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

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

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) [7], 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 [3, 6] and also for more complex tasks even with structured output [2, 4]. Our approach, however, is different. We present a simple architecture together with a novel supervised learning scheme that we call *backpropagation through structure* (BPTS) in analogy to *backpropagation through time* for recurrent networks [10]. It allows us to generate distributed representations for symbolic structures which are exclusively tuned for the intended task.

The next two sections describe the proposed architecture and the corresponding learning scheme in detail. Section 4 presents experimental results on a set of basic classification tasks (detectors) on logical terms. 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 [6].

2. Architecture

2.1. Labeling RAAM

The architecture we use is inspired by the Labeling RAAM (LRAAM) [8], an extension of the RAAM [7] model that learns fixed-width distributed representations for *labeled* variable-sized recursive data structures. As the 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, right).

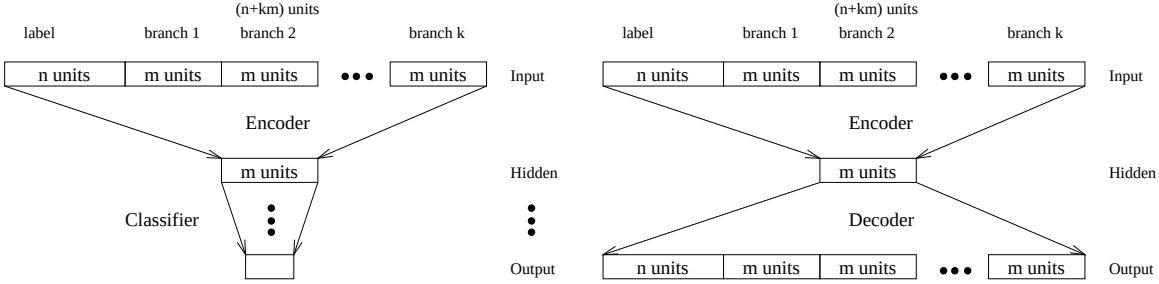

Fig. 1: The Folding Architecture (left side) and the standard LRAAM (right side).

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 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.

2.2. Folding Architecture

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 3) could be easily extended to handle inference tasks of more complex nature.

Figure 1 (left side) shows the *folding 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. For the classification of a new structure, the network is virtually unfolded (just as in Figure 3) to compute the structure's representation, which is then used as input for the classifier. This would be done in the same way for LRAAM-derived representations. However, we use the virtual unfolding of the network also for the learning phase and propagate the classification error through the whole virtually unfolded network, instead of using a decoder part for finding unique representations. The idea behind this is, that for many inference tasks unique representations are not necessary as long as the information needed for the inference task is represented properly. Our learning scheme is completely supervised without moving targets.

3. Backpropagation Through Structure

The principle idea of using recursive backpropagation-like learning procedures for symbolic tree-processing has been mentioned first in [1, 9]. We assume in the following the reader to be familiar with the standard backpropagation algorithm (BP) and its variant *backpropagation through time* (BPTT) [10] that is used to train recurrent network models. For the sake of brevity we will only be able to give a sketch of the underlying principles of our approach. Refer to [5] for a detailed discussion and formal specification of the given algorithms.

Let us first have a closer look on the representation of structures we want to process. All kinds of recursive symbolic data structures we aim at can be mapped onto labeled directed acyclic graphs (DAGs). For example choosing a DAG-representation for a set of logical terms (see Figure 2)– that allows to represent different occurrences of a (sub-)structure only as one node – may lead to a considerable (even exponential) reduction of space complexity. But this complexity reduction also holds for the time complexity of training. For the LRAAM training as well as for BPTS (as we will show in the following two sections) the number of forward and backward phases per epoch is linear to the number of nodes in the DAG-representation of the training set.

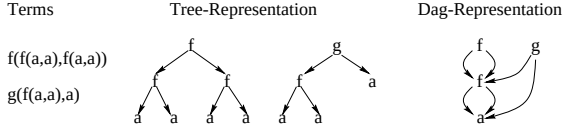

Fig. 2: Tree and DAG representation of a set of terms.

3.1. BPTS for Trees

For reasons of simplicity we first restrict our considerations to tree-like structures. In the *forward phase* the encoder (Figure 1, left) is used to compute a representation for a given tree in the same way as in the plain LRAAM. The following metaphor helps us to explain the *backward phase*. Imagine the encoder of the folding architecture is virtually unfolded (with copied weights) according to the tree structure (see Figure 3). Now the error passed from the classifier to the hidden layer is propagated through the unfolded encoder network.

