Università degli Studi di Pisa
Dipartimento di Informatica

Supervised Neural Networks for the Classification of Structures

Alessandro Sperduti, Antonina Starita
University of Pisa, Dipartimento di Informatica
Corso Italia 40, 56125 PISA, Italy
E-mail:perso@di.unipi.it

TR-16/95

Abstract

Up to now, neural networks have been used for classification of unstructured patterns and sequences. Dealing with complex structures, however, standard neural networks, as well as statistical methods, are usually believed to be inadequate because of their feature-based approach. In fact, feature-based approaches usually fail to give satisfactory solutions because of the sensitiveness of the approach to the a priori selected features and the incapacity to represent any specific information on the relationships among the components of the structures. On the contrary, we show that neural networks can represent and classify structured patterns. The key idea underpinning our approach is the use of the so called "complex recursive neuron". A complex recursive neuron can be understood as a generalization to structures of a recurrent neuron. By using complex recursive neurons, basically all the supervised networks developed for the classification of sequences, such as Back-Propagation Through Time networks, Real-Time Recurrent networks, Simple Recurrent Networks, Recurrent Cascade Correlation networks,

and Neural Trees can be generalized to structures. The results obtained by some of the above networks (with complex recursive neurons) on classification of logic terms are presented.

1 Introduction

Structured domains are characterized by complex patterns usually represented as lists, trees, and graphs of variable sizes and complexity. The ability to recognize and to classify these patterns is fundamental for several applications, such as medical and technical diagnosis (discovery and manipulation of structured dependencies, constraints, explanations), molecular biology and chemistry (classification of chemical structures, DNA analysis, quantitative structure-property relationship (QSPR), quantitative structure-activity relationship (QSAR)), automated reasoning (robust matching, manipulation of logical terms, proof plans, search space reduction), software engineering (quality testing, modularisation of software), geometrical and spatial reasoning (robotics, structured representation of objects in space, figure animation, layouting of objects), speech and text processing (robust parsing, semantic disambiguation, organising and finding structure in texts and speech), and other applications that use, generate or manipulate structures.

While neural networks are able to classify static information or temporal sequences, the current state of the art does not allow the efficient classification of structures of different sizes. Some advances on this issue have been recently presented in [SSG95], where preliminary work on the classification of logical terms, represented as labeled graphs, was reported. The aim, in that case, was the realization of a hybrid (symbolic/connectionist) theorem prover, where the connectionist part had to learn a heuristic for a specific domain in order to speed-up the search of a proof. Other related techniques for dealing with structured patterns can be found, for example, in [Pol90, Spe94b, Spe95, Spe94a].

The aim of this report is to show, at least in principle, that neural networks can deal with structured domains. Some basic concepts, that we believe very useful for expanding the computational capabilities of neural networks to structured domains, are given. Specifically, we propose a generalization of a recurrent neuron, i.e., the *complex recursive neuron* which is able to build a map from a domain of structures to the set of reals. This new defined neuron allows the formalization of several supervised models for structures which stem

very naturally from well known models, such as Back-Propagation Through Time networks, Real-Time Recurrent networks, Simple Recurrent Networks, Recurrent Cascade Correlation networks, and Neural Trees.

The report is organized as follows. In Section 2 we introduce structured domains and some preliminary concepts on graphs and neural networks. The complex recursive neurons are defined in Section 3, where some related concepts are discussed. Several supervised learning algorithms, derived by standard and well-known learning techniques, are presented in Section 4. Simulation results for a subset of the proposed algorithms are reported in Section 5 and conclusions drawn in Section 6.

2 Structured Domains

In this report, we deal with structured patterns which can be represented as labeled graphs. Some examples of these kind of patterns are shown in Figures 1-4, where we have reported some typical examples from structured domains such as conceptual graphs representing medical concepts (Figure 1), chemical structures (Figure 2), bubble chamber events (Figure 3), and logical terms (Figure 4). All these structures can also be represented by using a feature-based approach. Feature-based approaches, however, have the drawbacks of being very sensitive to the features selected for the representation and incapable to represent any specific information about the relationships among components. Standard neural networks, as well as statistical methods, are usually bound to this kind of representation and thus considered not adequate for dealing with structured domains. On the contrary, we will show that neural networks can represent and classify structured patterns.

2.1 Preliminaries

2.1.1 Graphs

We consider finite directed node labeled graphs without multiple edges. For a set of labels Σ, a *graph* X (over Σ) is specified by a finite set V_X of *nodes*, a set E_X of ordered couples of $V_X \times V_X$ (called the set of *edges*) and a function ϕ_X from V_X to Σ (called the *labeling function*). Note that graphs may have loops, and labels are not restricted to be binary. Specifically, labels may also be real-valued vectors. A graph X' (over Σ) is a *subgraph of* X if $\phi_{X'} = \phi_X$, $V_{X'} \subseteq V_X$,

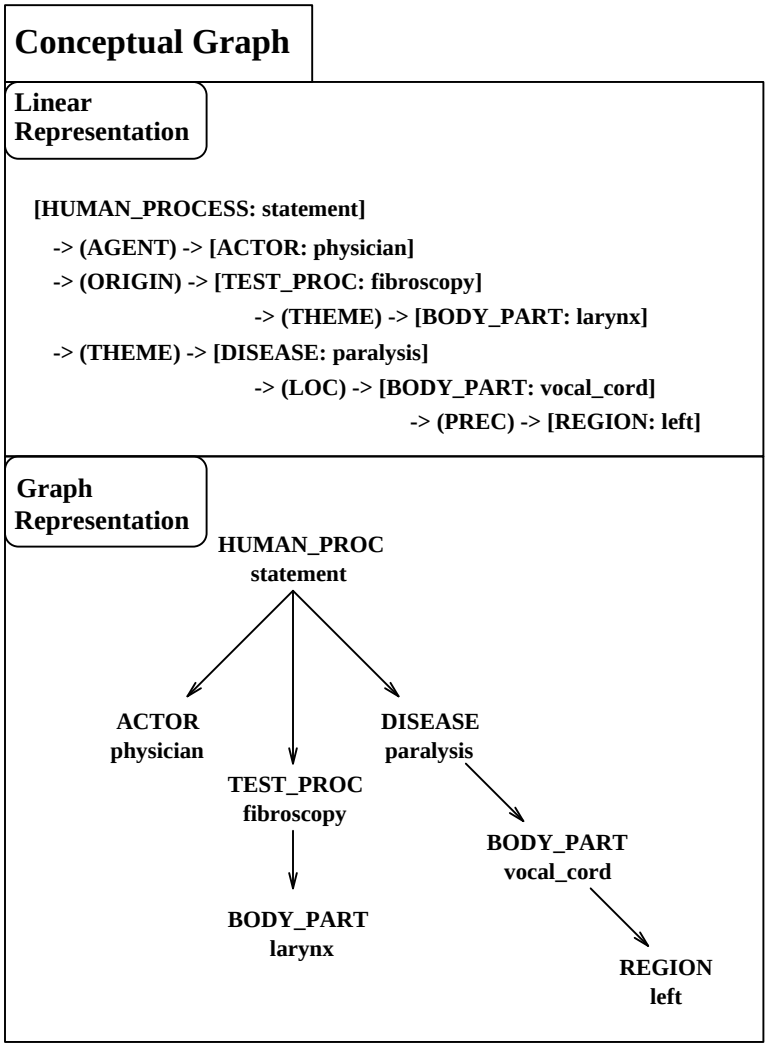


Figure 1: Example of conceptual graph encoding a medical concept

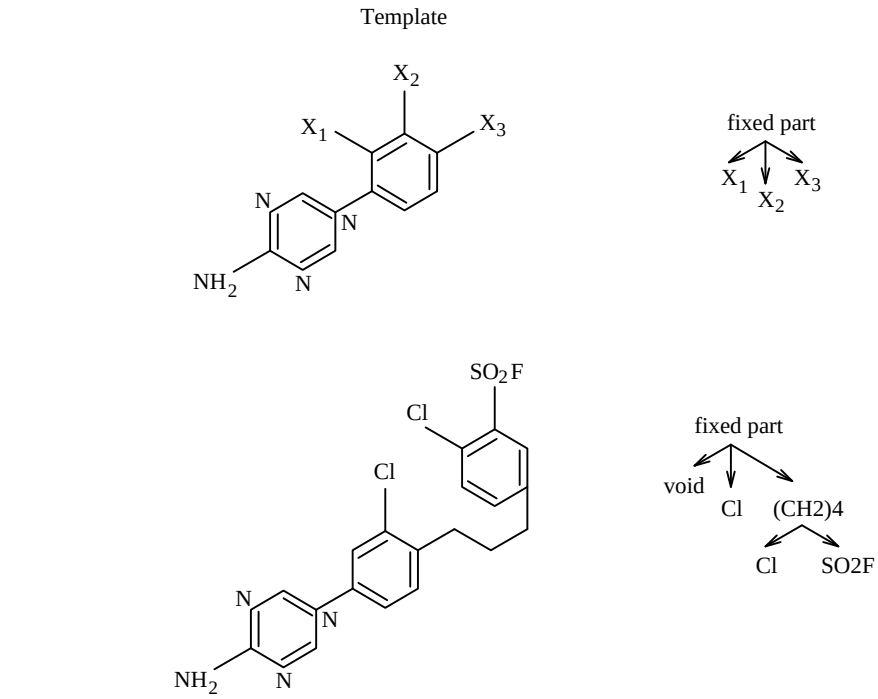
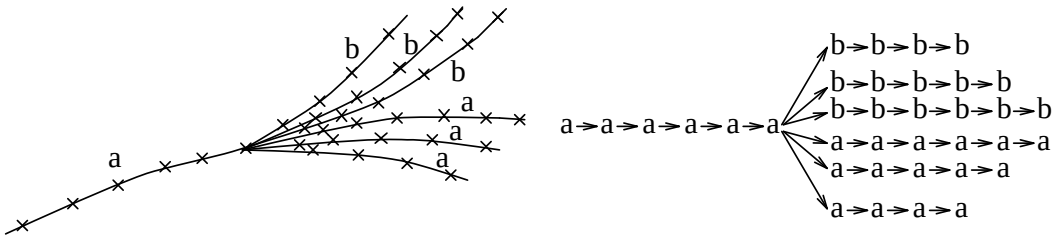


Figure 2: Chemical structures represented as trees.



(a) *(b)*

Figure 3: Bubble chamber events: *(a)* coded event; *(b)* corresponding tree representation.

