Inductive Learning in Symbolic Domains Using Structure-Driven Recurrent Neural Networks*

Andreas Küchler[1] and Christoph Goller[2]

[1] Department of Neural Information Processing
Computer Science, University of Ulm, D-89069 Ulm, Germany
[2] Automated Reasoning Group
Computer Science Institute, TU Munich, D-80290, München, Germany

Abstract. While neural networks are widely applied as powerful tools for inductive learning of mappings in domains of fixed-length feature vectors, there are still expressed principled doubts whether the domain can be enlarged to structured objects of arbitrary shape (like 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. Labeled directed acyclic graphs are the most general structures that can be handled. The processing in this architecture is driven by the inherent recursive nature of the given structures. Our approach can be viewed as a generalization of the well-known discrete-time, continuous-space recurrent neural networks and their corresponding training procedures. We give first results from experiments with inductive learning tasks consisting in the classification of logical terms. These range from the detection of a certain subterm to the satisfaction of complex matching constraints and also capture certain concepts of syntactical variables.

1 Introduction

In almost all fields of life people and systems assisting them have to deal with structured objects. Annotated graphs (trees, terms, diagrams) are a universal formalism for representing and modeling structural relations between objects (hierarchies, causal, temporal and spatial dependencies, explanations, etc.) and are successfully applied in fields like medical and technical diagnosis, molecular biology and chemistry, software engineering, geometrical reasoning, speech, language and text processing. Faced with the growing mass and the structural complexity of data there is an increasing need to have an automated tool which inductively "learns" to classify, recognize and evaluate structured objects or to discover regularities between them.

While neural networks are successfully applied as powerful inductive learning devices when dealing with numerical feature vectors of a fixed length, there are still expressed doubts [6] whether those numerical models in principle can be used

* This research was supported by the German Research Foundation (DFG) under grant No. Pa 268/10-1 and by the EC (ESPRIT BRP MIX-9119)

in a symbolic domain. Since the late eighties there have been several publications demonstrating that these limitations can be partially broken [11, 4, 2, 1, 13, 3, 8].

Recently a simple neural network architecture (the *folding architecture*) together with a novel supervised learning scheme, *backpropagation through structure* (BPTS) was presented which is capable of processing labeled directed acyclic graphs (LDAG) of arbitrary size [7]. Here we will extend this model, give a formal description and discuss whether and how it could be utilized for inductive learning tasks in symbolic domains. Our conjectures are supported by promising results from experiments on inductive classification tasks.

This paper is organized as follows. First the meaning of the terms *inductive learning* and *symbolic domain* have to be related to our context and fixed by some formal definitions. Section 3 gives an introduction to the folding architecture by separating statical from dynamical aspects. BPTS, a gradient-descent supervised learning procedure for the folding architecture, is derived by generalizing the well-known BPTT approach to DAG-processing in Section 4. First experiments and results in applying the proposed method to inductive classification tasks on logical terms are reported in Section 5. Section 6 relates our approach to some other numerical and symbolic models. We conclude with some remarks about the justification of having different coexistent models in the field of inductive learning in symbolic domains.

2 Preliminaries

2.1 The Intended Inductive Learning Tasks

The class of inductive learning tasks on symbolic domains (ILS) we consider here is defined as a tuple ILS $= (\Xi, \mathcal{P})$, where Ξ is an unknown function of the type $\Xi : S \rightarrow I\!R^q$ with $q \in I\!N, S \subseteq \mathcal{S}$ and S is a (possibly infinite) subset of the general symbolic domain $\mathcal{S}$ (see Section 2.2). $\mathcal{P}$ is a large but finite set of examples partially describing Ξ, i.e. $\mathcal{P} = \{(s_1, \mathbf{t}_1), (s_2, \mathbf{t}_2), \ldots, (s_p, \mathbf{t}_p)\}$ where $\Xi(s_i) = \mathbf{t}_i, s_i \in S, \mathbf{t}_i \in I\!R^q$ and $i, p, q \in I\!N, 1 \leq i \leq p$. The learning task is to infer an *approximation* Ξ_A (as good as possible) to the unknown function Ξ based on the given finite example set $\mathcal{P}$ only. An ILS is defined without any explicit theory or knowledge about the function Ξ.

A special case is the function Ξ restricted to $\Xi : S \rightarrow \{c_1, c_2, \ldots, c_p\}$ with $p \in I\!N$. This is a description of inductive p-class *classification* tasks where the c_i ($i \in I\!N, 1 \leq i \leq p$) are the corresponding class labels. The objective here is to infer a classifier Ξ_C with a high accuracy, i.e. with a high probability of correctly classifying a randomly selected instance $(s, \Xi(s)), s \in S$.

In this article we focus on ILS as explained above, but it is clear that in principle Ξ could be extended to structure-structure mappings, i.e. $\Xi : S^1 \rightarrow S^2$ where $S^1, S^2 \subseteq \mathcal{S}$. However, special care has to be taken about an adequate definition of approximation in the context of a corresponding inductive learning task.

