A General Framework for Adaptive Processing of Data Structures

Paolo Frasconi, *Member, IEEE*, Marco Gori, *Senior Member, IEEE*, and Alessandro Sperduti

***Abstract*—A structured organization of information is typically required by symbolic processing. On the other hand, most connectionist models assume that data are organized according to relatively poor structures, like arrays or sequences. The framework described in this paper is an attempt to unify adaptive models like artificial neural nets and belief nets for the problem of processing structured information. In particular, relations between data variables are expressed by directed acyclic graphs, where both numerical and categorical values coexist. The general framework proposed in this paper can be regarded as an extension of both recurrent neural networks and hidden Markov models to the case of acyclic graphs. In particular we study the supervised learning problem as the problem of learning transductions from an input structured space to an output structured space, where transductions are assumed to admit a recursive hidden state-space representation. We introduce a graphical formalism for representing this class of adaptive transductions by means of recursive networks, i.e., cyclic graphs where nodes are labeled by variables and edges are labeled by generalized delay elements. This representation makes it possible to incorporate the symbolic and subsymbolic nature of data. Structures are processed by unfolding the recursive network into an acyclic graph called encoding network. In so doing, inference and learning algorithms can be easily inherited from the corresponding algorithms for artificial neural networks or probabilistic graphical model.**

***Index Terms*—Graphical models, graphs, learning data structures, problem-solving, recurrent neural networks, recursive neural networks, sequences, syntactic pattern recognition.**

I. Introduction

THE integration of symbolic and subsymbolic systems is a fundamental research topic for the development of intelligent and efficient systems capable of dealing with tasks whose nature is neither purely symbolic nor subsymbolic. It is common opinion in the scientific community that quite a wide variety of real-world problems require hybrid solutions, i.e., solutions combining techniques based on neural networks, fuzzy logic, genetic algorithms, probabilistic networks, expert systems, and other symbol-based techniques. A very popular view of hybrid systems is one in which numerical data are processed by a subsymbolic module, while structured data are processed by the symbolic counterpart of the system. Unfortunately, because of the different nature of numerical and structured representations, a tight integration of the different components seems to be very difficult.

In some approaches to hybrid systems, the role of the prior knowledge is that of providing a partial specification of the transduction to be learned. Although interesting and promising, some approaches to the incorporation of symbolic knowledge into adaptive models, like neural networks, seem to be inherently limited by the complementary role played by learning and symbolic knowledge: the more symbolic rules are injected, the harder the learning becomes [1].

In this paper we propose a different view of hybrid systems, in which the incorporation of symbolic knowledge does not involve primarily the desired input–output transduction, but the nature of the data themselves. There are a number of different application domains in which data are strongly structured, and where the processing cannot ignore the topological information expressing relations among different portions of the data. Most of the times, in real-word problems data are not only significantly structured, but many composing features have a subsymbolic nature. As will be put forward in Section I-B, adaptive models typically used for processing arrays and sequences are not adequate to process such complex data structures.

We show that structured information can be represented and processed in a framework which is amenable to both neural and belief networks. The possibility to represent and process structures in a neural and/or probabilistic fashion greatly increases the potential of integration between subsymbolic and symbolic components in a hybrid system.

In the remainder of this section we argue for the relevance of structured information in several application domains and then motivate the formulation of a general framework for adaptive computation of data structures. In Section II, we formalize structured learning domains by means of directed acyclic graphs, where both numerical and categorical values coexist. A similar data organization seems to be very common in a number of different application domains, briefly sketched in Section I-A. In Section III, we introduce deterministic and probabilistic structural transductions, with particular reference to recursive state-space representations and their associated graphical models. In Section IV, we present two classes of adaptive models for structural processing: in the deterministic setting, the temporal processing which takes place in recurrent neural networks is extended to the case of graphs by connec-

Manuscript received March 15, 1997; revised November 17, 1997. This work was supported in part by Italian MURST and by the Italian National Research Council.

P. Frasconi is with Dipartimento di Sistemi e Informatica, Università di Firenze, 50139 Firenze, Italy.

M. Gori is with Dipartimento di Ingegneria dell'Informazione, Università di Siena, 53100 Siena, Italy.

A. Sperduti is with Dipartimento di Informatica, Università di Pisa, Pisa, Italy.

Publisher Item Identifier S 1045-9227(98)06190-6.