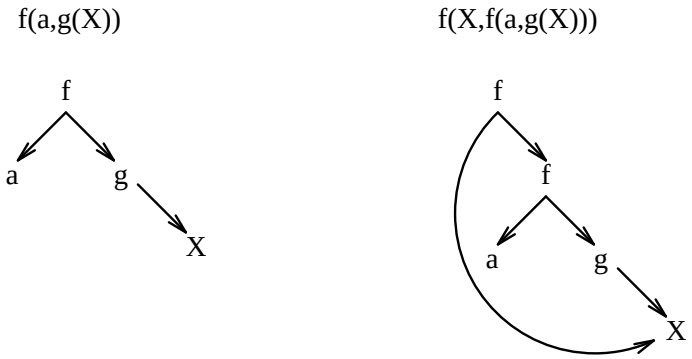


Figure 4: Example of logic terms represented as labeled directed acyclic graphs.

and $E_{X'} \subseteq E_X$. For a finite set V, $\#V$ denotes its cardinality. Given a graph X and any node $x \in V_X$, the function $out\_degree_X(x)$ returns the number of edges leaving from x, i.e., $out\_degree_X(x) = \#\{(x,z) \mid (x,z) \in E_X \wedge z \in V_X\}$. Given a total order on the edges leaving from x, the node $y = out_X(x,j)$ in V_X is the node pointed by the jth pointer leaving from x. The *valence* of a graph X is defined as $\max_{x \in V_X}\{out\_degree_X(x)\}$. A *labeled directed acyclic graph (labeled DAG)* is a graph, as defined above, without loops. A node $s \in V_X$ is called a *supersource* for X if every node in X can be reached by a path starting from s. The root of a tree (which is a special case of directed graph) is always the (unique) *supersource* of the tree.

2.1.2 Structured Domain, Target Function, and Training Set

We define a *structured domain* $\mathcal{D}$ (over Σ) as any (possibly infinite) set of graphs (over Σ). The *valence of a domain* $\mathcal{D}$ is defined as the maximum among the valences of the graphs belonging to $\mathcal{D}$. Since we are dealing with learning, we need to define the target function we want to learn. In approximation tasks, a *target function* $\xi()$ over $\mathcal{D}$ is defined as any function $\xi : \mathcal{D} \rightarrow \Re^k$, were k is the output dimension, while in (binary) classification tasks we have $\xi : \mathcal{D} \rightarrow \{0,1\}$ (or $\xi : \mathcal{D} \rightarrow \{-1,1\}$.) A training set T over a domain $\mathcal{D}$ is defined as a set of couples $(X, \xi(X))$, where $X \in \mathcal{U} \subseteq \mathcal{D}$ and $\xi()$ is a target function defined over $\mathcal{D}$.

2.1.3 Standard and Recurrent Neuron

The output $o^{(s)}$ of a standard neuron is given by

$$o^{(s)} = f(\sum_i w_i I_i), \tag{1}$$

where *f()* is some non-linear squashing function applied to the weighted sum of inputs I[1]. A recurrent neuron with a single self-recurrent connection, instead, computes its output $o^{(r)}(t)$ as follows

$$o^{(r)}(t) = f(\sum_i w_i I_i(t) + w_s o^{(r)}(t-1)), \tag{2}$$

where *f()* is applied to the weighted sum of inputs I plus the self-weight, w_s, times the previous output. The above formula can be extended both considering several interconnected recurrent neurons and delayed versions of the outputs. For the sake of presentation, we skip these extensions.

3 The First Order Complex Recursive Neuron

Using standard and recurrent neurons, it is very difficult to deal with complex structures. This is because the former was devised for processing of unstructured patterns, while the latter can only naturally process sequences. Thus, neural networks using these kind of neurons can face approximation and classification problems in structured domains only using some complex and very unnatural encoding scheme which maps structures onto fixed-size unstructured patterns or sequences. We propose to solve this inadequacy of the present state of the art in neural networks by introducing the *complex recursive neuron*. The complex recursive neuron is an extension of the recurrent neuron where instead of just considering the output of the unit on the previous time step, we have to consider the outputs of the unit for all the nodes which are pointed by the current input node (see Figure 5). The output $o^{(c)}(x)$ of the complex recursive

[1] The threshold of the neuron is included in the weight vector by expanding the input vector with a component always to 1.

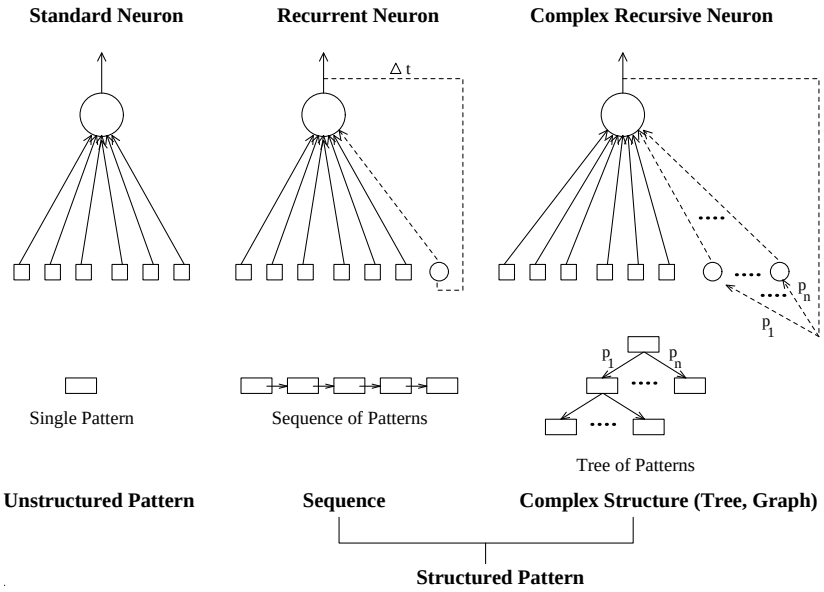


Figure 5: Neuron models for different input domains. The standard neuron is suited for the processing of unstructured patterns, the recurrent neuron for the processing of sequences of patterns, and finally the proposed complex recursive neuron can deal very naturally with structured patterns.

neuron to a node x of a graph X is defined as

$$o^{(c)}(x) = f(\sum_{i=1}^{N_L} w_i l_i + \sum_{j=1}^{out\_degree_X(x)} \hat{w}_j o^{(c)}(out_X(x,j))), \tag{3}$$

where N_L is the number of units encoding the label $\boldsymbol{l} = \phi_X(x)$ attached to the current input x, and $\hat{w}_j$ are the weights on the recursive connections. Thus, the output of the neuron for a node x is computed recursively on the output computed for all the nodes pointed by it.

Note that, if the valence of the considered domain is n, then the complex recursive neuron will have n recursive connections, even if not all of them will be used for computing the output of a node x with $out\_degree_X(x) < n$.

When considering N_c interconnected complex recursive neurons, eq. (3) becomes

$$\boldsymbol{o}^{(c)}(x) = \boldsymbol{F}(\boldsymbol{W}\boldsymbol{l} + \sum_{j=1}^{out\_degree_X(x)} \widehat{\boldsymbol{W}}_j \boldsymbol{o}^{(c)}(out_X(x,j))), \tag{4}$$

where $\boldsymbol{F}_i(\boldsymbol{v}) = f(v_i)$, $\boldsymbol{l} \in \Re^{N_L}$, $\boldsymbol{W} \in \Re^{N_c \times N_L}$, $\boldsymbol{o}^{(c)}(x)$, $\boldsymbol{o}^{(c)}(out_X(x,j)) \in \Re^{N_c}$, $\widehat{\boldsymbol{W}}_j \in \Re^{N_c \times N_c}$.

In the following, we will refer to the output of a complex neuron dropping the upper index.

3.1 Generation of Neural Representations for Graphs

To understand how complex recursive neurons can generate representations for directed graphs, let us consider a single complex recursive neuron u and a single graph X. The following conditions must hold:

Number of Connections: the complex recursive neuron u must have as many recursive connections as the valence of the graph X;

Supersource: the graph X must have a reference *supersource*.

Note that, if the graph X does not have a supersource, then it is always possible to define a convention for adding to the graph X an extra node s (with a minimal number of outgoing edges) such that s is a supersource for the new graph;

If the above conditions are satisfied, we can adopt the convention that the graph X is represented by $o(s)$, i.e., the output of u to s. Consequently, due

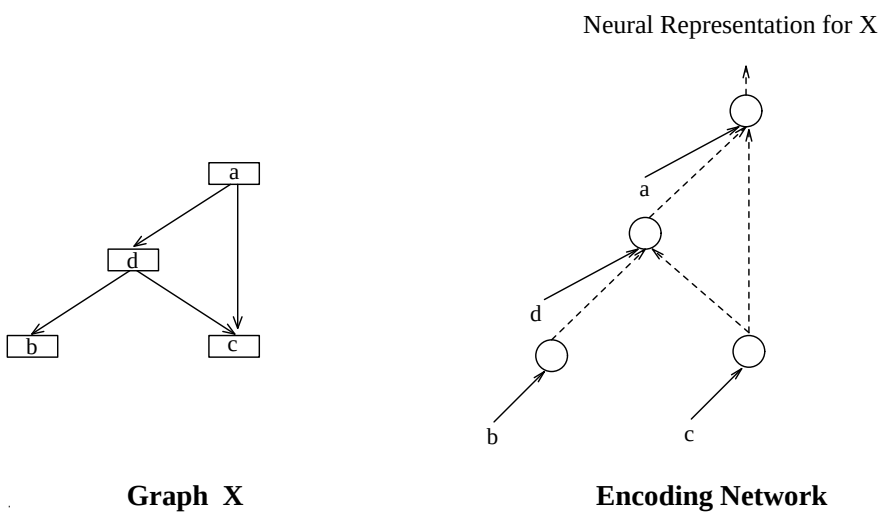


Figure 6: The Encoding network for an acyclic graph. The graph is represented by the output of the Encoding network.

to the recursive nature of eq. (3), it follows that the neural representation for an acyclic graph is computed by a feedforward network (*encoding network*) obtained by replicating the same complex recursive neuron u and connecting these copies according to the topology of the structure (see Figure 6). If the structure contains cycles then the resulting encoding network is recurrent (see Figure 7) and the neural representation is considered to be well-formed only if $o(s)$ converges to a stationary value.

The encoding network fully describes how the representation for the structure is computed and it will be used in the following to derive the learning rules for the complex recursive connections.

When considering a structured domain, the number of recursive connections of u must be equal to the valence of the domain. The extension to a set of N_c complex neurons is trivial: if the valence of the domain is k, each complex neuron will have k groups of N_c recursive connections each.

