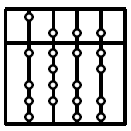
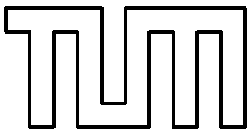


FAKULTÄT FÜR INFORMATIK
der Technischen Universität München
Lehrstuhl VIII
Forschungsgruppe Automated Reasoning



Learning Task-Dependent Distributed Representations by Backpropagation Through Structure

Report AR-95-02

Christoph Goller
Institut für Informatik
TU München
Germany

Andreas Küchler
Computer Science
University of Ulm
Germany

TU München, Arcisstr. 21, 80290 München, Germany

Learning Task-Dependent Distributed Representations by Backpropagation Through Structure

Christoph Goller
Automated Reasoning Group
Computer Science Institute
Technical University Munich
D-80290 München
Germany
goller@informatik.tu-muenchen.de

Andreas Küchler
Neural Information Processing Department
Computer Science
University of Ulm
D-89069 Ulm
Germany
kuechler@informatik.uni-ulm.de

AR-95-02

Abstract

While neural networks are very successfully applied to the processing of fixed-length vectors and variable-length sequences, the current state of the art does not allow the efficient processing of structured objects of arbitrary shape (like logical terms, trees or graphs). We present a connectionist architecture together with a novel supervised learning scheme which is capable of solving inductive inference tasks on complex symbolic structures of arbitrary size. The most general structures that can be handled are *labeled directed acyclic graphs*. The major difference of our approach compared to others is that the structure-representations are exclusively tuned for the intended inference task. Our method is applied to tasks consisting in the classification of logical terms. These range from the detection of a certain subterm to the satisfaction of a specific unification pattern. Compared to previously known approaches we got superior results on that domain.

1 Introduction

Classical symbolic systems typically use complex composite symbolic structures like trees, graphs, lists and terms as knowledge representation formalisms. In the late eighties connectionism had been blamed of being unable to represent and process complex composite symbolic structures like trees, graphs, lists and terms. One of the most convincing counterexamples to this criticism was given with the Recursive Autoassociative Memory (RAAM) [Pol90], a method for generating fixed-width distributed representations for variable-sized recursive data structures.

There have been several publications showing the appropriateness of representations produced by the RAAM for subsequent classification tasks [Cad94, GSS95b] and also for more complex tasks even with structured output [DSB92, Chr91, Nic94]. Our approach, however, is different. We present a novel learning scheme that we call *backpropagation through structure* (BPTS) in analogy to *backpropagation through time* for recurrent networks [RGM86, Wer90]. It allows us to devise distributed representations exclusively with respect to the task that has to be performed.

In Section 2 we describe the learning scheme in detail. The space and time complexity of our learning scheme is not higher than that of the ordinary RAAM training method. Furthermore, if we do not deal with structured output, we can get rid of the moving target problem. Section 3 presents a set of basic classification tasks (detectors) on logical terms. It is an extended version of the problem set, used in [GSS95b]. In Section 4 we show, that all classification problems can be solved with smaller networks, less training epochs and better classification results, than with using the ordinary RAAM learning scheme in [GSS95b].

2 The Architecture

2.1 Labeling RAAM

The Labeling RAAM (LRAAM) [Spe94] is an extension of the RAAM [Pol90] model[1] that learns fixed-width distributed representations for *labeled* variable-sized recursive data structures. As most general example for such structures, a labeled directed graph will be used. The general structure for an LRAAM is that of a three-layer feedforward network (see Figure 1, part A).