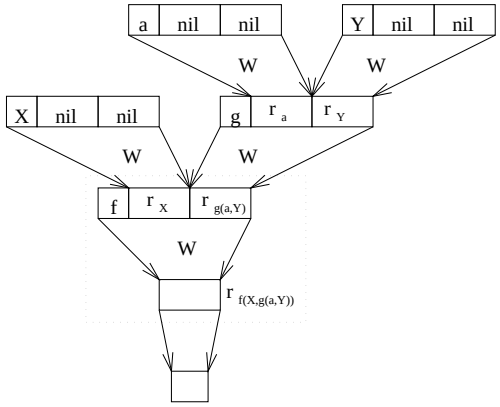

Fig. 3: The Encoder unfolded by the structure $f(X, g(a, Y))$.

With a similar argument as for BPTT [10] we can see, that the exact gradient is computed. 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).

3.2. 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 $\vec{\delta_{t_i}} \neq \vec{\delta_{t_j}}$

problem name	symbols used	rules for 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)
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 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)
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)

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

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. 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 above (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.

4. Experiments

The characteristics of each term classification problem we used to evaluate our approach are summarized in Table 1. The set of problems is an extension of the one used in [6]. They range from the detection of a certain subterm to the satisfaction of a specific unification pattern. The second column reports the set of symbols (with associated arity) compounding the terms, 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 3), and the last column the

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
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

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

maximum depth[1] 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.

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. 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. 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 the folding architecture with BPTS is listed together with the results of a combined LRAAM-classifier[2] (LRAAMC) architecture ($\Rightarrow$ △) whenever results for the LRAAMC were available [6].

A more detailed discussion of the problem classes viewed in relation to the results can be found in [5]. Our folding 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

[1] 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.

[2] LRAAMC uses a special dynamic pattern selection strategy for network training and applies different learning parameters to the LRAAM ($\Rightarrow \eta$) and the classifier ($\Rightarrow \epsilon$).

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 3.2) 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 the folding architecture together with BPTS superior to the LRAAMC? 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 and sometimes contradictory to the classification task. 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.

5. Conclusion

By using the folding architecture together with BPTS the encoding of structures is melted with further inference into one single process. We have shown that our method is really superior over previously known approaches, at least on the set of term-classification problems presented here.

In order learn more about the generalization capabilities of the architecture it will be important to analyze the generated representations. We plan to experiment with more complex architectures (e.g. additional layers in the encoder or in the classifier) and with examples coming from "real" applications. One we are currently working on is a hybrid (symbolic/connectionist) reasoning system, in which our methods will be used to learn search control heuristics from examples.

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

[1] G. Berg, "Connectionist Parser with Recursive Sentence Structure and Lexical Disambiguation," in *Proceedings of the AAAI 92*, pp. 32–37, 1992.

[2] D. S. Blank, L. A. Meeden, and J. B. Marshall, "Exploring the Symbolic/Subsymbolic Continuum: A Case Study of RAAM," in *The Symbolic and Connectionist Paradigms: Closing the Gap*, (J. Dinsmore, ed.), LEA Publishers, 1992.

[3] V. Cadoret, "Encoding Syntactical Trees with Labelling Recursive Auto-Associative Memory," in *Proceedings of the ECAI 94*, pp. 555–559, 1994.

[4] L. Chrisman, "Learning Recursive Distributed Representations for Holistic Computation," *Connection Science*, no. 3, pp. 345–366, 1991.

[5] C. Goller and A. Küchler, "Learning Task-Dependent Distributed Representations by Backpropagation Through Structure," AR-report AR-95-02, Institut für Informatik, Technische Universität München, 1995.

[6] C. Goller, A. Sperduti, and A. Starita, "Learning Distributed Representations for the Classification of Terms," in *Proceedings of the IJCAI 95*, pp. 509–515, 1995.

[7] J. B. Pollack, "Recursive Distributed Representations," *Artificial Intelligence*, vol. 46, no. 1-2, 1990.

[8] A. Sperduti, "Encoding of Labeled Graphs by Labeling RAAM," in *NIPS 6*, (J. D. Cowan, G. Tesauro, and J. Alspector, eds.), pp. 1125–1132, 1994.

[9] A. Stolcke and D. Wu, "Tree Matching with Recursive Distributed Representations," Tech. Rep. TR-92-025, International Computer Science Institute, Berkeley, California, 1992.

[10] P. J. Werbos, "Backpropagation Trough Time: What it Does and How to Do it," *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, Oct. 1990.