The next section will give a detailed specification of the symbolic domain $\mathcal{S}$.

2.2 The Symbolic Domain

Graphs have been proven as a universal formalism for representing and modeling different kinds of knowledge (hierarchies, causal, temporal and spatial dependencies, explanations, arbitrary relations between objects, etc.) and are applied in nearly all scientific fields.

In this work we deal with an important subclass of general graphs. A *labeled directed graph* G is a triple (V, E, λ), where V is a finite set, E is a binary relation on V and $\lambda : V \rightarrow \Sigma$ a labeling function with the range of a finite alphabet Σ. The set V is usually called the *vertex set* of G, and its elements are called *vertices* or *nodes*. The set E is called the *edge set* of G, and its elements are called *edges*. A *labeled directed acyclic graph* (LDAG) is a cycle-free labeled directed graph, i.e. there exists a topological sort of the nodes. The *outdegree* (*indegree*) of a node v is the number of outgoing (ingoing) edges, $|\{x \mid (v, x) \in E\}|$ ($|\{x \mid (x, v) \in E\}|$). The outdegree of G is given by the maximum outdegree of all nodes $v \in V$.

A *topological sort* of a DAG $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in that ordering. Note that topological sorts can be performed in time complexity of $\Theta(V + E)$.

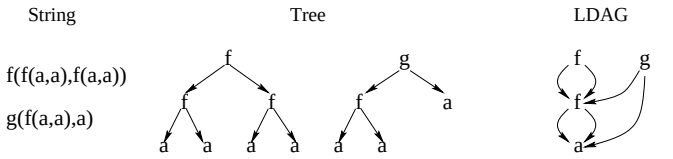

Fig. 1. String, Tree and LDAG representation of a set of terms.

A *rooted* LDAG (RLDAG) is a DAG where there is exactly one node with indegree zero – the *root node*.[3] In our case the *symbolic domain* $\mathcal{S}$ is defined by the universe of all possible RLDAGs. For example *terms* over a given signature – the syntactical elements of every programming language – can be transformed into compact RLDAG representations. Each subterm of a term is represented by a vertex such that multiple occurrences of identical subterms are represented by exactly one vertex[4]. Edges are drawn from each term (subterm) representation to the representations of its argument terms (subterms). A *set of terms* can be packed into an augmented LDAG $G = (R, V, E, \lambda)$ by applying this procedure also to identical subterms occurring in different terms (see Figure 1) and maintaining a set R ($R \subseteq V$) of root nodes coming from each individual term RLDAG representation.

[3] Every LDAG can be uniquely completed to a RLDAG by introducing a new node r and inserting edges from r to all other nodes with indegree zero.

[4] This can lead to an exponential reduction of the number of vertices and edges, needed to represent the term. The whole transformation can be done in time $O(n \log(n))$ with respect to the original size of the term.

3 The Folding Architecture

The objective is to develop a neural architecture which can be utilized for solving a given ILS $= (\Xi, \mathcal{P})$ where $\Xi : S \rightarrow I\!R^q$, $q \in I\!N$, $|\mathcal{P}| = p$ and S is a given subset of the universe of RLDAGs. The principled idea is to combine a component for encoding elements of S into suitable connectionist distributed representations with a component to compute (approximation or classification) tasks on distributed representations.

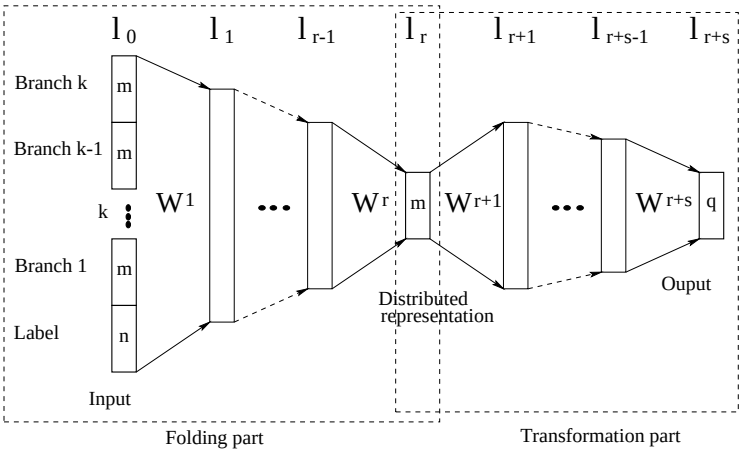

Fig. 2. The generic folding architecture

3.1 The Static View

Our generic *folding* architecture is layered and the static view is that of a specially scaled multilayer feedforward network (Figure 2). The first $r+1$ layers $\{l_0, \ldots, l_r\}$ constitute the *folding* part, the next layers $\{l_r, \ldots, l_{r+s}\}$ including layer l_r the *transformation* part, where $s, r \geq 1$. All layers are fully-connected with real-valued weights in a feedforward manner and each unit in the layers is provided with a sigmoid transfer function.