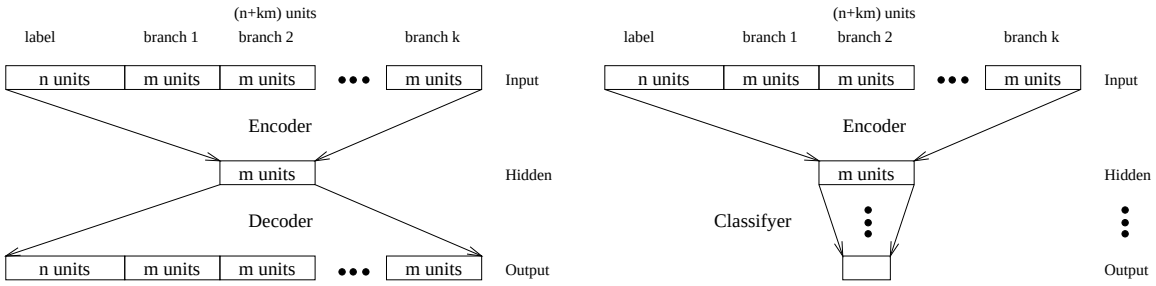


Figure 1: The standard LRAAM (A) and our architecture (B).

The idea is to obtain a compressed representation (hidden layer activation) for a node of a labeled directed graph by allocating a part of the input (output) of the network to represent the label and the rest to represent its subgraphs with a fixed maximum number of subgraphs k using a special representation for the empty subgraph. The network

[1] In the following sections we will always refer to the LRAAM-model as it is more suitable for the representation of terms than the plain RAAM.

is trained by backpropagation in an autoassociative way using the compressed representations recursively. As the representations are consistently updated during the training, the training set is dynamic (moving target), starting with randomly chosen representations. Once the training is complete, the patterns of activation representing graphs can be decoded recursively to retrieve information.

Notice that multiple graphs can be encoded in the same LRAAM and that in order to learn a representation for one graph, representations for all its subgraphs have to be learned.

2.2 Transformational and Confluent Inference

In the past several attempts have been made to integrate RAAM-style models with other modules (e.g. feedforward networks) into an entire connectionist architecture which is capable of performing structure-sensitive inference tasks on composed objects. Chrisman [Chr91] identified two different principles of the generation and processing of distributed reduced representations w.r.t. a given inference task: *transformational* and *confluent inference*. The former could be achieved by first computing unique distributed representations with the RAAM, freezing them and then using them as input for subsequent modules doing the inference task [DSB92, Cad94]. Transformational inference does not imply a very tight coupling. RAAM and inference module may consist of separately trained networks based on different models. Confluent inference takes the intended inference task into account *while* learning distributed representations for structures. RAAM and inference modules have to be in strong interaction, network models and training procedures have to be linked together [Chr91, GSS95b].

2.3 A Simple Architecture for the Classification of Structures

Our idea – in contrast to previous approaches – is that for many inference tasks unique representations are not necessary as long as the information needed for the inference task is represented properly. This is intuitive for our term-classification problems (see Section 3) and one result of [GSS95b] was, that the more difficult examples could be solved with the confluent training scheme while it was impossible to devise unique representations. In our model, the encoding of structures into distributed representations is melted with further inference into one single process and therefore has to be considered as rigorous confluent (in the sense of [Chr91]).

For reasons of clarity and simplicity we will concentrate on inference tasks consisting of the classification of logical terms in the following. However note, that our architecture together with the corresponding training procedure (Section 2.4) could be easily extended to handle inference tasks of more complex nature. Figure 1 (part B) shows the architecture we use. The first two layers occupy the role of the standard LRAAM encoder part, the hidden units are connected to a simple sigmoid feedforward layer, in our case just one unit for classification. There is no decoder.

2.4 Backpropagation Through Structure

The principle idea of using recursive backpropagation-like learning procedures for symbolic tree-processing has been mentioned first in [Ber91, SW92]. We assume in the following the reader to be familiar with the standard backpropagation algorithm (BP) [RGM86] and its variant *backpropagation through time* (BPTT) [Wer90] that is used to train recurrent network models.

2.4.1 Representing Structures as DAG's

Let us first have a closer look on the way structures are encoded by the LRAAM. All kinds of recursive symbolic data structures we aim at can be mapped onto labeled directed acyclic [2] graphs (DAGs). In order to compute the representation of a graph, the representations of all subgraphs have to be computed first. During the training phase of the LRAAM for each node one phase of forward propagation of activations and one phase of backward propagation (each through the three layers of the network) of errors per epoch is needed. Choosing a DAG-representation for structures – that allows to represent different occurrences of a (sub-)structure in the training set only as one node – may lead to a considerable (even exponential) reduction of complexity for the standard LRAAM. This argument also holds for our architecture (see Section 2.4.3). Instead of choosing a tree-like representation we therefore prefer a DAG-like representation for our terms as shown in Figure 2.