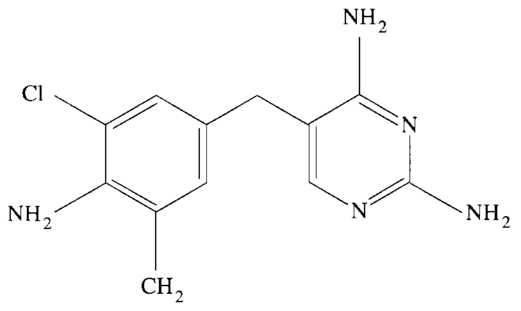

Fig. 1. Typical chemical compound, naturally represented by an undirected graph.

tionist models, that we refer to as *recursive neural networks*,[1] while in the probabilist setting, hidden Markov models are extended to *hidden recursive models*. Applications of adaptive recursive processing are reviewed in Section V. Finally, some guidelines for further development of the theory proposed in this paper are outlined in Section VI.

A. Learning from Structured Information: Application Domains

In several application domains, the information which is relevant for solving a given problem is encoded, sometimes implicitly, into the relationships between basic entities.

Example 1.1 Chemistry: Chemical compounds are usually represented as undirected graphs. Each node of the graph is an atom or a group of atoms, while arcs represent bonds between atoms (see Fig. 1).

One fundamental problem in chemistry is the prediction of the biological activity of chemical compounds. quantitative structure-activity relationship (QSAR) is an attempt to face the problem relying on compound structures. The biological activity of a drug is fully determined by the micromechanism of interaction of the active molecules with the bioreceptor. Unfortunately, discovering this micromechanism is very hard and expensive. Hence, because of the assumption that there is a direct correlation between the activity and the structure of the compound, the QSAR approach is a way of approaching the problem by comparing the structure of all known active compounds with inactive compounds, focusing on similarities and differences between them. The aim is to discover which substructure or which set of substructures characterize the biomechanism of activity, so as to generalize this knowledge to new compounds.

Example 1.2 Software Engineering: Another very important example of an application which uses *structured* information is certainly software engineering. One of the major goals of software engineering is to evaluate the quality of the software. This evaluation is usually based on metrics that are correlated with properties of interest. A number of metrics (see, e.g., McCabe complexity [2]) have been developed which try to codify the above properties of a (portion of) program numerically. These features are usually based on an

[1]As detailed in the following, *recurrent* neural networks and recursive neural networks reduce to the same model when the domain is restricted to sequences. For historical reasons, however, we shall use the name recurrent neural networks when referring to models operating on sequences.

```
program name (list);
var
. . .
begin
if T1 then
      a
else
      b;
c;
while T2 do
      begin
      d;
      if T3 then
      while T4 do
            e
      end;
f
end.
```

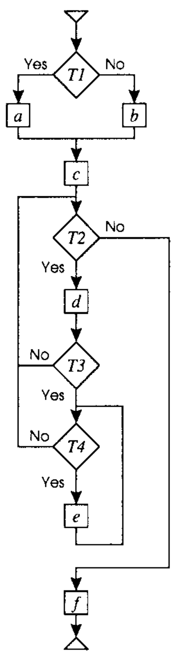

Fig. 2. A portion of software code with the corresponding flowgraph. Metrics for the software evaluation turn out to be functions acting on graph-based domains.

intermediate representation which has the advantage of being (in some sense) independent of the specific language, while preserving the essential static and dynamic aspects of the program. One example of intermediate representation is given by *dependence graphs*. In a dependence graph, statements are represented as nodes, while directed edges are used to represent the statement ordering implied by the dependencies in a source program. Depending on the specific application, different kinds of dependence graphs can be used (e.g., *control flow* graphs, *control* dependence graphs, *data* dependence graphs, and *instance* dependence graphs).

It is commonly accepted that most procedural languages can be expressed as a flowgraph using a number of basic elements, such as *decision node, junction node, and begin* and *end node* [3]. Let $\mathcal{F}^{\#}$ be the set of flowgraphs derived by all possible programs (see, e.g., Fig. 2). A *software metric* $m(\cdot)$ is a function $m(\cdot)$: $\mathcal{F}^{\#} \rightarrow \mathbb{N}^{+}$ used to estimate the complexity of a portion of software. The aim is to use $m(\cdot)$ as an indicator of the quality, testability, reusability, and maintainability of the program.