3.2 Optimized Training Set

When considering DAGs, the training set can be organized so to improve the computational efficiency of both the reduced representations and the learning rules. In fact, given a training set T of DAGs, if there are graphs $X_1, X_2 \in T$

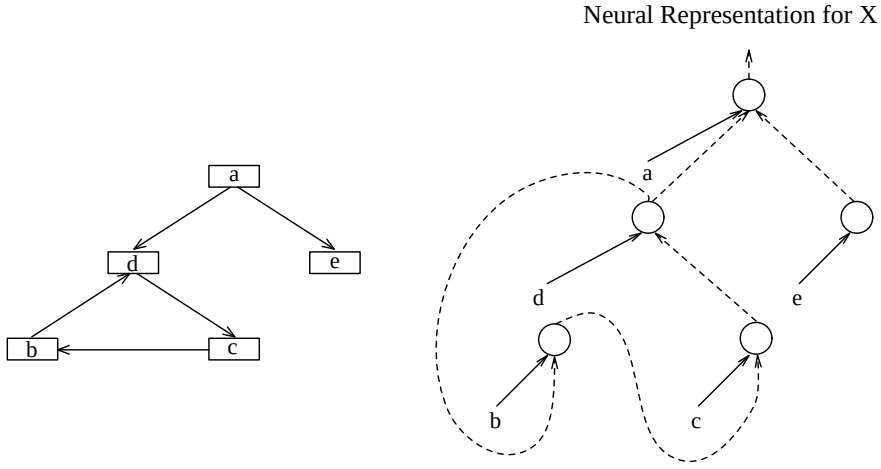


Figure 7: The Encoding network for a cyclic graph. In this case, the Encoding network is recurrent and the graph is represented by the output of the Encoding network at a fixed point of the network dynamics.

which share a common subgraph $\hat{X}$, then we need to explicitly represent $\hat{X}$ only once. The optimization of the training set can be performed in two stages (see Fig. 8 for an example):

1. all the DAGs in the training set are merged into a single minimal DAG, i.e., a DAG with minimal number of nodes;

2. a topological sort on the nodes of the minimal DAG is performed to determine the updating order on the nodes for the network.

Both stages can be done in linear time with respect to the size of all DAGs and the size of the minimal DAG, respectively[2]. Specifically, stage (1.) can be done by removing all the duplicates subgraphs through a special subgraph-indexing mechanism (which can be implemented in linear time). The advantage of having a sorted training set is due to the fact that all the reduced representations (and also their derivatives with respect to the weights, as we will see when considering learning) can be computed by a single ordered scan of the training set.

[2] This analysis is due to Christoph Goller.

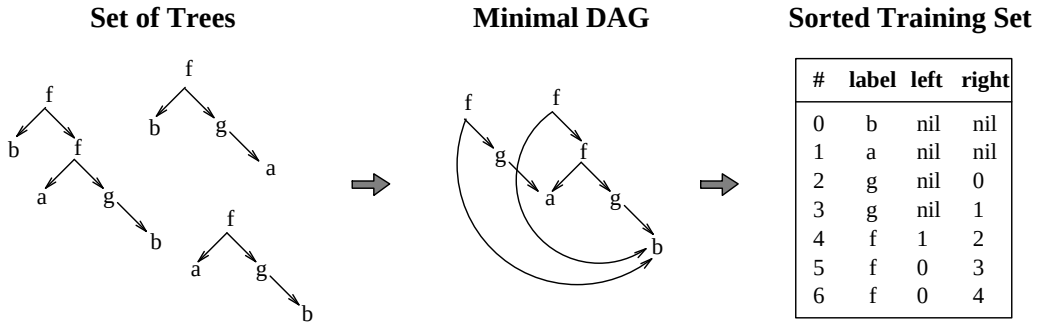


Figure 8: Optimization of the training set: the set of structures (in this case, trees) is transformed into the corresponding minimal DAG, which is then used to generate the sorted training set. The sorted training set is then transformed into a set of sorted vectors using the numeric codes for the labels and used as training set for the network.

3.3 Well-formedness of Neural Representations

When considering cyclic graphs, to guarantee that each encoded graph gets a proper representation through the encoding network, we have to guarantee that for every initial state the trajectory of the encoding network converges to an equilibrium, otherwise it would be impossible to perform the processing of a nonstationary representation. This is particularly important when considering cyclic graphs. In fact, acyclic graphs are guaranteed to get a convergent representation because the resulting encoding network is feedforward, while cyclic graphs are encoded by using a recurrent network. Consequently, the well-formedness of representations can be obtained by defining conditions guaranteeing the convergence of the encoding network. Regarding that, it is well known that if the weight matrix is symmetric, an additive network with first order connections possess a Liapunov function and is convergent ([Hop84, CG83]). Moreover, Almeida [Alm87] proved a more general symmetry condition than symmetry of the weight matrix, i.e., a system satisfying *detailed balance*

$$w_{ij} f(net_j) = w_{ji} f(net_i), \tag{5}$$

is guaranteed to possess a Liapunov function as well. On the other hand, if the norm of the weight matrix (not necessarily symmetric) is sufficiently small,

e.g., satisfying

$$\sum_i \sum_j w_{ij}^2 < \frac{1}{max_i\ f'(net_i)}, \tag{6}$$

the network's dynamics can be shown to go to a unique equilibrium for a given input (Atiya [Ati88]).

The above results are sufficient when the encoding network is static, however we will see that in several cases, e.g., in classification tasks, the encoding network changes with learning. In this cases, these results can be exploited to define the initial weight matrix, but there is no guarantee that learning will preserve the stability of the encoding network.

4 Supervised Models

In this section, we discuss how several standard supervised algorithms for neural networks can be extended to structures.

4.1 Back-propagation Through Structure

The task addressed by back-propagation through time networks is to produce particular output sequences in response to specific input sequences. These networks are, in general, fully recurrent, in which any unit may be connected to any other. Because of that, they cannot be trained by using plain back-propagation. A trick, however, can be used to turn an arbitrary recurrent network into an equivalent feedforward network when the input sequences have a maximum length T. In this case, all the units of the network can be duplicated T times (*unfolding of time*), so that the state of a unit in the recurrent network at time τ is hold from the τth copy of the same unit in the feedforward network. By preserving the same weight values through the layers of the feedforward network, it is not difficult to see that the two networks will behave identically for T time steps. The feedforward network can be trained by back-propagation, having care to preserve the identity constraint between weights of different layers. This can be guaranteed by adding together the individual gradient contributions of corresponding copies of the same weight and then changing all copies by the total amount.

Back-propagation Through Time can be extended without difficulties to structures. The basic idea is to use complex recursive neurons for encod-

ing the structures; the obtained representations are then classified or used to approximate an unknown function by a standard feedforward network. An example of this kind of network for classification is given in Figure 9.

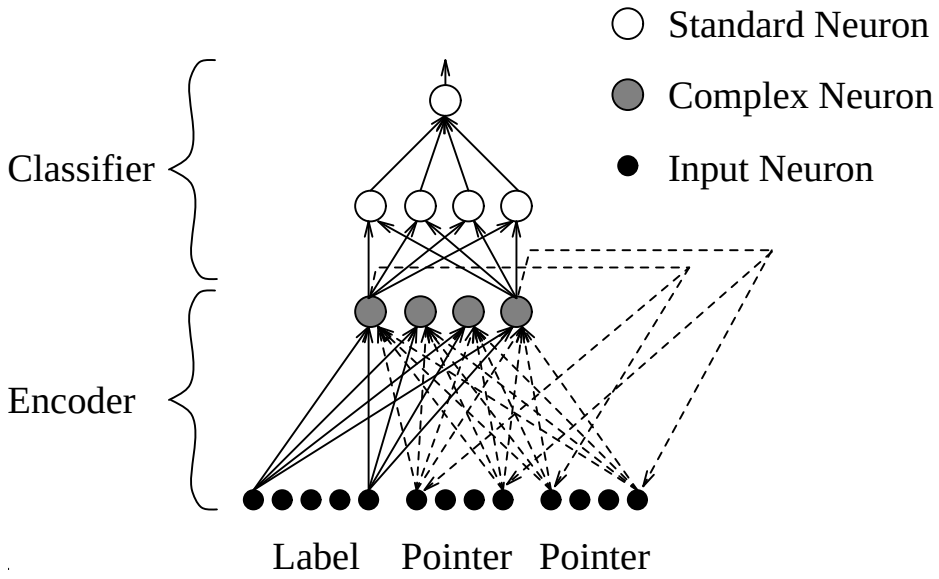


Figure 9: A possible architecture for the classification of structures: the complex recursive neurons generate the neural representation for the structures, which are then classified by a standard feedforward network.

Given an input graph X, the network output $\boldsymbol{o}(X)$ can be expressed as the composition of the encoding function $\Psi()$ and the classification (or approximation) function $\Phi()$ (see Figure 10):

$$\boldsymbol{o}(X) = \Phi(\Psi(X)). \tag{7}$$

Learning for the set of weights $\boldsymbol{W}_\Phi$ can be implemented by plain backpropagation on the feedforward network realizing $\Phi()$

$$\Delta \boldsymbol{W}_\Phi = -\eta \frac{\partial Error(\Phi(\boldsymbol{y}))}{\partial \boldsymbol{W}_\Phi}, \tag{8}$$

where $\boldsymbol{y} = \Psi(X)$, i.e., the input to the feedforward network, while learning for

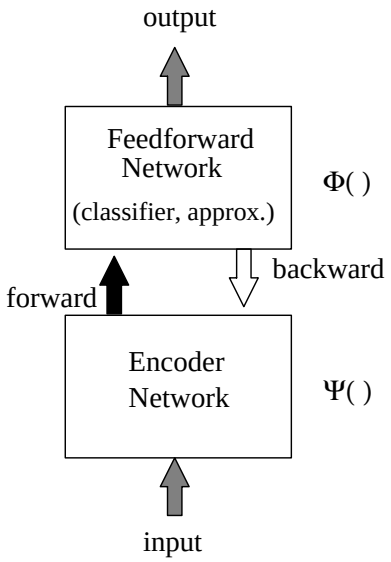


Figure 10: The functional scheme of the proposed network. The Encoder and the Classifier are considered as two distinct entities which exchange information: the Encoder forwards the neural representations of the structures to the Classifier; in turn the Classifier returns to the Encoder the deltas which are then used by the Encoder to adapt its weights.

the set of weights, $\boldsymbol{W}_\Psi$, realizing $\Psi()$ can be implemented by

$$\Delta \boldsymbol{W}_\Psi = -\eta \frac{\partial Error(\Phi(\boldsymbol{y}))}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}_\Psi}, \tag{9}$$

where the first term represents the error coming from the feedforward network and the second one represents the error due to the encoding network.

How the second term is actually computed depends on the family of structures in the training set. If the training set is composed by DAGs, then plain back-propagation can be used on the encoding network. Otherwise, i.e., if there are graphs with cycles, then *recurrent back-propagation* [Pin88] must be used. Consequently, we treat separately these two cases.

4.1.1 Case I: DAGs

This case has been treated by Goller and Küchler in [GK95]. Since the training set contains only DAGs, the computation of $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}_\Psi}$ can be realized by backpropagating the error from the feedforward network through the encoding network of each structure. As in back-propagation through time, the gradient contributions of corresponding copies of the same weight are collected for each structure. The total amount is then used to change all the copies of the same weight. If the learning is performed by *structure* then the weights are updated after the presentation of each single structure, otherwise, the gradient contributions are collected through the whole training set and the weights changed after that all the structures in the training set have been presented to the network.

4.1.2 Case II: Cyclic Graphs

When considering cyclic graphs, $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}_\Psi}$ can be computed only resorting to *recurrent back-propagation*. In fact, if the input structure contains cycles, then the resulting encoding network is cyclic.