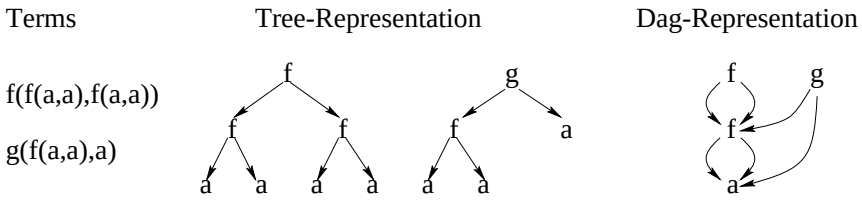


Figure 2: Tree and DAG representation of a set of terms.

2.4.2 BPTS for Trees

For reasons of simplicity we first restrict our considerations to tree-like structures. In the *forward phase* the encoder (Figure 1, part B) is used to compute a representation for a given tree in the same way as in the plain LRAAM. This is done by recursively passing the previous computed representations of the direct subtrees to the input layer of the encoder. This encoding process is started at the leaves of the tree and generates a representation for the tree which is then passed to the next layer yielding the classification result at the output unit. Following metaphor helps us to explain the *backward phase*. Imagine the encoder is virtually unfolded (with copied weights) according to the tree structure (see Figure 3).

[2] We do not deal with cyclic structures here.

Now the error passed from the classifier to the hidden layer is propagated through the unfolded encoder network.

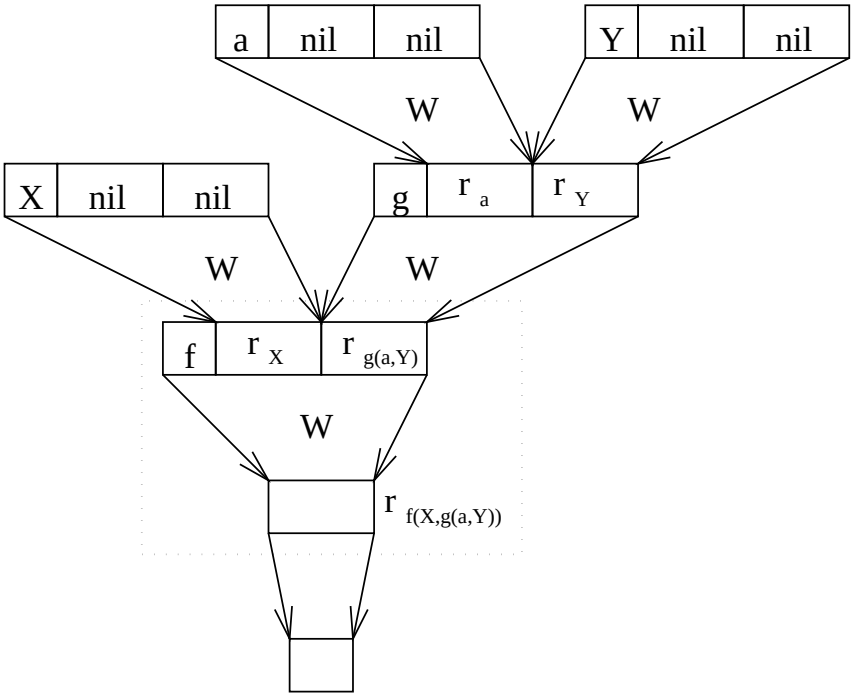


Figure 3: The encoding network unfolded by the structure $f(X, g(a, Y))$.

This unfolding by structure in analogy to unfolding a recurrent network in time (BPTT) has motivated us to introduce the term *backpropagation through structure* (BPTS).