Example 1.3 Problem Solving in Artificial Intelligence: A rich source of applications based on structured information are those related to problem solving in artificial intelligence. One often has to perform a search in a tree which typically gives rise to a combinatorial explosion of the search space. Examples of systems based on such expensive search are theorem provers, deductive databases, and expert systems. For all these systems we should search for a solution, or a proof, by exploring every branch of the search tree defined by the problem at hand (see, e.g., Fig. 13 in Section V-B). Exhaustive search guarantees completeness, i.e., if there is a solution, it will be found within finite time. This brute force approach, however, is only practical and feasible for problems of small

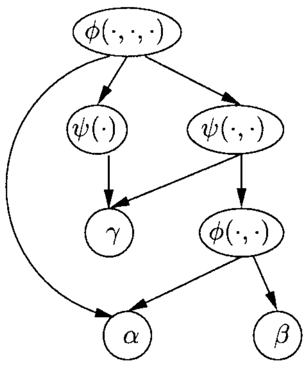

Fig. 3. A directed acyclic graph representing the logical term $\phi(\alpha, \psi(\gamma), \psi(\gamma, \phi(\alpha, \beta)))$.

dimensions. When facing larger problems, it is commonly recognized that some heuristics aimed at identifying the most promising paths of the search tree speed up the search significantly. A well-known search algorithm guided by heuristics is A* [4], that uses the dynamic programming principle for seeking an optimal-cost path. A heuristic evaluation function computes ratings for inference steps or for states in search-trees. These ratings are then used to select the next inference step which must be performed, or the next state to be explored. In order to be useful heuristics should be simple and it should work for the majority of cases, i.e., it can be understood as *approximation* to a complete perfect search-guiding strategy (oracle).

Unfortunately, heuristics are very expensive to be devised since they typically summarize the knowledge of an expert of a given domain. Moreover, they are too specific, so that even a slight change in the domain may require a new heuristic to be devised from scratch. A way to overcome these problems is to learn a heuristic in a supervised fashion from data samples (i.e., states and inference steps and their ratings) obtained by solutions which are already been found. Several tasks that must be performed for learning a control heuristic, like finding a rating for the applicable rules according to the current context or selecting the next subgoal, can be regarded as problems of learning a classification of logical terms [5]. In fact, positive samples are states or inference steps on solution-paths within the search-tree, while negative samples are states or inference steps on failure-paths.

Terms in first-order logic can be easily represented as a directed acyclic graphs, as shown in Fig. 3. Here vertices are labeled by function names and edges are used to link functions to their arguments. Constants (like α, β, γ) are considered to be functions with zero arity and, therefore, are always found on leaf vertices. This example will be used sometimes in the following, to explain concepts and definitions intuitively in the proposed computational scheme.

Example 1.4 Pattern Recognition: Pattern recognition is another source of applications in which one may be interested in adaptive processing of data structures. This was recognized early with the introduction of syntactic and structural pattern recognition [6]–[9], that are based on the premise that the *structure* of an entity is very important for both classification and description.

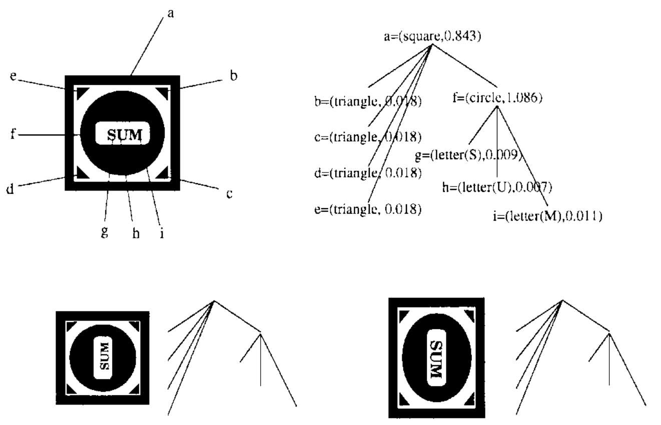

Fig. 4. A logo with the corresponding representation based on both symbolic and subsymbolic information.

Fig. 4 shows a logo with a corresponding structural representation based on a tree, whose nodes are components properly described in terms of geometrical features. This representation is invariant with respect to roto-translations and naturally incorporate both symbolic and numerical information.