In the standard formulation, a recurrent back-propagation network $\mathcal{N}$ with n units is defined as

$$\boldsymbol{o}^{(rbp)}(t+1) = \boldsymbol{F}(\boldsymbol{W}^{(rbp)}\boldsymbol{o}^{(rbp)}(t) + \boldsymbol{I}^{(rbp)}), \tag{10}$$

where $\boldsymbol{I}^{(rbp)} \in \Re^n$ is the input vector for the network, and $\boldsymbol{W}^{(rbp)} \in \Re^n \times \Re^n$ the weight matrix. The learning rule for a weight of the network is given by

$$\Delta w_{rs}^{(rbp)} = \eta o_s^{(rbp)} o_r'^{(rbp)} \sum_k e_k (\boldsymbol{L}^{-1})_{kr} \tag{11}$$

where all the quantities are taken at a fixed point of the recurrent network, e_k is the error between the current output of unit k and the desired one, $L_{ji} = \delta_{ji} - o_j'^{(rbp)} w_{ji}$ (δ_{ji} is a Kronecker delta), and the quantity

$$y_j^{(rbp)} = \sum_k e_k (\boldsymbol{L}^{-1})_{kj} = \sum_i o_i'^{(rbp)} w_{ij} y_i^{(rbp)} \ + \ e_i \tag{12}$$

can be computed by relaxing the *adjoint* network $\mathcal{N}'$, i.e., a network obtained from $\mathcal{N}$ by reversing the direction of the connections. The weight w_{ji} from neuron i to neuron j in $\mathcal{N}$ is replaced by $o_i^{(rbp)} w_{ij}$ from neuron j to neuron i in $\mathcal{N}'$. The activation functions of $\mathcal{N}'$ are linear and the output units in $\mathcal{N}$ become input units in $\mathcal{N}'$ with e_i as input.

Given a cyclic graph X, let $m = \#V_X$, N_c be the number of complex neurons and $\boldsymbol{o}_i(t)$ the output at time t of these neurons for $x_i \in V_X$. Then we can define

$$\boldsymbol{o}(t) = \begin{bmatrix} \boldsymbol{o}_1(t) \\ \boldsymbol{o}_2(t) \\ \vdots \\ \boldsymbol{o}_m(t) \end{bmatrix} \tag{13}$$

as the global state vector for our recurrent network, where for convention we impose $\boldsymbol{o}_1(t)$ to be the output of the neurons representing the *supersource* of X.

To account for the labels, we have to slightly modify eq. (10)

$$\boldsymbol{o}(t+1) = \boldsymbol{F}(\widehat{\boldsymbol{W}}_\Psi^X \boldsymbol{o}(t) + \boldsymbol{W}_\Psi^X \boldsymbol{I}^X), \tag{14}$$

where

$$\boldsymbol{I}^X = \begin{bmatrix} \boldsymbol{l}_1 \\ \vdots \\ \boldsymbol{l}_m \end{bmatrix} \tag{15}$$

with $\boldsymbol{l}_i = \phi_X(x_i)$, $i = 1, \ldots, m$,

$$\boldsymbol{W}_{\Psi}^{X} = \underbrace{\begin{bmatrix} \boldsymbol{W} & & 0 \\ & \ddots & \\ 0 & & \boldsymbol{W} \end{bmatrix}}_{m \text{ repetitions of } \boldsymbol{W}} \tag{16}$$

and $\widehat{\boldsymbol{W}}_{\Psi}^{X} \in \Re^{mN_c \times mN_c}$ is defined according to the topology of the graph.

In this context, the input of the adjoint network is $e_k = (\frac{\partial Error(\Phi(\boldsymbol{y}))}{\partial \boldsymbol{y}})_k$, and the learning rules become:

$$\Delta \hat{w}_{rs} = \eta o_s o'_r \sum_k e_k (\boldsymbol{L}^{-1})_{kr}, \tag{17}$$

$$\Delta w_{rs} = \eta I_s o'_r \sum_k e_k (\boldsymbol{L}^{-1})_{kr}. \tag{18}$$

To hold the constraint on the weights, all the changes referring to the same weight are added and then all copies of the same weight are changed by the total amount. Note that each structure gives rise to a new adjoint network and independently contributes to the variations of the weights. Moreover, the above formulation is more general than the one previously discussed and it can be used also for DAGs, in which case the adjoint network becomes a feedforward network representing the back-propagation of the errors.

4.2 Extension of Real-Time Recurrent Learning

The extension of Real-Time Recurrent Learning [WZ89] to complex recursive neurons does not present particular problems when considering graphs without cycles. Cyclic graphs, however, present problems, and in general yield to a training algorithm that can be considered only loosely in real time. In the following, we will discuss only acyclic graphs. In order to be concise, we will only show how derivatives can be computed in real time, leaving to the reader the development of the learning rules, according to the chosen network architecture and error function.

Let N_c be the number of complex neurons, s the *supersource* of X, and

$$\boldsymbol{y} = \boldsymbol{o}(s) = \boldsymbol{F}(\boldsymbol{W}\boldsymbol{l}_s + \sum_{j=1}^{out\_degree_X(s)} \widehat{\boldsymbol{W}}_j \boldsymbol{o}(out_X(s,j))) \tag{19}$$

be the representation of X according to the encoding network, where

$$\boldsymbol{W}_\Psi = [\boldsymbol{W}, \widehat{\boldsymbol{W}}_1, \ldots, \widehat{\boldsymbol{W}}_n], \tag{20}$$

and n is the valence of the domain. The derivatives of $\boldsymbol{y}$ with respect to $\boldsymbol{W}$ and $\widehat{\boldsymbol{W}}_i$ ($i \in [1, \ldots, n]$) can be computed from eq. (19):

$$\frac{\partial y_t}{\partial W_{tk}} = o_t'(s)((\boldsymbol{l}_s)_k + \sum_{j=1}^{out\_degree_X(s)} (\widehat{\boldsymbol{W}}_j)_t \frac{\partial \boldsymbol{o}(out_X(s,j))}{\partial W_{tk}}), \tag{21}$$

where $t = 1, \ldots, N_c$, $k = 0, \ldots, N_L$ and $(\widehat{\boldsymbol{W}}_j)_t$ is the tth row of $\widehat{\boldsymbol{W}}_j$;

$$\frac{\partial y_t}{\partial (\widehat{\boldsymbol{W}}_i)_{tq}} = o_t'(s)(o_q(out_X(s,i)) + \sum_{j=1}^{out\_degree_X(s)} (\widehat{\boldsymbol{W}}_j)_t \frac{\partial \boldsymbol{o}(out_X(s,j))}{\partial (\widehat{\boldsymbol{W}}_i)_{tq}}), \tag{22}$$

where $t = 1, \ldots, N_c$, and $q = 1, \ldots, N_c$.

These equations are recursive on the structure X and can be computed by noting that if v is such that $out\_degree_X(v) = 0$, then

$$\frac{\partial o_t(v)}{\partial W_{tk}} = (\boldsymbol{l}_v)_k, \text{ and } \frac{\partial o_t(v)}{\partial (\widehat{\boldsymbol{W}}_i)_{tq}} = 0. \tag{23}$$

This allows the computation of the derivatives in real time alongside with the computation of the reduced descriptors for the graphs.

4.3 LRAAM-based Networks and Simple Recurrent Networks

In this section, we present a class of networks which are based on the LRAAM model [SSG95]. Learning in these networks is implemented via the combination of a supervised procedure with an unsupervised one and, since it is based on truncated gradients, it is suited for both acyclic and cyclic graphs. We will see that this class of networks contains, as a special case, a network which results to be the extension of a Simple Recurrent Network [Elm90] to structures.

In this type of networks (see Figure 11, the first part of the network is constituted by an LRAAM (note the double arrows on the connections) whose task is to devise a compressed representation for each structure. This compressed representation is obtained by using the standard learning algorithm

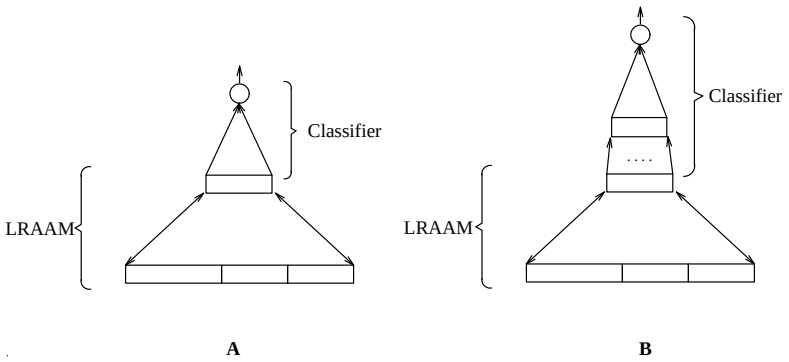


Figure 11: LRAAM-based networks for the classification of structures.

for LRAAM [SSG94]. The classification task is then performed in the second part of the network through a multi-layer feed-forward network with one or more hidden layers (network **A**) or a simple sigmoidal neuron (network **B**). Several options for the training of networks **A** and **B** are available. The different options we have for the training can be characterized by the proportion of the two different learning rates (for the classifier and the LRAAM) and by the different degrees **x**, and **y** of presence of the following two basic features:

- the training of the classifier is started not until **x** percent of the training set is correctly encoded and successively decoded by the LRAAM;
- the error coming from the classifier is backpropagated across **y** levels of the structures encoded by the LRAAM[3].

Note that, even if the training of the classifier is started only when all the structures in the training set are properly encoded and decoded, still the classifier's error can change the compressed representations which, however, are maintained consistent[4] by learning in the LRAAM.

The reason for allowing different degrees of interaction between the classification and the representation tasks is due to the necessity of having different degrees of adaptation of the compressed representations to the requirements

[3]The backpropagation of the error across several levels of the structures can be implemented by unfolding the encoder of the LRAAM (the set of weights from the input to the hidden layer) according to the topology of the structures.

[4]A consistent compressed representation is a representation of a structure which contains all the information sufficient for the reconstruction of the whole structure.

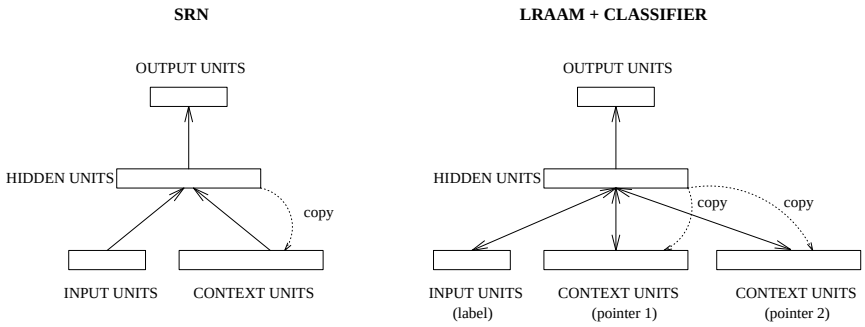


Figure 12: The network **B**, with **x**= 0 and **y**= 1 (right side), can be considered as a generalization of the Simple Recurrent Network by Elman (left side).

of the classification task. If no interaction at all is allowed, i.e., the LRAAM is trained first and then its weights frozen (**y**= 0), the compressed representations will be such that similar representations will correspond to similar structures, while if full interaction is allowed, i.e., the LRAAM and the classifier are trained simultaneously, the compressed representations will be such that structures in the same class will get very similar representations[5]. On the other hand, when considering DAGs, by setting **y**= max_depth, where max_depth is the number of nodes traversed by the longest path in the structures, the classifier error will be backpropagated across the whole structure, thus implementing the backpropagation through structure defined in Section 4.1.