Let us first consider the case, that for each occurrence of a (sub)term one dedicated node in the training set is reserved (tree-representation). With a similar argument as for BPTT [Wer90] we can see, that the exact gradient is computed. Imagine, that each copy of the encoder part has its own set of weights. Then, by the correctness of ordinary BP, the exact gradient is computed. If the weight matrices of the different copies are identified, it is clear, that we just have to sum the components coming from different copies to get the exact gradient. The exact formulation is given below:

For each (sub-)tree t, $t.x \in R^{n+km}$ is the input vector of the encoder, $\vec{\delta_t} \in R^m$ the vector of (error-) deltas for t's representation, and $\prod(t.x, t')$ the projection of $t.x$ onto t's subtree t'. Let further W be the encoder matrix, f' the derivative of the transfer function and $\odot$ the multiplication of two vectors by components.

ΔW is calculated as sum over all (sub-)trees (1). The $\vec{\delta_{t'}}$ for each subtree t' is calculated by propagating the $\vec{\delta_t}$ of the one definite parent node t of t' back according to (2):

$$\Delta W = \eta \sum_t \vec{\delta_t}(t.x)^T \qquad (1) \qquad\qquad \vec{\delta_{t'}} = \prod(W^T \vec{\delta_t} \odot f'(t.x), t') \qquad (2)$$

For each (sub-)tree in the training set one epoch requires exactly one forward and one backward phase through the encoder. The training set is static (no moving target).

2.4.3 BPTS for DAGs

However, if we use a DAG-representation and represent a (sub-)structure t only as one node independently from the number of its occurrences, then there may be different $\vec{\delta_t}$ in (1) and (2) for each occurrence of t. We call this situation a *delta conflict*.

Suppose the (sub-)structures t_i and t_j are identical. Of course this means that corresponding substructures within t_i and t_j are identical too. This clearly gives us $t_i.x = t_j.x$, but we may have $\vec{\delta_{t_i}} \neq \vec{\delta_{t_j}}$

For calculating ΔW we only need the sum of $\vec{\delta_{t_i}}$ and $\vec{\delta_{t_j}}$. This is shown by the following transformation of (1) which holds because of the linearity of matrix multiplication:

$$\Delta W = \eta \sum \ldots + \vec{\delta_{t_i}}(t_i.x)^T + \vec{\delta_{t_j}}(t_i.x)^T = \eta \sum \ldots + (\vec{\delta_{t_i}} + \vec{\delta_{t_j}})(t_i.x)^T$$

The $\vec{\delta_{t'}}$'s of corresponding children t' of t_i and t_j can be calculated more efficiently too, by propagating the sum of $\vec{\delta_{t_i}}$ and $\vec{\delta_{t_j}}$ back in (2). A similar transformation (linearity of $\prod, \odot$ and matrix multiplication) for (2) shows this.

Summing up all different δ's coming from each occurrence of a (sub-)structure is a correct (steepest gradient) solution of the delta conflict and enables a very efficient implementation of BPTS for DAGs. We just have to organize the nodes of the training set in a topological order. The forward phase starts with the leaf-nodes and proceeds according to the reverse ordering – ensuring that representations for identical substructures have to be computed only once. The backward phase follows the topological order beginning at the root-nodes. In this way the δ's of all occurrences of a node are summed up before that node is processed. Again for each node in the training set one epoch requires exactly one forward and one backward phase through the encoder.

2.4.4 Online versus Batch Mode

Similar to standard BP, BPTS can be used in *batch* or in *online* mode. BPTS-batch updates the weights after the whole training set has been presented. By the optimization techniques discussed in Section 2.4.3 (δ-summation and DAG-representation) each node has to be processed only once per epoch. This does not hold for online mode because the weights are updated immediately after one structure has been presented and therefore substructures have to be processed for each occurrence separately.