Of course, the extraction of robust representations from patterns is not a minor problem. The presence of a significant amount of noise is likely to significantly affect representations that are strongly based on symbols. Hence, depending on the problem at hand, the structured representation that we derive should emphasize the symbolic or the subsymbolic information. For example, the logo shown in Fig. 4 could be significantly corrupted by noise so as to make it unreasonable to recognize the word "SUM." In that case, one should just consider all the words as subsymbolic information collected in a single node.

B. Motivations and Related Approaches

The common feature shared between the application domains sketched in the previous section is that the required predictions should be based neither on simple arrays of features nor on sequences, but on dynamic data structures incorporating also numerical information. One could argue that, sometimes, machine learning models conceived for dealing with sequences can be straightforwardly adapted to process data structures. For instance, the processing of binary trees by recurrent neural networks or hidden Markov models can take place on sequential representations based on traversing the trees.[2] This approach, however, has two major drawbacks. First, since the number of nodes grows exponentially with the height of the trees, even short trees give rise to long sequences, thus making learning very hard.[3] Second, the sequential mapping of data structures is likely to break some nice regularities

[2]A representation based on nested parenthesis is a way of creating a sequential representation that makes it possible to reconstruct the tree. Another way of providing a unique sequential representation of binary trees is that of considering both the inorder and preorder visits.

[3]For recurrent neural networks this is the well-known problem of learning long-term dependencies [10].

inherently associated with the data structure, thus making the generalization very hard.

In the last few years, some interesting approaches to the representation and processing of structured information have been proposed in the field of connectionist models. Hinton [11] has introduced the concept of distributed reduced descriptors in order to allow neural networks to represent compositional structures. Concrete examples of distributed reduced descriptors are the recursive autoassociative memory (RAAM) by Pollack [12] and the holographic reduced representations by Plate [13]. More recently, the labeling RAAM model (LRAAM) [14]–[16] has been proposed as an extension of RAAM's, while some advances on the LRAAM access by content capabilities has been discussed in [17]. LRAAM's make it possible to carry out the synthesis of distributed reduced descriptors for fixed outdegree directed labeled graphs. In the field of natural language processing, very good results on the classification of distributed representations of syntactical trees devised by an LRAAM according to the typology of dialogue acts were obtained by Cadoret [18].

II. Definitions and Background Topics

A. Structured Domains

Instances in the learning domain are structured pieces of information described by annotated directed ordered acyclic graphs (DOAG's). Here by a DOAG we mean a DAG D with vertex set vert(D) and edge set edg(D), where for each vertex $v \in \mathrm{vert}(D)$ a total order on the edges leaving from v is defined. For example, in the case of graphs representing logical terms (see Fig. 3), the order on outgoing edges is immediately induced by the order of the function arguments. In problems of structure classification, we shall require the DOAG either to be empty or to possess a supersource, i.e., a vertex $s \in \mathrm{vert}(D)$ such that every vertex in vert(D) can be reached by a directed path starting from s. The reasons for this requirement are related to the processing scheme that will be defined in Section III. Note that if a DOAG does not possess a supersource, it is still possible to define a convention for adding an extra vertex s (with a minimal number of outgoing edges), such that s is a supersource for the expanded DOAG [19].

Given a DOAG D and $v \in \mathrm{vert}(D)$, we denote by an$[v]$ the set of parents of v, by ch$[v]$ the set of children of v, by de$[v]$ the set of descendants of v, and by pa$[v]$ the set of ancestors of v. The *indegree* of v is the cardinality of the set pa$[v]$ the *outdegree* of v is the cardinality of the set ch$[v]$. In the following, we shall denote by $\#^{(i,o)}$ the class of DOAG's having maximum indegree i and maximum outdegree o. In our logical terms example (see Fig. 3), the maximum outdegree corresponds to the maximum arity of the functions being considered. A generic class of DOAG's with bounded (but unspecified) indegree and outdegree, will simply be denoted by #.

Graphs used for storing structured information are *marked*, or *labeled*, in the sense that vertices and edges contain sets of domain variables, called *labels*. Vertices or edges containing an empty set of variables are said to be unlabeled. We assume that all the labels in a graph are disjoint sets. The domain variables contained into labels are also called *attributes*. In general, some attributes are numerical (i.e., they take on continuous values) and some categorical (i.e., they take on discrete or symbolic values). The presence of an edge (v, w) in a marked graph indicates that the variables contained in v and w are related is some way. If the edge (v, w) is labeled, then the variables in (v, w) characterize the relationship between variables in v and variables in w. Graphs with edge labels, however, can be reduced to graphs having only labels on the nodes. A straightforward method for reducing structures with labeled edges to structures with unlabeled edges is to move each label attached to an edge leaving a given node v into the label attached to node v.