The number $q \in I\!N$ of units (neurons) in the output layer l_{r+s} is task-specific as well as the maximum outdegree k which is defined by the domain S. The number of *hidden* layers and the number of neurons in each layer concerning the folding ($\{l_1, \ldots, l_{r-1}\}$) and transformation ($\{l_{r+1}, \ldots, l_{r+s-1}\}$) part is not predefined by the given ILS. Neither is m, the dimension of the representation layer l_r. The input layer l_0 is constituted by $n + k \cdot m$ units, n holding the representation for the vertex labels and k times m units provided for distributed representations of DAGs. The weight matrix corresponding to the layer l_j is denoted by W_j with $1 \leq j \leq r+s$, each unit is provided with an individual bias.

3.2 Processing Dynamics

There are no explicit feedback connections in the folding architecture. The recurrent processing is completely driven by the inherent recursive structure of the presented data from the symbolic domain. Informally speaking the folding part is used to "fold" a given structure $s \in S$ into a distributed representation (in $\mathbb{R}^m$). This is achieved by recursively setting the previously computed representations of the direct substructures together with the coding of the associated label at the input layer (Equation 3) and propagating them through the folding part (Equation 2). This process starts at the leaves (by using the label coding and k times the coding of the "empty structure") and ends up with a representation of the whole structure which is then pushed through the whole transformation part (Equation 1). An evaluation of the whole structure can be taken from the output layer.

We will now define the processing dynamics formally. For reasons of simplicity let $s \in S$ be a labeled tree, i.e. a RLDAG $G = (V, E, \lambda)$ where each node in V has an indegree of at most 1 (e.g. see Figure 1). Further let $t \in V$ be a node of s, C a coding function that maps symbolic labels to numeric codes ($C : \Sigma \rightarrow \mathbb{R}^n$) and $nil \in \mathbb{R}^m$ a special representation for the empty DAG. The output of neuron[5] i in layer l at recursion stage t is written as $o_i^{(l)}(t)$. $\theta_i^{(l)}$ denotes the bias associated with neuron i at layer l, $w_{ij}^{(l+1)}$ the weight of the connection between neuron i of the layer l and neuron j of layer $l+1$ and f is a (differentiable) sigmoid function. The continuous-space network dynamics are described by the following equations.

$$o_j^{(l+1)}(t) = f(net_j^{(l+1)}(t)) = f\left(\sum_i o_i^{(l)}(t)\, w_{ij}^{(l+1)} + \theta_j^{(l+1)}\right)$$

$$\text{for} \quad r \leq l < r+s, \quad t \text{ is a root node} \tag{1}$$

$$\text{for} \quad 0 \leq l < r, \quad \text{any } t \tag{2}$$

This means that the folding part for every node t is processed in a standard feedforward manner. However the transformation part is only involved for root nodes.

The outputs $o_i^{(0)}(t)$ are composed of the numeric codes for the labels and the previously computed representations of the dirct substructures. Let V_t be the set of direct successor nodes of t, i.e. $V_t = \{t_0, t_1, \ldots, t_{d-1}\} = \{v \mid (t, v) \in E\}$ and $0 \leq d \leq k$. The i-th component of a vector is denoted by $[\cdot]_i$. The outputs $o_i^{(r)}(t_x)$ produced by successor nodes t_x of t with $0 \leq x < d$ at layer r are fed back to layer 0.

$$o_i^{(0)}(t) = \begin{cases} [C(\lambda(t))]_i & : 0 \leq i < n \\ o^{(r)}_{(i-n)\,\mathbf{mod}\,m}\left(t_{(i-n+m)\,\mathbf{div}\,m}\right) & : n \leq i < n+dm \\ [nil]_{(i-n)\,\mathbf{mod}\,m} & : n+dm \leq i < n+km \end{cases} \tag{3}$$

[5] The first neuron will be indexed with 0.

4 Backpropagation Through Structure

The principled idea of using recursive backpropagation-like learning procedures for tree-processing has been mentioned first in [1], recently worked out and generalized to DAGs by [7]. We assume in the following the reader to be familiar with the standard backpropagation algorithm (BP) and its variant *backpropagation through time* (BPTT) [15]. For the sake of brevity we will only be able to give a sketch of the underlying principles of our approach called *backpropagation through structure* (BPTS).

4.1 Computing the Gradient Information

Given an ILS $= (\Xi, \mathcal{P})$ where $\Xi : S \rightarrow I\!R^q$, $q \in I\!N$, $\mathcal{P} = \{(s_1, \mathbf{t}_1), \ldots, (s_p, \mathbf{t}_p)\}$ and S consisting of trees only, let *root* denote the function mapping structures to their root nodes. In order to develop a supervised learning procedure an appropriate error measure[6] E in the space of parameters $W^1, \ldots, W^{r+s}$ w.r.t. the training set $\mathcal{P}$ has to be defined first.

$$E = \sum_{i=1}^{p} \sum_{j=0}^{q-1} \frac{1}{2} \left([\mathbf{t}_i]_j - o_j^{(r+s)}(root(s_i)) \right)^2 \; ; \quad \Delta w_{ij}^{(l)} = -\eta \frac{\partial E}{\partial w_{ij}^{(l)}} \text{ for } 1 \leq l \leq r+s \quad (4)$$

Obviously, the folding architecture gives a good approximation to Ξ w.r.t the training examples if the mean squared error E (Equation 4, left) is minimized in the weight space. BPTS is a simple gradient-descent procedure following the weight update rule given in Equation 4 (right), where η is the learning rate[7].