3 Classification Tasks for Terms

In this section we give a short description of the term classification problems used to test our approach. The set of problems is an extension of the one used in [GSS95b]. We consider only the representation and classification of *ground* terms, i.e. terms that do not involve variables, however, the classification tasks we propose involve the concept of logical variable. We have summarized the characteristics of each problem in Table 1. The first column of the table reports the name of the problem, the second one the set of symbols

Term Classification Problems

Problem	Symbols	Positive Examples.	#terms (tr.,test)	#subterms (tr.,test)	depth (pos.,neg.)
lblocc1 long	f/2 i/1 a/0 b/0 c/0	no occurrence of label c	(259,141)	(444,301)	(5,5)
termocc1	f/2 i/1 a/0 b/0 c/0	the (sub)terms i(a) or f(b,c) occur somewhere	(173,70)	(179,79)	(2,2)
termocc1 very long	f/2 i/1 a/0 b/0 c/0	the (sub)terms i(a) or f(b,c) occur somewhere	(280,120)	(559,291)	(6,6)
termocc2	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	the (sub)terms i(a) or f(b,c) occur somewhere	(400,200)	(2563,1394)	(7,7)
inst4	f/2 a/0 b/0 c/0	instances of f(X,f(a,Y))	(175,80)	(179,97)	(3,2)
inst4 long	f/2 a/0 b/0 c/0	instances of f(X,f(a,Y))	(290,110)	(499,245)	(7,6)
inst5	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	instances of t(a,i(X),f(b,Y)), g(i(f(b,X)),j(Y)) or g(i(X),i(f(b,Y)))	(268,132)	(1242,642)	(7,6)
inst5 simil neg.	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	instances of t(a,i(X),f(b,Y)), g(i(f(b,X)),j(Y)) or g(i(X),i(f(b,Y)))	(259,141)	(1394,776)	(7,7)
instocc1	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	instances of f(j(X),f(a,Y)), g(f(a,Y),i(X)) or i(j(f(a,Y))) occur somewhere	(217,83)	(1247,497)	(7,6)
instocc1 simil neg.	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	instances of f(j(X),f(a,Y)), g(f(a,Y),i(X)) or i(j(f(a,Y))) occur somewhere	(262,138)	(2840,1931)	(9,9)
inst1	f/2 a/0 b/0 c/0	instances of f(X,X)	(200,83)	(235,118)	(3,2)
inst1 long	f/2 a/0 b/0 c/0	instances of f(X,X)	(202,98)	(403,204)	(6,6)
inst7	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	instances of t(i(X),g(X,b),b)	(191,109)	(1001,555)	(6,6)
inst7 simil neg.	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	instances of t(i(X),g(X,b),b)	(276,124)	(1667,749)	(7,7)

Table 1: Description of a set of classification problems involving logical terms.

(with associated arity) compounding the terms, and the third column shows the rule(s) used to generate the positive examples with upper case letters representing all-quantified logical variables. The fourth column reports the number of terms in the training and test set respectively, the fifth column the number of subterms (using the DAG-representation as described in Section 2.4.1), and the last column the maximum depth[3] of terms.

For each problem about the same number of positive and negative examples is given. Both positive and negative examples have been generated randomly. Training and test sets are disjoint and have been generated by the same algorithm. For examples with many different symbols and complex patterns defining the positive class, randomly generating negative examples leads to terms not very similar to positive ones. In these cases, we have generated additional problem instances (named `simil neg.`) with specially generated negative examples, differing from positive ones only by one symbol.

It must be noted that the set of proposed problems range from the detection of a particular symbol (label) in a term to the satisfaction of a specific unification pattern. Specifically, in the unification patterns for the problems `inst1` and `inst7` the variable X occurs twice making these problems much more difficult than `inst4` or `inst5`, because any classifier for these problems would have to compare arbitrary subterms corresponding to X. However concerning the data sets presented here, this is true only for inst1. For problem `inst7`, randomly generating negative examples didn't result in an instance of $t(i(X), g(Y, b), b)$ with $X \neq Y$ neither in the training nor in the test set, what may make

[3]We define the depth of a term as the maximum number of edges between the root and leaf nodes in the term's LDAG-representation.