Assume that an additional equivalence relation is defined among domain variables, where variables within an equivalence class have the same type and the same semantics. If this is the case, then we say that two labels are *similar* if they contains at most one element from each equivalence class and intersect the same subset of equivalence classes. A graph is *uniformly labeled* if all its labels are similar. For example, in the logo recognition problem described in Fig. 4, labels at each vertex are the variables perimeter, area, and shape, as measured from the corresponding image element. Perimeter, area, and shape are equivalence classes for the whole set of domain attributes. In this case the tree is uniformly labeled. Note that the first two attributes are numerical, while the last is categorical. Unless explicitly stated, throughout the paper we assume that graphs are uniformly labeled. Under this assumption, we may take one abstract representative for each equivalence class and form a set of abstract representatives for the domain variables, that correspond to the set of attribute names (e.g., {Shape, Perimeter, Area} in the logo example). Labels are therefore fixed-size tuples of attributes. Our general notation for labels is defined as follows. Let N be the number of equivalence classes. In this context N is called *label size*. Assume a conventional order and denote by Y_i the representative for the ith equivalence class (how the order is chosen does not really matters, since it is only used as a notation for distinguishing attribute names). Then the abstract set of representatives is denoted as $\{Y_1, \cdots, Y_N\}$. Following standard notation, we shall use uppercase letters for variables and lowercase letters for realizations. If Y_i is a categorical variable, the set of admissible states (or alphabet) for Y_i will be denoted by $\mathcal{Y}_i = \{y_i^1, \cdots, y_i^{r_i}\}$. If Y_j is a numerical variable, we shall assume that realizations of Y_j are real numbers, i.e., $y_j \in \mathcal{Y}_j = \mathbb{R}$. Let $\mathcal{Y} = \mathcal{Y}_1 \times \mathcal{Y}_2 \times \cdots \times \mathcal{Y}_N$ be the set of all possible realizations for labels. This set will be referred to as the *label space* or label *domain*. Label spaces will be denoted by calligraphic letters. The size of the label space $\mathcal{Y}$ will be denoted $|\mathcal{Y}|$.

A uniformly labeled DOAG will be denoted by the boldface uppercase letter corresponding to the label space of the graph. So, for example, $\boldsymbol{Y}$ denotes a DOAG with labels in $\mathcal{Y}$. Labels are accessed by vertex subscripts: $\boldsymbol{Y}_v$ denotes the label attached to vertex v. If A is an (ordered) set of vertices, then $\boldsymbol{Y}_A$ denotes the (ordered) set of labels attached to vertices in A.

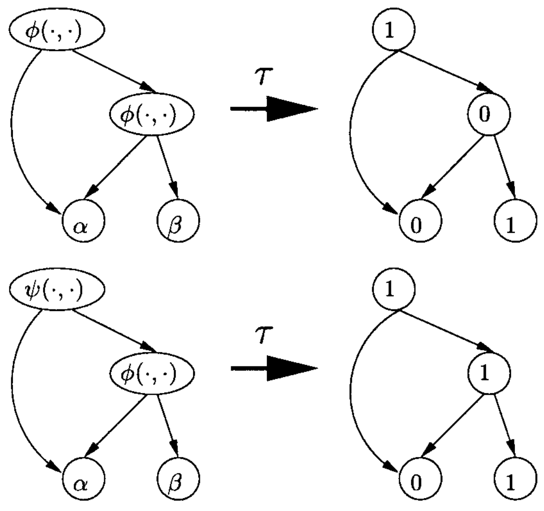

Fig. 5. Example of IO-isomorph transduction in the logical terms domain.

Given a data structure $\boldsymbol{Y}$, the DOAG obtained by ignoring all node labels will be referred to as the *skeleton* of $\boldsymbol{Y}$, denoted skel$(\boldsymbol{Y})$. Clearly, any two data structures can be distinguished because they have different skeletons, or, if they have the same skeleton, because they have different node labels. The class of all data structures defined over the local universe domain $\mathcal{Y}$ and skeleton in $\#^{(i,o)}$ will be denoted as $\mathcal{Y}^{\#^{(i,o)}}$.