The following metaphor helps us to explain the derivation and computation of the weight update rule. Imagine the folding part of the architecture is *virtually unfolded* (with copied weights) according to a given tree structure $s \in S$ (see Figure 3). The resulting feedforward network exhibits a computation process that is equivalent to the dynamics described in section 3.2. Thus, BPTS can be derived and formulated in analogy to the standard BP procedure. We keep the notations introduced (in Section 3.2) and define f' as the first derivative of f and $\delta_j^{(l)}(t)$ as the partial error (which is contributed by a node t) propagated back from the output layer to neuron j in layer l.
Equation 5 shows how the error for the output layer of root nodes is computed.

$$\delta_j^{(r+s)}(t) = f'(net_j^{(r+s)}(t)) \left([\mathbf{t}_i]_j - o_j^{(r+s)}(t) \right) \; : \quad t = root(s_i), (s_i, \mathbf{t}_i) \in \mathcal{P} \quad (5)$$

Equation 6 and 7 describe, how the error is propagated through the transformation and the folding layer. Again, the transformation layer is involved for root nodes only.

$$\delta_j^{(l)}(t) = f'(net_j^{(l)}(t)) \left(\sum_k \delta_k^{(l+1)}(t)\, w_{jk}^{(l+1)} \right)$$

$$\text{for} \quad r \leq l < r+s, \quad t \text{ is a root node} \quad (6)$$

$$\text{for} \quad 0 \leq l < r, \quad \text{any } t \quad (7)$$

[6] Note the only constraint that has to be obeyed is that E has to be differentiable.
[7] In our experiments we also added a momentum term μ.

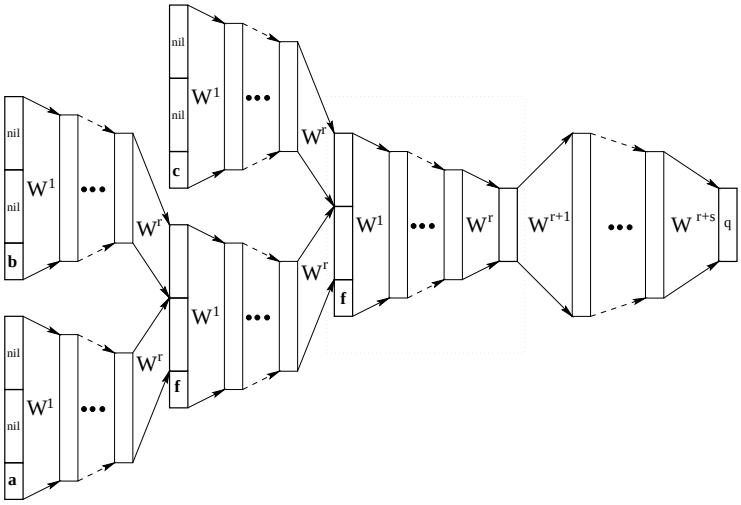

Fig. 3. Virtual unfolding, driven by the tree structure $f(f(a,b),c)$.

$\delta_j^{(0)}(t)$ has to be computed for $n \leq j < n+dm$ only, with d denoting the number of immediate successors nodes of t as in Section 3.2.
Equation 8 shows how an error term is recursively distributed among the direct successors of a node.

$$\delta_j^{(r)}(t) = \delta_{(n+xm+j)}^{(0)}(t') \;:\quad t \text{ is the } x\text{-th successor of } t'. \tag{8}$$

After a presentation of a set T of nodes occurring in structures of $\mathcal{P}$, all changes computed for each node $t \in T$ can be summed up according to the following equations.

$$\Delta w_{ij}^{(l)} = \eta \sum_{t \in T} o_i^{(l-1)}(t) \cdot \delta_j^{(l)}(t)\,, \quad \Delta\theta_j^{(l)} = \eta \sum_{t \in T} \delta_j^{(l)}(t) \quad (\text{for } 1 \leq l \leq r+s) \tag{9}$$