Best Results

Problem	Method	Topology		Learning Par.			% Dec.-Enc.	% Tr.	% Ts.	#epochs
		#L	#H	η	ϵ	μ				
lblocc1 long	△ lraamc	8	35	0.2	0.001	0.5	4.25	100	98.58	11951
	□ batch	8	2	0.001		0.6		99.61	100	108
	■ online	8	2	0.001		0.6		100	100	444
termocc1	△	8	25	0.1	0.1	0.2	100	98.84	94.29	27796
	□	8	5	0.001		0.6		97.69	95.71	5271
	■	8	5	0.001		0.6		100	94.29	6925
inst1	△	6	35	0.2	0.06	0.5	100	97	93.98	10452
	□	6	3	0.001		0.6		97	93.98	278
	■	6	5	0.005		0.6		97.5	91.57	213
inst1 long	△	6	45	0.2	0.005	0.5	36.14	94.55	90.82	80000
	□	6	3	0.005		0.6		95.56	92.86	4250
	■	6	3	0.005		0.6		98.52	94.90	465
inst4	△	6	35	0.2	0.005	0.5	98.86	100	100	1759
	□	6	3	0.001		0.6		100	100	37
	■	6	3	0.001		0.6		100	100	68
inst4 long	△	6	35	0.2	0.005	0.5	8.97	100	100	6993
	□	6	3	0.003		0.6		100	100	150
	■	6	3	0.005		0.6		100	100	27
inst7	△	13	40	0.1	0.01	0.2	1.05	100	100	6158
	□	13	3	0.01		0.6		100	100	10
	■	13	3	0.01		0.6		100	100	57
termocc1 very long	□	8	6	0.001		0.6		99.64	98.33	3522
	■	8	6	0.001		0.6		99.29	94.17	2263
termocc2	□	13	4	0.001		0.6		62.5	61	867
	■	13	5	0.001		0.6		70	57.5	230
inst5	□	13	3	0.001		0.6		98.88	94.70	2155
	■	13	3	0.001		0.6		99.63	97.73	95
inst5 simil neg.	□	13	3	0.001		0.6		94.21	93.62	1558
	■	13	5	0.001		0.6		98.06	92.20	198
instocc1	□	13	6	0.001		0.6		96.31	80.72	338
	■	13	7	0.001		0.6		97.24	72.29	369
instocc1 simil neg.	□	13	5	0.001		0.6		79.77	52.17	561
	■	13	6	0.001		0.6		94.28	55.07	544
inst7 simil neg.	□	13	4	0.001		0.6		96.02	95.97	1692
	■	13	4	0.001		0.6		98.91	93.55	793

Table 2: The best results obtained for each classification problem.

`inst7` unexpectedly easy. In `inst7_simil_neg` however, we have added such negative examples.

4 Experiments

Table 2 shows the best results we have obtained. The different columns describe the problem class, the method used, the topology of the network ($\Rightarrow$ #L, number units in the labels, $\Rightarrow$ #H, number of units for the representations), training parameters ($\Rightarrow \eta$, learning parameter, $\Rightarrow \mu$, momentum), the performance ($\Rightarrow$ %Tr./%Ts. the percentage of terms of the training/test set correctly classified) and the number of learning epochs needed to achieve these results. The simulations have been carried out with two to three restarts on the average case to improve the performance by slightly varied parameters. Thus it should be possible to further improve the results by parameter optimization. We have applied and compared BPTS in online ($\Rightarrow$ ■) and batch ($\Rightarrow$ □) mode on each problem instance. For-online mode, the training set was presented in the same order every epoch. Only one learning parameter η, one momentum $\mu = 0.6$ and one transfer function `tanh`

was used throughout the whole three-layered network architecture. Network weights were initialized randomly with values from $]-1.0, 1.0[$. Since we have considered two-class decision problems the output unit was taught with values $-1.0/1.0$ for negative/positive membership. The classification performance measure was computed by fixing the decision boundary at 0. The performance of BPTS is listed together with the results of a combined LRAAM-classifier (LRAAMC) architecture ($\Rightarrow \triangle$) whenever results for the LRAAMC were available [GSS95b]. It must be noted that the later approach uses a special dynamic pattern selection strategy for network training and applies different learning parameters to the LRAAM ($\Rightarrow \eta$) and the classifier ($\Rightarrow \epsilon$).