In the following, we give a few examples of structured domains and briefly recall how learning problems on these domains can be approached. It turns out that most nonsymbolic machine learning tools cannot easily deal with general classes of data structures.

Example 2.1 Trivial Graphs: A trivial graph has a single node and no edges. The class $\mathcal{Y}^{\#^{(0,0)}}$ is the class of trivial data structures. Data of this kind will be referred to as *trivially structured* in this paper. Models and techniques for dealing with such data are largely predominant in the machine learning literature. For instance, feedforward neural networks, decision trees, and Bayesian classifiers assume that each instance is described by a simple tuple of predictive attributes, sometimes encoded by fixed size arrays of real numbers. Models of this kind will be referred to as *unstructured* since no relation is defined among the variables that characterize the instances in the learning domain.

Example 2.2 Strings and Sequences: Let us consider the class of linear chains $\#^{(1,1)}$ and let $\mathcal{Y}$ be a finite alphabet. Topological order in linear chains is a total order and can be associated with serial order in strings. Hence, the class $\mathcal{Y}^{\#^{(1,1)}}$ corresponds to the set of all strings of finite length over the alphabet $\mathcal{Y}$, i.e., $\mathcal{Y}^{\#^{(1,1)}} = \mathcal{Y}^*$ is the free monoid over $\mathcal{Y}$ (also known as the Kleene closure of $\mathcal{Y}$).

The topological order in linear chains can also be associated with temporal order in a discrete-time process, i.e., there exists a bijection between nodes in linear chains and natural numbers. In other words, a node t in a linear chain can be unambiguously associated with a discrete-time index t. In this way, if $\mathcal{Y}$ is a continuous space (e.g., $\mathcal{Y} = \mathbb{R}^n$) the class $\mathcal{Y}^{\#^{(1,1)}}$ describes the set of all finite sequences of continuous vectors.

Some early attempts have been reported for dealing with serially ordered data using adaptive models for trivially structured data. A well-known example is NetTalk, a feedforward neural network trained to map English text (a sequence of serially ordered characters) into a sequence of phonetic acoustic parameters for speech synthesis [20]. The difficulty of similar approaches is that variable length sequences must be first transformed into fixed width vectors. This is typically achieved by choosing an appropriate moving window of fixed size, like in time-delay neural networks. A significant drawback of this approach is that the depth of temporal dependencies must be fixed in advance, instead of being learned from data. These difficulties were recognized very early in the connectionist community (see, e.g., [21]) and adaptive models for dealing with sequentially ordered data are now well known. The most significant examples are neural architectures such as recurrent neural networks (RNN's) [22] and probabilistic models such as hidden Markov models (HMM's) [23] or input–output HMM's (IOHMM's) [24].

Example 2.3 Binary Trees: Let us consider the class of binary trees $\#^{(1,2)}$. When making predictions on data structures in $\mathcal{Y}^{\#^{(1,2)}}$ context at any given node is split into two separate ordered pieces of information, namely the left and the right context. In principle, unstructured models might be used to adaptively process binary tree data structures. One could in fact rely on the extension of the moving window approach for sequences: each tree is encoded by fixed-size vectors, which are subsequently used as input to an unstructured model for making predictions. This approach is motivated by the fact that unstructured models such as feedforward neural networks only have a fixed number of input units while trees are variable in size. The encoding process must be defined *a priori*. Unlike temporal windows, in which the only degree of freedom involves the window length, encoding binary trees in fixed-size vectors involves arbitrary structural choices, since it requires the selection of a fixed-size subgraph.

B. Generalized Shift Operators

A discrete-time operator applied to a temporal variable Y_t is essentially a symbolic transformation that maps Y_t into an expression involving the variable Y measured at different time steps. In particular, the shift operator[4] q^{-1} applied to Y_t returns the variable Y at time $t-1$: $q^{-1}Y_t = Y_{t-1}$. In our graphical framework, a finite-length temporal sequence corresponds to a linear chain. The supersource (the head of the chain) corresponds to the last time step in the sequence, and thus time indexes decrease following the direction of the arrows. Hence, q^{-1} can be thought of as an operator that gives access to the (unique) child of a given node, when a sequence is graphically represented as a linear chain. Shift operators can be composed so that $q^{-t} = q^{-1}q^{1-t}$, where $1 = q^0$ is the *neutral* operator defined as $q^0 Y_t = Y_t$.