It is interesting to note that the SRN by Elman can be obtained as a special case of network **B**. In fact, when considering network **B** (with **x**= 0 and **y**= 1) for the classification of lists (sequences) the same architecture is obtained, with the difference that there are connections from the hidden layer of the LRAAM back to the input layer[6], i.e., the decoding part of the LRAAM. Thus, when considering lists, the only difference between a SRN and network **B** is in the unsupervised learning performed by the LRAAM. However, when forcing the learning parameters for the LRAAM to be null, we obtain the same learning algorithm as in SRN. Consequently, we can claim that SRN is a special case of

[5] Moreover, in this case, there is no guarantee that the LRAAM will be able to encode and to decode consistently all the structures in the training set, since the training is stopped when the classification task is performed correctly.

[6] The output layer of the LRAAM can be considered the same as the input layer.

network **B**. This can be better understood by looking at the right side of Figure 12, where we have represented network **B** in terms of elements of a SRN. Of course, the *copy* function for network **B** is not as simple as the one used in a SRN, since the right relationships among components of the structures to be classified must be preserved[7].

4.4 Cascade-Correlation for Structures

The Cascade-Correlation algorithm [FL90] grows a standard neural network using an incremental approach for classification of unstructured patterns. The starting network $\mathcal{N}_0$ is a network without hidden nodes trained with a Least Mean Square algorithm; if network $\mathcal{N}_0$ is not able to solve the problem, a hidden unit u_1 is added such that the *correlation* between the output of the unit and the residual error of network $\mathcal{N}_0$ is maximised[8]. The weights of u_1 are frozen and the remaining weights are retrained. If the obtained network $\mathcal{N}_1$ cannot solve the problem, the network is further grown, adding new hidden units which are connected (with frozen weights) with all the inputs and previously installed hidden units. The resulting network is a *cascade* of nodes. Fahlman extended the algorithm to classification of sequences, obtaining good results [Fah91]. In the following, we show that Cascade-Correlation can further be extended to structures by using complex recursive neurons. For the sake of simplicity, we will discuss the case of acyclic graphs, leaving to the reader the extension to cyclic graphs (see Section 4.1, cyclic graphs).

The output of the kth hidden unit, in our framework, can be computed as

$$o^{(k)}(x) = f(\sum_{i=1}^{N_L} w_i^{(k)} l_i + \sum_{v=1}^{k} \sum_{j=1}^{out\_degree_X(x)} \hat{w}_{(v,j)}^{(k)} o^{(v)}(out_X(x,j)) + \sum_{q=1}^{k-1} \bar{w}_q^{(k)} o^{(q)}(x)), \tag{24}$$

where $w_{(v,j)}^{(k)}$ is the weight of the kth hidden unit associated to the output of the vth hidden unit computed on the jth component pointed by x, and $\bar{w}_q^{(k)}$ is the weight of the connection from the qth (frozen) hidden unit, $q < k$, and

[7] The *copy* function needs a stack for the memorization of compressed representations. The control signals for the stack are defined by the encoding-decoding task.

[8] Since the maximization of the correlation is obtained using a gradient ascent technique on a surface with several maxima, a pool of hidden units is trained and the best one selected.

the kth hidden unit. The output of the output neuron $u^{(out)}$ is computed as

$$o^{(out)}(x) = f(\sum_{i=1}^{k} \tilde{w}_i o^{(i)}(x)), \tag{25}$$

where $\tilde{w}_i$ is the weight on the connection from the ith (frozen) hidden unit to the output unit.

Learning is performed as in standard Cascade-Correlation, with the difference that, according to equation (24), the derivatives of $o^{(k)}(x)$ with respect to the weights are now:

$$\frac{\partial o^{(k)}(x)}{\partial w_i^{(k)}} = (l_i + \sum_{j=1}^{out\_degree_X(x)} \hat{w}_{(k,j)}^{(k)} \frac{\partial o^{(k)}(out_X(x,j))}{\partial w_i^{(k)}}) f' \tag{26}$$

$$\frac{\partial o^{(k)}(x)}{\partial \bar{w}_q^{(k)}} = (o^{(q)}(x) + \sum_{j=1}^{out\_degree_X(x)} \hat{w}_{(k,j)}^{(k)} \frac{\partial o^{(k)}(out_X(x,j))}{\partial \bar{w}_q^{(k)}}) f' \tag{27}$$

$$\frac{\partial o^{(k)}(x)}{\partial \hat{w}_{(v,t)}^{(k)}} = (o^{(v)}(out_X(x,t)) + \sum_{j=1}^{out\_degree_X(x)} \hat{w}_{(v,j)}^{(k)} \frac{\partial o^{(k)}(out_X(x,j))}{\partial \hat{w}_{(v,t)}^{(k)}}) f' \tag{28}$$

where $i = 1, \ldots, N_L$, $q = 1, \ldots, (k-1)$, $v = 1, \ldots, k$, $t = 1, \ldots, out\_degree_X(x)$, f' is the derivative of $f()$. The above equations are recurrent on the structures and can be computed by observing that for a leaf node y equation (26) reduces to $\frac{\partial o^{(k)}(y)}{\partial w_i^{(k)}} = l_i$, and all the remaining derivatives are null. Consequently, we only need to store the output values of the unit and its derivatives for each component of a structure. Figure 13 shows the evolution of a network with two pointer fields. Note that, if the hidden units have self-recurrent connections only, the matrix defined by the weights $\hat{w}_{(v,j)}^{(k)}$ is not triangular, but diagonal.

4.4.1 Computational Complexity

The algorithm implements a gradient technique (local search) and thus it is not possible to guess how many iterations are necessary for convergence. In the following, to have a feeling of the complexity, we suppose that each hidden unit is trained for a bounded number of iterations[9]. Let P be the number of

[9] In practice, a bound on the number of iterations for the training of a single hidden unit is always used.

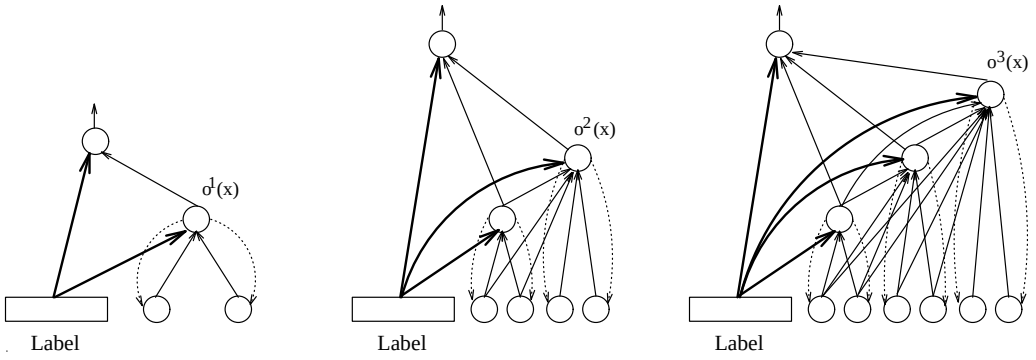


Figure 13: The evolution of a network with two pointer fields. The units in the label are fully connected with the hidden units and the output unit.

nodes in the minimal DAG. The number of free parameters (connections and threshold) n_k for the kth hidden unit is

$$n_k = \underbrace{N_L}_{label\ connections} + \underbrace{N_V(\frac{k(k+1)}{2})}_{recursive\ connections} + \underbrace{(k-1)}_{frozen\ units\ connections} + \underbrace{1}_{threshold} \quad (29)$$

where N_V is the valence of the domain. Consequently, observing that during learning the output of the frozen hidden units computed on the training set can be stored, the output of the kth hidden unit on a single pattern (i.e., graph node) can be calculated in $O(k^2 N_V)$. This complexity dominates the cost of computing the output of the output unit which is proportional to ($\underbrace{N_L}_{label's\ bits}$ + $\underbrace{k}_{frozen\ units}$). Concerning learning, it is trivial to note that the cost of training a single hidden unit dominates the cost of training the output unit. According to eq.s (26)-(28), the derivatives for the kth hidden unit with respect to a single pattern can be computed in $O(k^2 N_V^2)$ in time, since $out\_degree_X(x) \leq N_V$. Thus, considering the full training set it takes $O(Pk^2 N_V^2)$ in time. Finally, when building a network with k hidden units, the computation of all the derivatives takes $O(Pk^3 N_V^2)$ in time[10]. The complexity in space is dominated by the space necessary for storing the derivatives of the current trained unit, i.e., $O(Pk^2 N_V)$.

[10]The total number of computations in one step is proportional to $\sum_{i=1}^{k} Pi^2 N_V^2 = PN_V^2 \sum_{i=1}^{k} i^2 = PN_V^2(\frac{k(k+1)(2k+1)}{6})$.

When considering a diagonal connection matrix, i.e., hidden units with self-recursive connections only, the complexity of learning becames $O(Pk^2N_V^2)$ in time and $O(PkN_V)$ in space.

4.5 Extension of Neural Trees

Neural Trees (NT) have been recently proposed as a fast learning method in classification tasks. They are decision trees [BFOS84] where the splitting of the data for each node, i.e., the classification of a pattern according to some features, is performed by a perceptron [SN90] or a more complex neural network [SM91]. After learning, each node at every level of the tree corresponds to an exclusive subset of the training data and the leaf nodes of the tree completely partition the training set. In the operative mode, the internal nodes route the input pattern to the appropriate leaf node which represents the class of it. An example of how a binary neural tree splits the training set is shown in Figure 14.

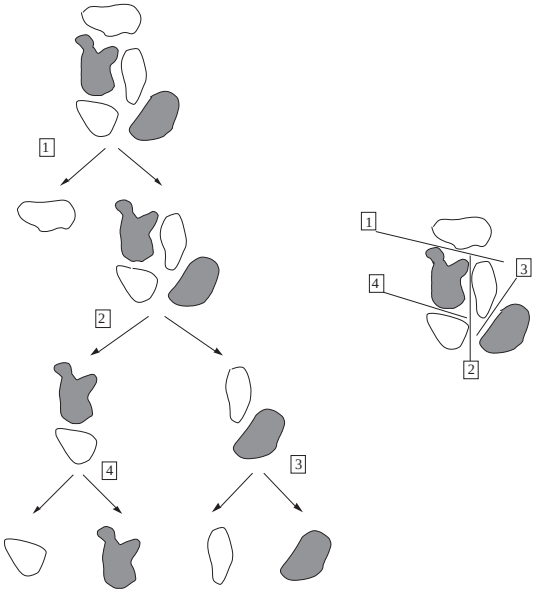


Figure 14: An example of neural tree.

One advantage of the neural tree approach is that the tree structure is constructed dynamically during learning and not linked to a static structure

like a standard feed-forward network. Moreover, it allows incremental learning, since subtrees can be added as well as deleted to recognize new classes of patterns or to improve generalization. Both supervised [SN90, Set90, Aa92, SM91] and unsupervised [PI92, LFJ92, Per92] splitting of the data have been proposed. We are interested in supervised methods.