The classification results (and the low number of hidden units to achieve these results) for `lblocc1 long`, `inst4`, `inst4` and `inst7` (100% on training and test data) suggest that these problems are of very easy nature while `termocc2` may be considered as real touchstone for future approaches. Our proposed architecture supplied with BPTS leads to nearly the same or slightly better classification performance than the LRAAMC approach. But this has been achieved by a topology which uses one order of magnitude less hidden units and by a learning procedure which needs more than one order of magnitude less training epochs to converge to the same performance. Beside complexity considerations (see section 2.4.3) BPTS-batch seems to have no significant advantages over BPTS-online at the current stage of our investigations based on the given experimental results.

Why is BPTS superior to the LRAAMC approach with respect to the size of the network and the convergence of the learning procedure? Obviously, the LRAAMC has to solve the additional task of developing unique representations, i.e. to minimize the encoding-decoding error ($\Rightarrow$ % Dec.-Enc., in Table 2). This may be much more difficult than solving the classification problem. Furthermore, as the error from the classifier is not propagated recursively through the structure in the LRAAMC, the exact gradient is not computed and in contrast to our approach the representations of subterms are not optimized directly for the classification task concerning their parents.

As mentioned before, representations of structures are optimally tuned for the inference task in our approach. Figure 4 shows representations for the terms in `lblocc1 long` developed by BPTS in batch-mode with two hidden units after 80 and 108 training epochs. The longer the training with BPTS continued the more the representations of the two classes were torn apart into two distinct regions of the representation space. Secondly, the separation bounds ($\Rightarrow$ SB) (marked by five sloping lines parallel to each other, $tanh(x_1w_1 + x_2w_2 + \Theta) = \{-0.8, -0.4, 0, 0.4, 0.8\}$) are slightly turned and pushed closer to each other. After 108 epochs only one term representation could be still found at the upper left corner yielding one miss-classified structure and a classification performance of 99.61% Tr. and 100% Ts.

5 Conclusion

We have shown, that our method is really superior over previously known approaches, at least on the set of term-classification problems presented in Section 3. While these classifi-

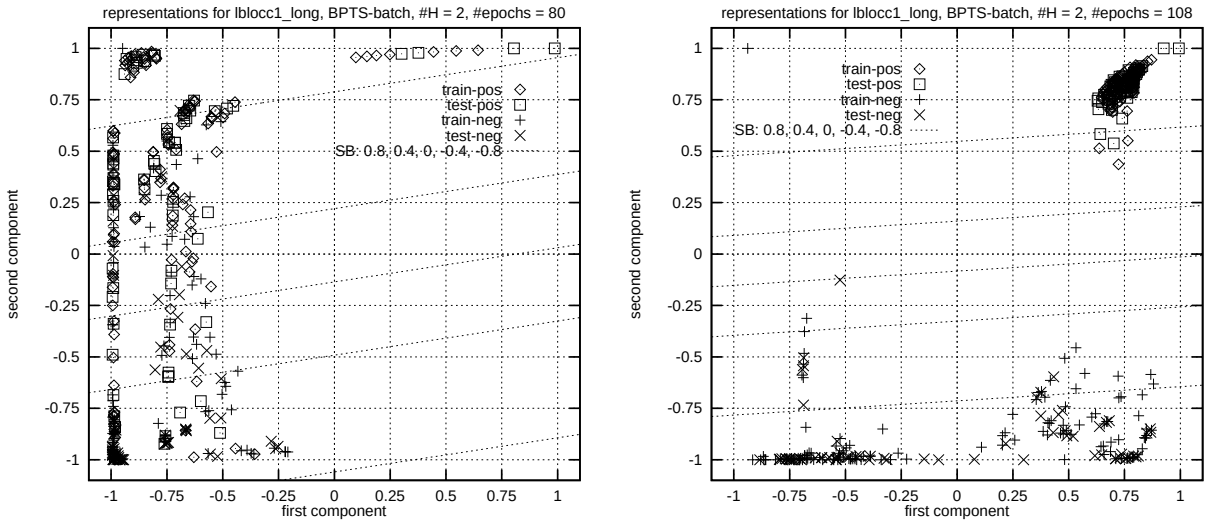


Figure 4: Representations for terms in `lblocc1_long` developed by BPTS in batch-mode with two hidden units after 80 and 108 training epochs.

cation tasks, if knowledge about their principal nature is available in advance, are trivially solved by ad hoc symbolic procedures, it is very difficult to induce their nature when only a set of positive and negative examples is given. We are convinced, that the classification problems, we are dealing with cannot be solved by symbolic learning methods with a comparable performance. Therefore we regard our and related connectionist approaches to logic-oriented learning as a very necessary supplement to classical symbolic AI that could be used in hybrid (symbolic/connectionist) systems.