4.2 Training Schedule for DAGs

We speak of BPTS applied in *batch* mode if a weight update (according to Equation 9) is done after the presentation of the whole training set $\mathcal{P}$. Let us assume that two substructures starting at the nodes t_a and t_b are identical. This means $\forall_{j,l} net_j^{(l)}(t_a) = net_j^{(l)}(t_b)$, $o_j^{(l)}(t_a) = o_j^{(l)}(t_b)$. We notice (by Equation 9) that δ's stemming from different predecessors can be simply summed up

$$\Delta w_{ij}^{(l)} = \eta \sum_t o_i^{(l-1)}(t_a)\cdot\delta_j^{(l)}(t_a) + o_i^{(l-1)}(t_b)\cdot\delta_j^{(l)}(t_b) + \ldots = \eta \sum_t o_i^{(l-1)}(t_a)\cdot(\delta_j^{(l)}(t_a) + \delta_j^{(l)}(t_b)) + \ldots$$

and this sum can be propagated back into identical substructures (by Equations 7 and 8):

$$\delta_j^{(l)}(t_a) + \delta_j^{(l)}(t_b) = f'(net_j^{(l)}(t_a)) \sum_k \left(\delta_k^{(l+1)}(t_a) + \delta_k^{(l+1)}(t_b)\right) w_{jk}^{(l+1)} \tag{10}$$

This observation enables an efficient implementation of BPTS for DAGs. First, in a preprocessing step all structures $\{s \mid (s, \Xi(s)) \in \mathcal{P}\}$ (trees or DAGs) of the whole training set are packed into one single augmented DAG $G = (R, V, E, \lambda)$ (by representing each identical substructure as exactly one node, see Section 2.2). Let O_T be a topological ordering on the nodes V imposed by G.

A *training epoch* is defined by a *forward* (computing $o(t)$ according to the Equations 1–3), a *backward* phase (computing δ according to the Equations 5–8) and an *update* of the weights (according to the Equation 9). The forward phase starts with nodes having an outdegree of zero (leaves) and proceeds according to the reverse (O_T, descending) ordering – ensuring that distributed representations for identical substructures have to be computed only once. The backward phase follows the topological ordering O_T beginning at the root nodes of G. In this way δ's passed over different edges to a node t are summed up before t is processed. Thus, each node of G has to be processed only twice per epoch. The training stops after a predefined number of epochs, when the total error E is below a certain small threshold or when other performance criterions on data not used for training are satisfied.

The time complexity of one BPTS epoch in batch mode on $G = (R, V, E, \lambda)$ is characterized (a pessimistic estimation) by $O(|V| \cdot n_f^2 + |R| \cdot n_t^2)$, where n_f (n_t) is the total number of neurons in the folding (transformation) part of the architecture. The overall space complexity is epoch-independent and is expressed by $O(n_w + |V| \cdot n_f)$, where n_w is the total number of weights. Therefore, the compact DAG-representation for terms can lead to an exponential reduction of the overall space complexity and the time complexity of a BPTS epoch (see Section 2.2).

5 Experiments

5.1 Term Classification Tasks

For a first evaluation of the folding architecture and BPTS we used a set of 2-class[8] classification problems on logical terms. In most papers about ILS with connectionist methods (see e.g. [3, 4, 2]) linguistic applications have been chosen for tests and evaluation. We, however, identified some basic abstract problem classes and generated instances of these classes with different complexity for our experiments. These problem classes are no more than a first step towards a more detailed evaluation for ILS. It is clear, that many other interesting classes have to be investigated.

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 variables (see below). Whenever we give example terms or pattern terms in the following, we will use Prolog notation, that is, lower case letters represent function symbols and constants, upper case letters represent all-quantified logical variables. We discriminate between four different

[8] We will always talk about the positive and negative class in this context.

problem classes named *label-occurrence*, *term-occurrence*, *instance* and *instance-occurrence*. For label-occurrence problems the difference between terms from the negative and the positive class concerns the occurrence of a special label (e.g. the constant c occurs in $f(t(a, i(b), j(i(c))))$, but not in $f(t(a, i(b), j(i(d))))$). A term-occurrence problem is specified by a special pattern-term (e.g. $i(a)$) that has to occur as subterm in all positive examples, but never occurs in negative ones. For an instance problem all positive terms are instances of a special pattern-term containing all-quantified variables and negative examples mustn't have this property (e.g. $f(i(a), f(a, b))$ is an instance of the pattern term $f(X, f(a, Y))$, however $f(a, f(b, a))$ is not). If all terms from the positive class contain at least one instance of a special pattern term and all terms from the negative class do not, we speak of an instance-occurrence problem. In the context of instance and instance occurrence problems pattern-terms with multiple occurrences of variables demand special attention. A classifier for problems with such pattern-terms has to compare arbitrary subterms corresponding to different occurrences of variables, what may make these problems very difficult. Besides the four mentioned problem classes, we also considered disjunctive and conjunctive combinations, e.g. the occurrence of at least one out of a set of pattern terms (disjunctive) or the occurrence of all (conjunctive) of them as characteristic property for terms from the positive class.

The characteristics of each problem used for our experiments are summarized in Table 1. The first column of the table reports the name of the problem, the second one the signature (terms were composed of), and the third column shows the rule(s) used to generate the positive examples. The fourth column reports the number of terms in the training and test set respectively and the last column the maximum depth[9] of terms in the positive and negative class. For each problem about the same number of positive and negative examples was given. Both positive and negative examples have been generated randomly. Training and test sets are disjoint and have been generated by the same algorithm.