The learning and classification algorithms for binary neural trees is described in Figure (15), respectively. Algorithms for general trees can be found in [SM91]. The extension of these algorithms to structures is straightforward: the standard discriminator (or network) associated to each node of the tree is replaced by a complex recursive discriminator (or network) which can be trained with any of the learning algorithms we have presented so far.

5 Experimental Results

We have tested some of the proposed architectures on several classification tasks involving logic terms. In the next subsection, we present the classification problems. Then, we discuss the results obtained with LRAAM-based networks and Cascade-Correlation for structures.

5.1 Description of the Classification Problems

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 (with associated arity) compounding the terms, the third column shows the rule(s) used to generate the positive examples of the problem[11], the fourth column reports the number of terms in the training and test set respectively, the fifth column the number of subterms in the training and test set, and the last column the maximum depth[12] of terms in the training and test set. 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.

[11] Note that the terms are all ground.

[12] 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.

Learning Algorithm

Let $\{(\boldsymbol{I}_k, l_k)\}$ be the training set, where $\boldsymbol{I}_k$ is a real valued feature vector and l_k the label of the associated class, $k = 1, \cdots, P$. Let $T(t)$ be the current training set for node t, D_t the discriminator associated to t, if it is a nonterminal node, or the label of a class if it is a leaf.

1. Set $t = root$ and $T(root) = \{(\boldsymbol{I}_k, l_k)\}$;
2. If $T(t) = \emptyset$ then stop;
3. Train the discriminator D_t to split $T(t)$ in two sets: $T_l(t)$, where the output of D_t is 0, and $T_r(t)$, where the output of D_t is 1;
4. (a) Add to t a left node t_l; if $T_l(t)$ contains only patterns with label l_j for any j, then t_l is a leaf with training set $T(t_l) = \emptyset$, otherwise it is an internal node with training set $T(t_l) = T_l(t)$;
   (b) Add to t a right node t_r; if $T_r(t)$ contains only patterns with label l_j for any j, then t_r is a leaf with training set $T(t_r) = \emptyset$, otherwise it is an internal node with training set $T(t_r) = T_r(t)$;
5. repeat the same algorithm for t_l and t_r, starting from step 2.

The discriminators can be implemented by a simple perceptron or by a feed-forward network.

Classification Algorithm

The class of a pattern $\boldsymbol{I}_k$ can be established by the following algorithm:

1. Set $t = root$;
2. If the node t is a leaf, classify $\boldsymbol{I}_k$ by the associated label l_k and stop, otherwise:
   (a) If $D_t(\boldsymbol{I}_k) = 0$ then set $t = t_l$;
   (b) otherwise set $t = t_r$;
3. go to step 2.

Figure 15: The Learning and Classification Algorithms for Binary Neural Trees.

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)
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)
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)
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 a set of classification problems involving logic terms.

It must be noted that the set of proposed problems range from the detection of a particular atom (label) in a term to the satisfaction of a specific unification pattern. Specifically, in the unification patterns for the problems `inst1`, and `inst1_long` the variable X occurs twice making these problems much more difficult than `inst4_long`, because any classifier for these problems would have to compare arbitrary subterms corresponding to X.

5.2 LRAAM-based networks

Best Results for an LRAAM-based Network

Problem	#L-unit	#H-unit	Learning Par.	% Dec.-Enc.	% Tr.	% Ts.	#epochs
lblocc1 long	8	35	$\eta = 0.2,\ \epsilon = 0.001,\ \mu = 0.5$	4.25	100	98.58	11951
termocc1	8	25	$\eta = 0.1,\ \epsilon = 0.1,\ \mu = 0.2$	100	98.84	94.29	27796
inst1	6	35	$\eta = 0.2,\ \epsilon = 0.06,\ \mu = 0.5$	100	97	93.98	10452
inst1 long	6	45	$\eta = 0.2,\ \epsilon = 0.005,\ \mu = 0.5$	36.14	94.55	90.82	80000
inst4	6	35	$\eta = 0.2,\ \epsilon = 0.005,\ \mu = 0.5$	98.86	100	100	1759
inst4 long	6	35	$\eta = 0.2,\ \epsilon = 0.005,\ \mu = 0.5$	8.97	100	100	6993
inst7	13	40	$\eta = 0.1,\ \epsilon = 0.01,\ \mu = 0.2$	1.05	100	100	6158

Table 2: The best results obtained for almost all the classification problems by an LRAAM-based network.

In Table 2, we have reported the best result we obtained for almost all the problems described in Table 1, over 4 different network settings (both in number of hidden units for the LRAAM and learning parameters) for the LRAAM-network with a single unit as classifier. The simulations were stopped after 30,000 epochs, apart for problem `inst1_long` for which we used a bound of 80,000 epochs, or when the classification problem over the training set was completely solved. We made no extended effort for optimizing the size of the network and the learning parameters, thus it should be possible to improve on the reported results. The first column of the table shows the name of the problem, the second one the number of units used to represent the labels, the third the number of hidden units, the fourth the learning parameters (η is the learning parameter for the LRAAM, ϵ the learning parameter for the classifier, μ the momentum for the LRAAM), the fifth the percentage of terms in the training set which the LRAAM was able to properly encode and decode, the sixth the percentage of terms in the training set correctly classified, the seventh the percentage of terms in the test set correctly classified, and the eighth the number of epochs the network emploied to reach the reported performances.

From the results, it can be noted that some problems get a very satisfactory solution even if the LRAAM performs poorly. Moreover, this behavior does not seem to be related with the complexity of the classification problem, since both problems involving the simple detection of an atom (label) in the terms (`lblocc1_long`) and problems involving the satisfaction of a specific unification rule (`inst4_long, inst7`) can be solved without the need of a fully developed LRAAM. Thus, it is clear that the classification of the terms is exclusively based on the encoding power of the LRAAM's encoder which is shaped both by the LRAAM error and the classification error. However, even if the LRAAM's decoder is not directly involved in the classification task, it helps the classification process since it forces the network to generate different representations for terms in different classes[13].

In order to give a feeling of how learning proceeds, the performance of the networks during training is shown for the problems `termocc1`, `inst4` and `inst4_long` in Figures 16-18, where the encoding-decoding performance curve of the LRAAM on the training set is reported, together with the classification

[13]Actually, the decoder error forces the LRAAM network to develop a different representation for each term, however, when the error coming from the classifier is very strong, it can happen that terms in the same class get almost identical representations.

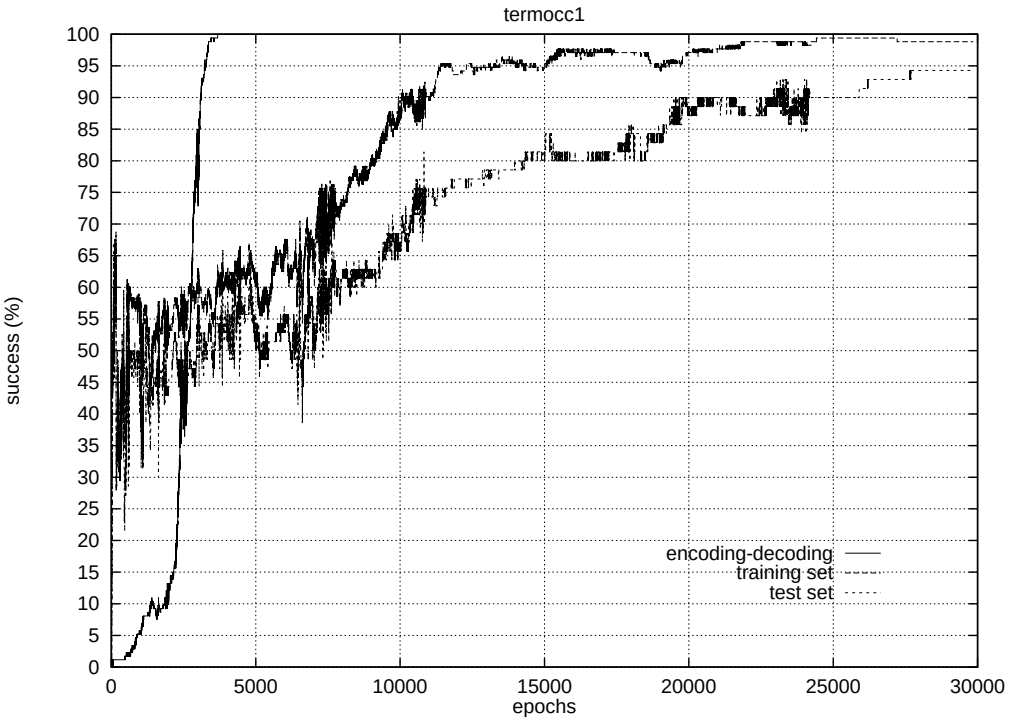


Figure 16: Performance curves for `termocc1`.

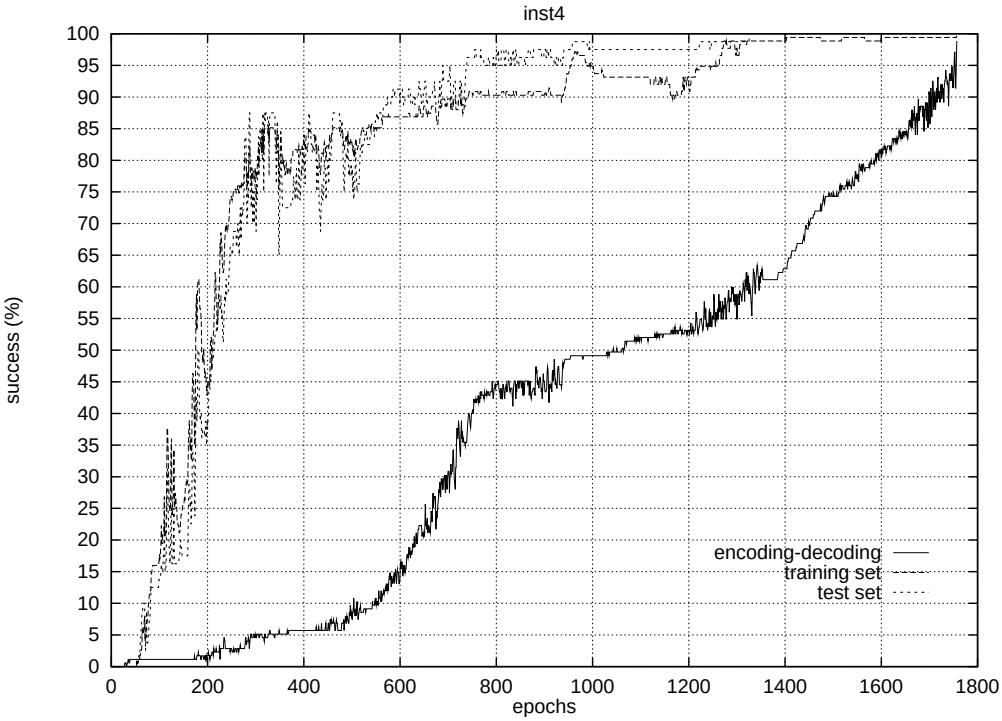


Figure 17: Performance curves for `inst4`.

curves on the training and test set.