We don't believe, that complicating the problems by dealing with real-valued, multi-dimensional or structured output would help us to learn more about learning methods and architectures at that moment. Our term-classification problems can become very complex. It is clear, that even undecidable term-classification problems can be constructed in an easy way. Therefore theoretical investigations concerning the expressive power of our architecture may be interesting even without regarding structured output. Furthermore we think that it is more interesting to use connectionist approaches for solving problems, that cannot be solved by classical symbolic AI like inducing the nature of our term-classification problems, than to use learning methods for deriving algorithms for problems, that have been solved efficiently by classical symbolic methods for years like e.g. unification.

We want to continue our research in the following directions: On the one hand, we want to analyze the representations generated, in order to learn more about the generalizations the networks made. Furthermore we plan to experiment with more complex architectures, e.g. additional layers in the encoder part or in the classifier. On the other hand, our experiments should also be supplemented by examples coming from real applications. The application we are currently working on is a hybrid reasoning system, in which our methods will be used to learn search control heuristics from examples [GSS95a].

Acknowledgment

This research was supported by the German Research Foundation (DFG) under grant No. Pa 268/10-1. Our thanks to Alessandro Sperduti for helpful comments and to Andreas Stolcke for providing us with his RAAM implementation.

References

[Ber91] G. Berg. Learning Recursive Phrase Structure: Combining the Strengths of PDP and X-Bar Syntax. Technical Report TR 91-5, Computer Science Department, State University of New York at Albany, 1991.

[Cad94] Vincent Cadoret. Encoding Syntactical Trees with Labelling Recursive Auto-Associative Memory. In *Proceedings of the ECAI 94*, 1994.

[Chr91] L. Chrisman. Learning Recursive Distributed Representations for Holistic Computation. *Connection Science*, (3):345–366, 1991.

[DSB92] J. B. Marshall D. S. Blank, L. A. Meeden. Exploring the symbolic/subsymbolic continuum: A case study of raam. In J. Dinsmore, editor, *The Symbolic and Connectionist Paradigms: Closing the Gap*. LEA Publishers, 1992.

[GSS95a] C. Goller, A. Sperduti, and A. Starita. Distributed Representations for Terms in Hybrid Reasoning Systems. IJCAI-95 Workshop on Connectionist-symbolic Integration: From Unified to Hybrid Approaches, 1995.

[GSS95b] C. Goller, A. Sperduti, and A. Starita. Learning Distributed Representations for the Classification of Terms. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[Nic94] L. F. Nicklasson. Structure sensitivity in connectionist models. In M.C. Mozer, P. Smolensky, D.S. Touretzky, J.L Elman, and A.S. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*. Lawrence Erlbaum Associates, 1994.

[Pol90] Jordan B. Pollack. Recursive Distributed Representations. *Artificial Intelligence*, 46(1-2), 1990.

[RGM86] D.E. Rumelhart, , G.E.Hinton, and J.L. McClelland. *Learning Internal Representations by Error Propagation*, volume 1 of *Parallel Distributed Processing*, chapter 8. MIT Press, Cambridge/London, 1986.

[Spe94] A. Sperduti. Encoding of Labeled Graphs by Labeling RAAM. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *NIPS 6*, pages 1125–1132. 1994.

[SW92] A. Stolcke and D. Wu. Tree Matching with Recursive Distributed Representations. Technical Report TR-92-025, International Computer Science Institute, Berkeley, California, 1992.

[Wer90] Paul J. Werbos. Backpropagation Trough Time: What it Does and How to Do it. *Proceedings of the IEEE*, 78(10):1550–1560, October 1990.