5.2 Results

Table 2 shows the best results we have obtained by experiments with the folding architecture and BPTS for each problem instance introduced in section 5.1, Table 1. The first column reports the problem name and the second one the topology of the network ($\Rightarrow n$, number units in the labels, $\Rightarrow m$, number of units for the representations). All problems were solved by a three-layer folding architecture with one single output unit ($r = 1, s = 1, q = 1$) with the exception of problem `termocc2` which was solved with an additional layer in the folding part ($r = 2$, and l_1 containing 6 units). The third column gives the training parameters ($\Rightarrow \eta$, learning parameter, $\Rightarrow \mu$, momentum) and the fourth column the steepness β of the sigmoid transfer function. Learning parameter, momentum and steepness were always identical for the whole network. The last three

[9] 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.

Problem	Symbols	Positive Examples.	#terms (train,test)	depth (pos.,neg.)
lblocc1 long	f/2 i/1 a/0 b/0 c/0	no occurrence of label c	(259,141)	(5,5)
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)	(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)	(7,7)
termocc4 me	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	the (sub)terms i(a) and f(b,c) occur somewhere	(500,500)	(5,5)
termocc4 me long	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	the (sub)terms i(a) and f(b,c) occur somewhere	(500,500)	(9,9)
termocc5 short	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	the (sub)terms j(i(g(a,b))) or f(j(b),i(c)) or t(b,i(c),d) occur somewhere	(1200,1200)	(5,5)
termocc5 me	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	the (sub)terms j(i(g(a,b))) or f(j(b),i(c)) or t(b,i(c),d) occur somewhere	(3000,3000)	(9,9)
inst1 long	f/2 a/0 b/0 c/0	instances of f(X,X)	(202,98)	(6,6)
inst4 long	f/2 a/0 b/0 c/0	instances of f(X,f(a,Y))	(290,110)	(7,6)
inst8	f/2 a/0 b/0 c/0	instances of f(X,f(a,X))	(300,300)	(7,7)
inst9	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	instances of t(i(X),g(Y,b),b)	(600,600)	(9,9)
inst9 sn	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	instances of t(i(X),g(Y,b),b)	(606,606)	(9,9)
inst7 sn	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)	(600,600)	(9,9)
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)	(7,6)
instocc2	t/3 f/2 g/2 i/1 j/1 a/0 b/0 c/0 d/0	an instance of t(i(X),g(Y,b),b) occurs somewhere	(500,500)	(9,9)

Table 1. A set of classification problems involving logical terms.

columns report the classification 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. As sigmoid transfer function `tanh` was used. Network weights were initialized randomly with values from $]-1.0, 1.0[$. Since we have considered 2-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 simulations have been carried out with several restarts (no more than ten) for each problem to improve the performance by slightly varied parameters and changes in the network topology. It should be possible to further improve the results. We can only present very first results and the number of experiments does not allow to infer statistically relevant propositions, but the results indicate that inductive learning and generalization on a symbolic domain can be done with neural networks. To give an impression of the performance of the learning we want to note, that one epoch took from less than 1 to 20 seconds on a SUN SPARC 10 workstation depending on the network topology and the number of examples in the training set.

Problem	Topology		Learning Par.		Steepness	% Tr.	% Ts.	#epochs
	n	m	η	μ	β			
lblocc1 long	8	2	0.001	0.6	0.5	99.61	100	108
termocc1 very long	8	6	0.001	0.6	0.5	99.64	98.33	3522
termocc2	13	4	0.0003	0.2	1	98.25	99	2859
termocc4 me	10	5	0.0005	0.3	1	99	92.4	7100
termocc4 me long	10	5	0.0005	0.3	1	90.2	85.2	1136
termocc5 short	10	6	0.0003	0.2	1	98.083	93.5	8733
termocc5 me	10	6	0.0002	0.2	1	92.367	91.7	1073
inst1 long	6	3	0.005	0.6	0.5	95.56	92.86	4250
inst4 long	6	3	0.003	0.6	0.5	100	100	150
inst8	6	5	0.005	0.5	1	98	97.667	201
inst9	10	4	0.002	0.5	1	100	99.833	40
inst9 sn	10	5	0.001	0.5	1	98.185	99.175	1045
inst7 sn	10	6	0.001	0.5	1	99	95.167	1997
inst5	13	3	0.001	0.6	0.5	98.88	94.70	2155
instocc2	10	5	0.001	0.5	1	98	98	233