5.2.1 Reduced Representations for Classification

In this section, we briefly discuss the representational differences between a basic LRAAM (without classifier) and the architecture we used. The basic LRAAM organizes the representational space in such a way that similar structures get similar reduced representations (see [Spe94b] for more details). This happens because, even if the LRAAM is trained in supervised mode both over the output of the network and over the relationships among components (i.e., the information about the pointers), the network is auto-associative and thus it decides by itself the representation for the pointers. Consequently, the learning mode for the LRAAM is, mainly, unsupervised. When a classifier is introduced (as in our system), the training of the LRAAM is no longer mainly unsupervised, since the error of the classifier constrains the learning. The resulting learning regime is somewhat between an unsupervised and a supervised mode.

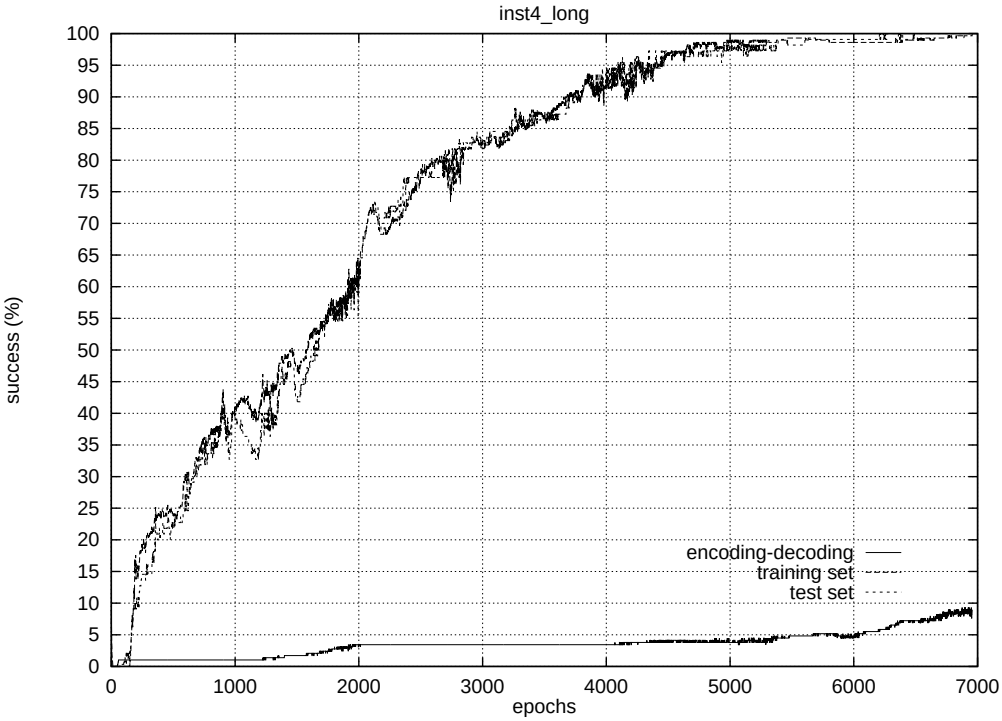


Figure 18: Performance curves for `inst4_long`.

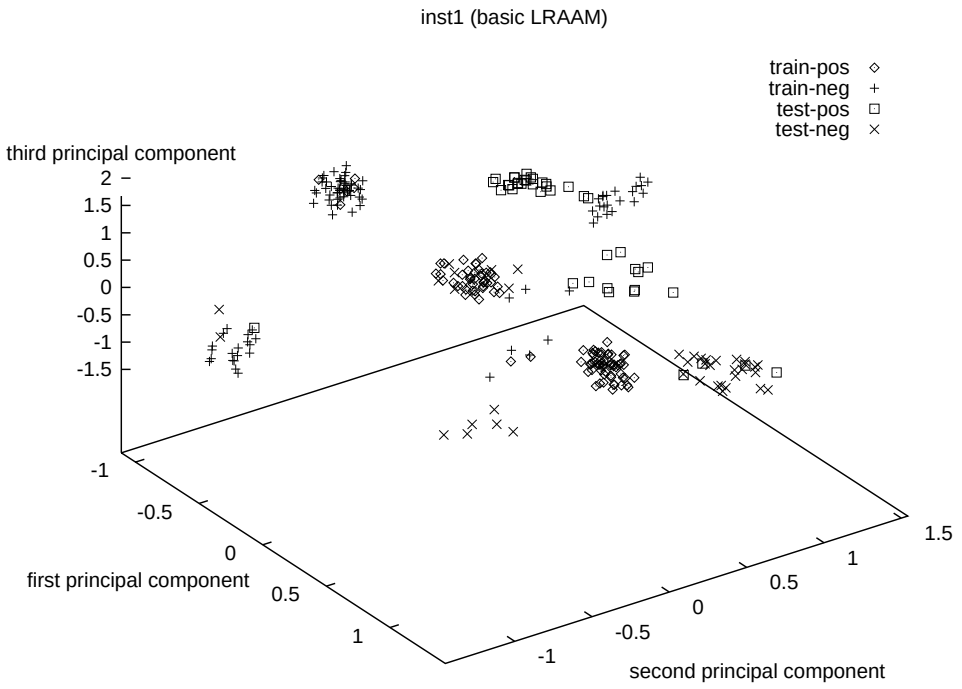


Figure 19: The first, second, and third principal component of the reduced representations, devised by a basic LRAAM on the training and test sets of the `inst1` problem, yield a nice 3D view of the term's representations.

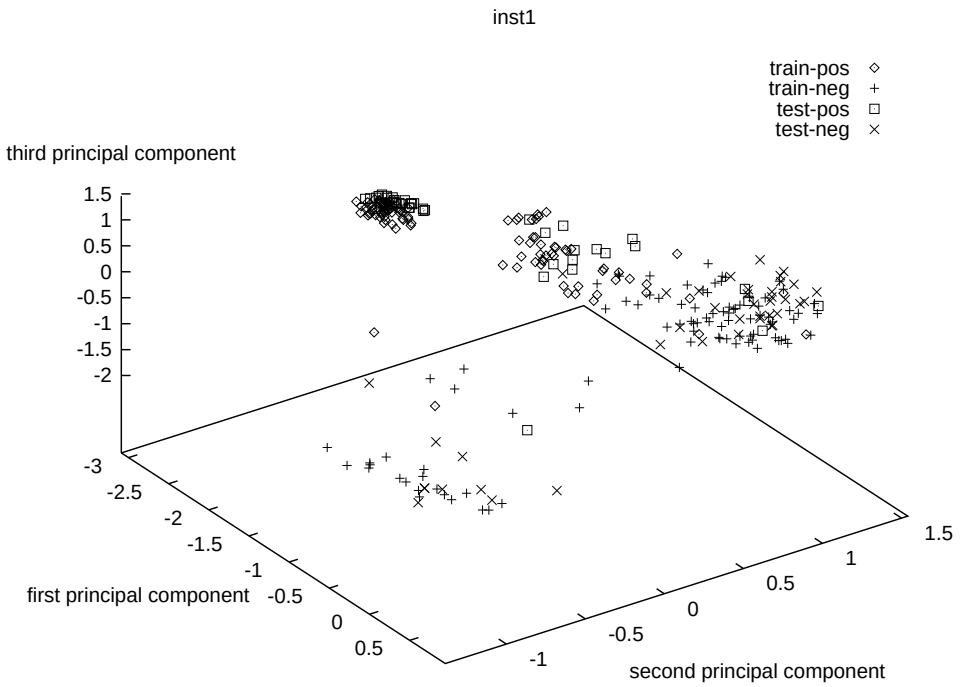


Figure 20: Results of the principal components analysis (first, second, and third principal component) of the reduced representations developed by the proposed network (LRAAM + Classifier) for the `inst1` problem.

In order to understand the differences between representations devised by a basic LRAAM and the ones devised in the present paper, we trained a basic LRAAM (of the same size of the LRAAM used by our network) over the training set of `inst1`, then we computed the first, second, and third principal component of the reduced representations obtained both for the training and test set[14]. These principal components are plotted in Figure 19. It can be noted that the obtained representations mainly cluster themselves in specific points of the space. Terms of the same depth constitute a single cluster, and terms of different depth are in different clusters. The same plot for the reduced representations devised by our network (as from Table 2, row 3) is presented in Figure 20. The overall differences with respect to the basic LRAAM plot consists in a concentration of more than half (57%) of the positive examples of the training and test sets in a well defined cluster, while the remaining representations are spread within two main subspaces. The well defined cluster can be understood as the set of representations for which there was no huge interference between the decoder of the LRAAM and the classifier (this allowed the formation of the cluster), while the remaining representations do not preserve the cluster structure since they have to satisfy competitive constraints coming from the classifier and the decoder. Specifically, the classifier tends to cluster the representations into two well defined clusters (one for each class), while the LRAAM decoder tends to develop well distinct reduced representations since they must be decoded to different terms.

The above considerations on the final representations for the terms are valid only if the LRAAM reaches a good encoding-decoding performance on the training set. However, as we have reported in Table 2, some classification problems can be solved even if the LRAAM performs poorly. In this case, the reduced representations contain almost exclusively information about the classification task. In Figure 21 and Figure 22 we have reported the results of a principal components analysis on the representations developed for the problems `inst4_long` and `inst7`, respectively. In the former, the first and second principal components suffice for a correct solution of the classification problem. In the latter, the second principal component alone gives enough information for the solution of the problem. Moreover, notice how the representations developed for `inst7` clustered with smaller variance than the representations

[14]We considered only the reduced representations for terms. No reduced representation for subterms was included in the analysis.

