WPGT* are decidable. However, for a general purpose system like SETHEO which in the version that was used does not have a special equality handling[3] (equality plays a major role for *WPGT*), *WPGT* can be very difficult and even intractable. For our purposes, *WPGT* have two big advantages. Firstly, it is known that there are good strategies for solving these problems. Therefore, we can be sure, that there is something

[3] For the experiments axiomatic equality was used.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 45-50

to learn here. Secondly, an arbitrary number of proof problems of different complexity are available.

In the experiments, 300 relatively simple *WPGT* were used for extracting sample data and several *FAs* were trained[4]. Testing was done on 19 problems which were intractable for the standard version of SETHEO within a time limit of 80*s* on a SUN Ultra 1. SETHEO controlled by the best learned heuristic was able to solve 17 of these test problems (time limit 80*s*) with an average search time of 2.9*s*. E-SETHEO[5], a version of SETHEO with special equality handling was also able to solve 17 problems within the time limit of 80*s*. However, the average search time was 5.5*s*.

5. Conclusion

The experiments show that for the domain of *WPGT* it is possible to automatically tune the general purpose system SETHEO by learning *HEFs* with *FAs* and achieve an even better performance than the specially developed E-SETHEO. These results are very promising. However, since word problems in group theory are generally regarded as trivial the next step is to experiment in a more realistic application domain.

The approach for learning heuristics which is presented in this paper could be useful for learning heuristics for general search-based systems. Last but not least the continuous differentiable approximations for the minimum and maximum (Section 3) could be useful for other neural network applications where training examples are not independent of each other.

References

[1] P Frasconi, M. Gori, S. Marinai, J. Sheng, , G. Soda, and A. Sperduti. Logo Recognition by Recursive Neural Networks. In *Proceedings of GREC97*, pages 144–151, 1997.

[2] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5), September 1998.

[3] C. Goller. *A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems.* PhD thesis, TU Munich, Computer Science, 1997.

[4] C. Goller and A. Küchler. Learning Task-Dependent Distributed Representations by Backpropagation Through Structure. In *Proceedings of the ICNN-96*, IEEE Press, June 1996.

[5] B. Hammer and V. Sperschneider. Neural Networks can approximate Mappings on Structured Objects. *Second International Conference on Computational Intelligence and Neuroscience*, March 1997.

[6] A. Küchler. On the Correspondence between Neural Folding Architectures and Tree Automata. Technical Report 98-06, Dept. of Neural Information Processing, Computer Science, University of Ulm, 1998.

[7] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *Journal of Automated Reasoning*, 1997.

[8] T. Schmitt and C. Goller. Relating Chemical Structure to Activity with the Structure Processing Neural Folding Architecture. In *Proceedings of the EANN'98*, Gibraltar, June 1998.

[4] A part of the SETHEO-tableau was used to represent the state of SETHEO.

[5] The winner of the 1996 international theorem proving competition.