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

Some interesting details should be mentioned: For `termocc1 very long` and `termocc2` we nearly got the same classification performance. However for `termocc2` an additional layer in the folding part was needed and it was impossible to get similar good results without it. Both problems are defined by the same pattern-terms, but the classification seems to become more difficult with the bigger signature of `termocc2`. However the problems `termocc4 me`, `termocc4 me long`, `termocc5 short` and `termocc5 me` are solvable without hidden layers in the folding part, though based on the same complex signature as `termocc2`. With the problem `inst1 long` we show, that a very simple case of multiple variable occurrence is solvable, however a perfect solution (100 % correct classification on the test set) was not possible. This is the case for `inst8` too, while `inst4 long` without multiple occurrence of a variable is solved perfectly. For examples with many different symbols and complex patterns defining the positive class, randomly generating negative examples leads to terms not very similar to the positive ones. This may be the reason, why e.g. `inst9` is learned so quickly. For `inst9 sn` we have generated special negative examples, differing from positive ones only slightly (e.g. by one or two symbols) and added them to the training and test set. This really had influence on the learning. One more unit for the representation layer and many more epochs of training were needed to achieve a classification performance comparable to that of `inst9`. The pattern-term defining `inst7 sn` is very similar to the one from `inst9 sn` and `inst9`. However the variable X occurs twice. Similar to `inst9 sn` we added negative examples, very similar to positive ones (e.g. instances of $t(i(X), g(Y, b), b$ with $X \neq Y$) to the negative class in the training and test set. No significant difference to `inst9 sn` can be noticed concerning the learning and classification performance.

6 Related Work

6.1 The (L)RAAM Model

The Labeling Recursive Autoassociative Memory (LRAAM) [13, 11] is a symmetrical three-layer feedforward network used to generate (unique) fixed-width distributed representations for labeled directed graphs. The architecture of the LRAAM can be imagined by mirroring the folding part $r = 1$ of our architecture at the representation layer l_r (Figure 2).

The objective of an (L)RAAM is to obtain a compressed representation (hidden layer activation) for a node by training (with the Backpropagation procedure) recursive autoassociations of the label coding together with representations of the direct successor nodes. There have been several attempts to solve inductive learning (classification) tasks on tree domains by using RAAM models. Distributed representations for trees are generated by the RAAM, "frozen" and then fed into other network components which are trained in a supervised way to solve the given task [3, 2]. Others combine the encoding task of the LRAAM with the inductive learning task of subsequent network modules by imposing a strong interaction on the corresponding training procedures [4, 8].

Our approach – the folding architecture trained in a supervised manner – takes a different point of view. The encoding of structures is melted with further transformations into one single process. The objective is not to generate unique representations for structures but to obtain representations that are exclusively tuned for the given ILS. First experimental comparisons give strong evidence that our method is really superior to the combined LRAAM model when dealing with inductive (classification) tasks [7].

6.2 Discrete-Time Continuous-Space Recurrent Neural Networks

Discrete-time continuous-space recurrent neural networks are successfully applied to robust inductive sequence processing, e.g. inductive grammatical inference, learning of dynamical systems behaviour, time series prediction [5]. There is a large variety of architectures (for a classification and experimental comparison see [9]). A common property among the members of this discrete-time recurrent network "zoo" is that neurons are allowed to receive directly (or indirectly, communicated by other neurons) feedback from their own activations with an arbitrary but a priori fixed time delay of discrete time steps.

Our approach – *structure-driven* continuous-space recurrent neural networks – may be viewed as a generalization of their time-discrete relations, since the delay of feedback is driven by the recursive nature of the presented data, i.e. activations can be delayed arbitrary time steps before being fed back.

We guess that many concepts developed for discrete-time recurrent networks (architectural variations like single-layer higher-order networks [9], learning procedures like RTRL, RBP [15], knowledge extraction and injection techniques [5], theoretical results regarding the computational power [12], etc.) can be lifted to structure-driven recurrent networks.

Recently, [14] proposed a framework for different architectures built of so-called *complex recursive neurons*. Neurons with direct feedback can be obtained as a special case of our folding architecture by setting the number of layers in the folding part to $r = 1$. Single-layer recurrent networks can be derived by setting $r = 1, s = 0$ and reserving some neurons from the representation layer l_1 as output units.

6.3 Evolving Transformation Systems

Recently a framework for inductive learning was given and lead to the provocative claim that neural networks cannot discover generalization in a symbolic environment [6]. The argumentation is based on a mainly symbolic model, the so-called *evolving transformation system* (ETS). An ETS has a *symbolic* and a *geometric* component. The former consists of a set of structured objects, a basic set of finite edit operations (to manipulate objects) and a finite set of operators to build new operations from that basic set. The geometric (or topological) component is defined by a family of distance functions measuring the distance between two structured objects. The edit operations are annotated with real-valued weights. The distance function is computed by taking the minimum sum of weights over all possible sequences of operations that can transform one object into the other. Solving an ILS (binary classification task) means to evolve the given set of operations and weights so that the distance between objects of different classes is maximized while the distance of objects within the same class is minimized.

It is argued that artificial neural network (ANN) models operate on an *algebraic* (matrix-vector operations in a finite-dimensional space) and a *geometric* component (metrics in that space). Further all metrics being consistent with the algebraic component are proven to be equivalent to the Euclidean metric. The ETS model is able to create an infinite family of (not equivalent) distance functions while ANNs have a uniquely defined geometric component. In [6] the authors draw the conclusion that ETS models are more powerful than ANNs and (together with some other arguments) ANNs cannot discover inductive generalization in a symbolic environment.