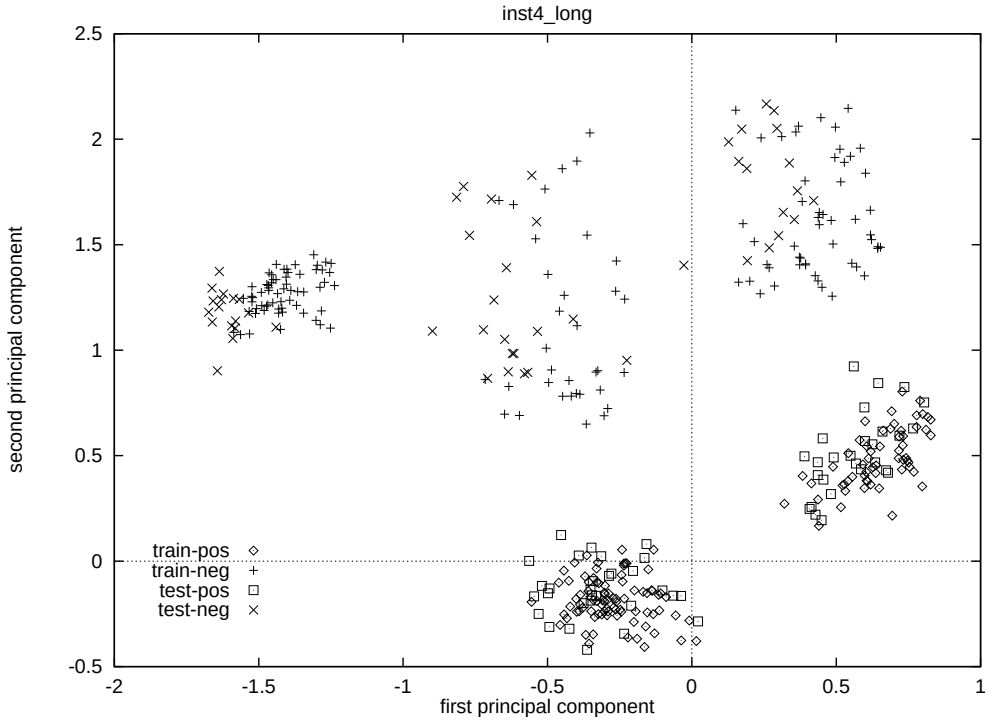


Figure 21: Results of the principal components analysis (first, and second principal component) of the reduced representations developed by the proposed network (LRAAM + Classifier) for the `inst4-long` problem. The resulting representations are clearly linearly separable.

developed for `inst4_long`, and how this is in accordance with the better performance in encoding-decoding of the latter than the former. Of course, this does not constitute enough evidence for concluding that the relationship between the variance of the clusters and the performance of the LRAAM is demonstrated. However, it seems to be enough for calling a more accurate study on this issue.

5.3 Cascade-Correlation for Structures

The results obtained by Cascade-Correlation for structures, shown in Table 3, are obtained for a subset of the problems using a pool of 8 units. The networks used have both triangular and diagonal recursive connections matrices and **no**

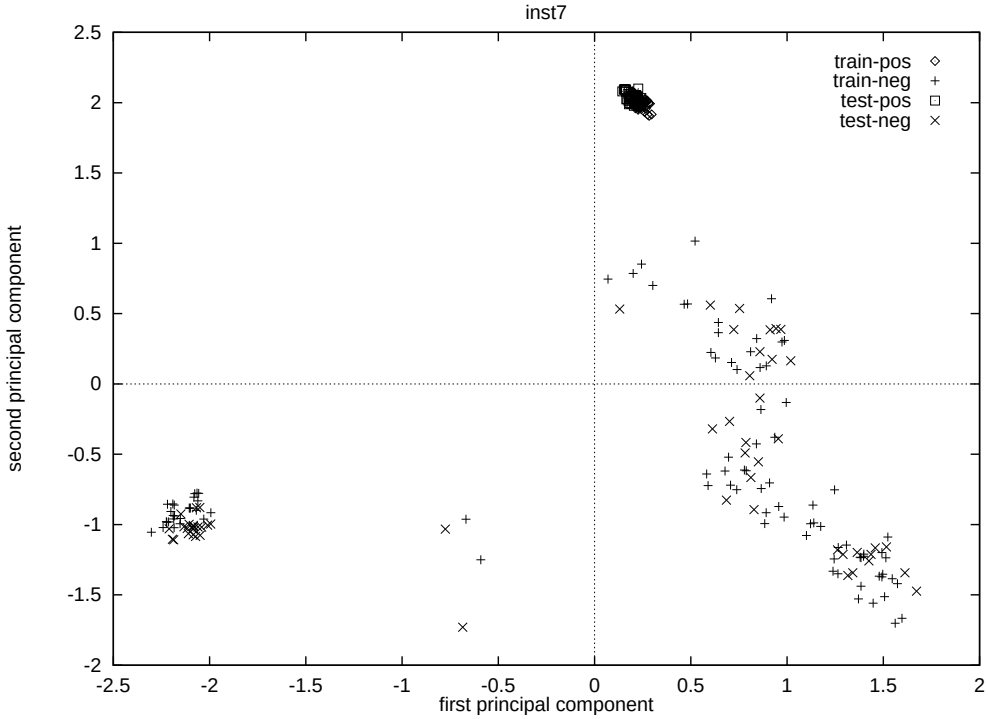


Figure 22: Results of the principal components analysis (first, and second principal component) of the reduced representations developed by the proposed network (LRAAM + Classifier) for the `inst7` problem. The resulting representations can be separated using only the second principal component.

Results

(Networks without connections between hidden units)

	Problem	# Label Units	# Trials	# Hidden Units Mean (Min - Max)	% Test Mean (Min - Max)
Triangular	termocc1 very-long	8	4	13.25 (8 - 19)	97.70 (95 - 100)
	inst4 long	6	5	7.2 (4 - 12)	99.64 (99.09 - 100)
	inst1	6	9	11.33 (7 - 19)	90.09 (86.75 - 92.77)
	inst1 long	6	16	7.81 (6 - 11)	91.65 (88.87 - 94.89)
Diagonal	termocc1 very-long	8	3	11 (8 - 16)	96.94 (95 - 99.17)
	inst4 long	6	3	12.66 (9 - 15)	98.48 (97.27 - 100)
	inst1	6	5	12.4 (7 - 21)	90.6 (87.95 - 92.77)
	inst1 long	6	3	18.66 (17 - 21)	82.99 (75.51 - 91.84)

Table 3: Results obtained on the test sets for each classification problem using both networks with triangular and diagonal recursive connection matrices. The same number of units (8) in the pool was used for all networks.

connection between hidden units. We decided to remove the connections between hidden units to reduce the probability of overfitting.

We made no extended effort for optimizing the learning parameters and the number of units in the pool, thus it should be possible to significantly improve on the reported results.

6 Conclusion

We have proposed a generalization of the standard neuron, namely the complex recursive neuron, for extending the computational capability of neural networks to processing of structures. On the basis of the complex recursive

neuron, we have shown how several of the learning algorithms defined for standard neurons, can be adapted to deal with structures. We believe, that other learning procedures, which are not covered by this report, can be adapted as well.

The proposed approach to learning in structured domains can be adopted for automatic inference in syntactic and structural pattern recognition. Specifically, in this report, we demonstrated the possibility to perform classification tasks involving logic terms. It must be noted that automatic inference can also be obtained by using *Inductive Logic Programming* [MR94]. The proposed approach, however, has its own specific peculiarity, since it can approximate functions from a structured domain (possibly with real valued vectors as labels) to the reals. Specifically, we believe that the proposed approach can fruitfully be applied to molecular biology and chemistry (classification of chemical structures, quantitative structure-property relationship (QSPR), quantitative structure-activity relationship (QSAR)), where it can be used for the automatic determination of *topological indexes* [Rou90], which are usually designed through a very expensive trial and error approach.

In conclusion, the proposed architectures extends the processing capabilities of neural networks, allowing the processing of structured patterns which can be of variable size and complexity. However, it must be pointed out that same of the proposed architectures have computational limitations. For example, Cascade-Correlation for structures has computational limitations due to the fact that frozen hidden units cannot receive input from hidden units introduced after their insertion into the network. These limitations, in the context of standard Recurrent Cascade-Correlation (RCC), have been discussed in [GCS+95], where it is demonstrated that certain finite state automata cannot be implemented by networks built up by RCC using monotone activation functions. Since our algorithm reduces to standard RCC when considering sequences, it follows that it has limitations as well.

Acknowledgment

We would like to thank Christoph Goller for the generation of the training and test sets used in this paper.

References

[Aa92] L. Atlas and al. A performance comparison of trained multilayer perceptrons and trained classification trees. *Proceedings of the IEEE*, 78:1614–1619, 1992.

[Alm87] L.B. Almeida. A learning rule for asynchronous perceptrions with feedback in a combinatorial environment. In M. Caudil and C. Butler, editors, *Proceedings of the IEEE First Annual International Conference on Neural Networks*, pages 609–618. CA: IEEE, 1987.

[Ati88] A. Atiya. Learning on a general network. In D.Z. Anderson, editor, *Neural Information Processing Systems*, pages 22–30. New York: AIP, 1988.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[CG83] M. A. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. on Systems, Man, and Cybernetics*, 13:815–826, 1983.

[Elm90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[Fah91] S. E. Fahlman. The recurrent cascade-correlation architecture. Technical Report CMU-CS-91-100, Carnegie Mellon, 1991.

[FL90] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. San Mateo, CA: Morgan Kaufmann, 1990.

[GCS+95] C.L. Giles, D. Chen, G.Z. Sun, H.H. Chen, Y.C. Lee, and M.W. Goudreau. Constructive learning of recurrent neural networks: Limitations of recurrent cascade correlation and a simple solution. *IEEE Transactions on Neural Networks*, 1995. To appear.

[GK95] C. Goller and A. Küchler. Learning Task-Dependent Distributed Structure-Representations by Backpropagation Through Structure. AR-report AR-95-02, Institut für Informatik, Technische Universität München, 1995.

[Hop84] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. In *Proc. Natl. Acad. Sci.*, pages 3088–3092, 1984.

[LFJ92] T. Li, L. Fang, and A. Jennings. Structurally adaptive self-organizing neural trees. In *International Joint Conference on Neural Networks*, pages 329–334, 1992.

[MR94] S. Muggleton and L. De Raedt. Inductive login programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[Per92] M. P. Perrone. A soft-competitive splitting rule for adaptive tree-structured neural networks. In *International Joint Conference on Neural Networks*, pages 689–693, 1992.

[PI92] M. P. Perrone and N. Intrator. Unsupervised splitting rules for neural tree classifiers. In *International Joint Conference on Neural Networks*, pages 820–825, 1992.

[Pin88] F. J. Pineda. Dynamics and architecture for neural computation. *Journal of Complexity*, 4:216–245, 1988.

[Pol90] J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–106, 1990.

[Rou90] D. H. Rouvray. *Computational Chemical Graph Theory*, page 9. Nova Science Publishers: New York, 1990.

[Set90] I. K. Sethi. Entropy nets: From decision trees to neural networks. *Proceeding of the IEEE*, 78:1605–1613, 1990.

[SM91] A. Sankar and R. Mammone. *Neural Tree Networks*, pages 281–302. Neural Networks: Theory and Applications. Academic Press, 1991.

[SN90] J. A. Sirat and J-P. Nadal. Neural trees: a new tool for classification. *Network*, 1:423–438, 1990.

[Spe94a] A. Sperduti. Encoding of Labeled Graphs by Labeling RAAM. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 1125–1132. San Mateo, CA: Morgan Kaufmann, 1994.

[Spe94b] A. Sperduti. Labeling RAAM. *Connection Science*, 6(4):429–459, 1994.

[Spe95] A. Sperduti. Stability properties of labeling recursive auto-associative memory. *IEEE Transactions on Neural Networks*, 6(6):1452–1460, 1995.

[SSG94] A. Sperduti, A. Starita, and C. Goller. Fixed length representation of terms in hybrid reasoning systems, report i: Classification of ground terms. Technical Report TR-19/94, Dipartimento di Informatica, Università di Pisa, 1994.

[SSG95] A. Sperduti, A. Starita, and C. Goller. Learning distributed representations for the classification of terms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 509–515, 1995.

[WZ89] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.