Here we want to express (at least) some scepticism about this generalized claim which seems to capture all thinkable neural network models. First, providing three layers, sigmoid transfer functions, an infinite number of units in the hidden layer and an infinite precision feedforward networks are proven to be universal approximators [10] and discrete-time continuous-space recurrent networks (Section 6.2) have the computational power of Turing-machines [12]. Obviously, structure-driven recurrent neural networks only rely on an unique *geometric* component (the Euclidean distance) but they are based on a mixed *algebraic-symbolic* component (the space of operations on structures constituted by the weights) which is evolved (directed to solve the given ILS) during the training process.

7 Conclusions

We proposed the folding architecture provided with the supervised gradient-based learning procedure BPTS, a model which may be viewed as only one instance of a whole class of possible structure-driven continuous-space recurrent neural networks. The processing of DAGs is entirely driven by their recursive symbolic nature. We have shown that inductive learning tasks in symbolic domains can be transformed into numerical optimization (supervised learning) problems. Although it is too early to judge the generalization properties of our model the reported experiments give strong evidence that a trained architecture yields a good approximative solution of the presented learning task. The weight set together with the architecture may be regarded as an adaptable operator (processing, combining and manipulating structures and substructures) which is evolved towards the solution.

Our approach requires a large set of training examples but does not need any explicit theory or knowledge about the learning task. This scenario justifies our model as a complement to symbolic approaches (like inductive logic programming). We are currently working on a hybrid (symbolic/connectionist) reasoning system, in which the folding architecture is applied to learn search control heuristics for an automated deduction system.

There are many open questions. Although there are some interesting theoretical results for feedfoward networks [10] and discrete-time continuous-space recurrent neural networks [12] it can only be speculated about the computational power of the folding architecture. Furthermore, as the folding architecture and BPTS is inferred from classical network models we also have to deal with well-known problems like model selection, convergence properties, local minima, resource consumption, etc.

One can observe the strict borders between symbolic and numeric computing begin to vanish. Both ETS (Section 6.3) and the model presented here are based on a mixed symbolic-numeric component. The hope is that an adequate combination will eliminate the disadvantages of the single components.

References

1. G. Berg, "Connectionist Parser with Recursive Sentence Structure and Lexical Disambiguation," in *Proceedings Tenth National Conference on Artificial Intelligence - AAAI-92 San Jose, CA, USA*, pp. 32–37, AAAI, Menlo Park, California, USA, 1992.
2. D. Blank, L. Meeden, and J. 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 11th Conference on Artificial Intelligence (ECAI 94)*, (A. Cohn, ed.), pp. 555–559, John Wiley & Sons, 1994.
4. L. Chrisman, "Learning Recursive Distributed Representations for Holistic Computation," *Connection Science*, no. 3, pp. 345–366, 1991.

5. C. L. Giles and C. W. Omlin, "Extraction, Insertion and Refinement of Symbolic Rules in Dynamically Driven Recurrent Networks," *Connection Science*, vol. 5, no. 3 & 4, pp. 307–337, 1993.
6. L. Goldfarb, J. Abela, V. C. Bhavsar, and V. N. Kamat, "Can a Vector Space Based Learning Model Discover Inductive Class Generalization in a Symbolic Environment?," *Pattern Recognition Letters*, no. 16, pp. 719–726, 1995.
7. C. Goller and A. Küchler, "Learning Task-Dependent Distributed Representations by Backpropagation Through Structure," in *Proceedings of the IEEE International Conference on Neural Networks (ICNN'96)*, 1996. to appear.
8. 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)*, (C. S. Mellish, ed.), pp. 509–515, Morgan Kaufmann Publishers, August 1995.
9. B. Horne and C. Giles, "An Experimental Comparison of Recurrent Neural Networks," in *Advances in Neural Information Processing Systems (NIPS 7)*, (G. Tesauro, D. Touretzky, and T. Leen, eds.), pp. 697–, MIT Press, 1995.
10. K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Network are Universal Approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.
11. J. B. Pollack, "Recursive Distributed Representations," *Artificial Intelligence*, no. 46, pp. 77–105, 1990.
12. H. T. Siegelmann and E. D. Sontag, "On the Computational Power of Neural Nets," *Journal of Computer and System Sciences*, vol. 50, pp. 132–150, 1995.
13. A. Sperduti, "Encoding of Labeled Graphs by Labeling RAAM," in *Advances in Neural Information Processing Systems (NIPS 6)*, (J. D. Cowan, G. Tesauro, and J. Alspector, eds.), pp. 1125–1132, 1994.
14. A. Sperduti and A. Starita, "Supervised Neural Networks for the Classification of Structures," Technical Report TR-16/95, University of Pisa, Dipartimento di Informatica, 1995.
15. R. J. Williams and D. Zipser, "Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity," in *Backpropagation: Theory, Architectures and Applications*, (Y. Chauvin and D. E. Rummelhart, eds.), ch. 13, pp. 433–486, Hillsdale, NJ: Lawrence Erlbaum Associates, 1994.

This article was processed using the LaTeX macro package with LLNCS style