When general DOAG's are considered, an ordered set of generalized shift operators can be defined and associated

[4]Other discrete-time operators have been introduced in the temporal domain. For example, the gamma operator [25] is defined by $\gamma = (q - 1 + \mu/)\mu, \mu$ being a constant between zero and one. A good review of discrete-time operators can be found in [26].

with the ordered set of children. For $k = 1, \cdots, o$, we denote by q_k^{-1} the shift operator associated with the kth child of a given node. For the class $\#^{(1,1)}$ (i.e., the class of linear chains) the subscript will be omitted. In some special classes of graphs, we sometimes use more descriptive subscript notations. For example, when considering the class of binary trees $\#^{(1,2)}, q_L^{-1}$ and q_R^{-1} will be used to denote the operators associated with the left child and the right child, respectively.

When considering a labeled graph $\boldsymbol{Y}$ with local universe domain $\mathcal{Y}$, the expression $q_k^{-1}\boldsymbol{Y}_v$ denotes the label found in the kth child of node v. For example, in Fig. 3, denoting by s the supersource (the node labeled by "$\phi(\cdot,\cdot,\cdot)$"), $q_1^{-1}\boldsymbol{Y}_s$ is the label in the first child of s (i.e., "α") and $q_2^{-1}\boldsymbol{Y}_s$ is the label in the second child of s (i.e., "$\psi(\cdot)$"). Note that $q_k^{-1}\boldsymbol{Y}_v$ is a nonempty label only if $k \leq |\mathrm{ch}[v]|$. Shift operators can be composed and expressions involving composition of shift operators specify directed paths in the DOAG. Note that the composition of shift operators is not commutative. In the example of Fig. 3, the composition $q_2^{-1}q_3^{-1}\boldsymbol{Y}_s$ yields the label "$\phi(\cdot,\cdot)$." However, the composition $q_3^{-1}q_2^{-1}\boldsymbol{Y}_s$ would yield an empty label since the node labeled by "$\psi(\cdot)$" has only two children. When considering the class $\#^{(1,m)}$ of $m$ary trees, the *inverse* shift operators q_k can also be defined as $q_k q_k^{-1} Y_v = Y_v, k = 1, \cdots, m$.

C. Probabilistic Graphical Models

In this paper, we shall describe a general graphical formalism for data structure processing. Since the formalism is largely inspired by causal semantics commonly attached to probabilistic graphical models (also known as belief networks or causal networks), we briefly review these models in this section.

Belief or conditional independence networks became popular in artificial intelligence as a tool for reasoning in probabilistic expert systems [27]. More in general, belief networks are effectively used in statistics for representing and manipulating complex probability distributions [28]. As a matter of fact, many learning systems, such as Boltzmann machines [29], multilayered perceptrons [30], (input–output) hidden Markov models [23], [24], (just to mention some of them) can be easily regarded as particular graphical models.

A belief network is an annotated graph in which nodes represent random variables in the universe of discourse, and *missing* edges encode a set of conditional independence statements among these variables. Given a particular state of knowledge, the semantics of belief networks determine whether collecting evidence about a set of variables modifies our knowledge about some other set of variables. Specifically, let $\boldsymbol{U}$ denote the universe of discourse and let $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ be disjoint subsets of $\boldsymbol{U}$. The variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ are said to be conditionally independent given $\boldsymbol{Z}$, denoted $\boldsymbol{X} \perp \boldsymbol{Y}|\boldsymbol{Z}$, if $P(\boldsymbol{X}|\boldsymbol{Z},\boldsymbol{Y}) = P(\boldsymbol{X}|\boldsymbol{Z})$ whenever $P(\boldsymbol{Y},\boldsymbol{Z}) > 0$. A conditional independence model is a collection of triplets $(\boldsymbol{X},\boldsymbol{Y},\boldsymbol{Z})$ such that $\boldsymbol{X} \perp \boldsymbol{Y}|\boldsymbol{Z}$ holds true. A graph with nodes associated with variables in $\boldsymbol{U}$ is an independency map for the independence model if the nodes associated with $\boldsymbol{X}$ and $\boldsymbol{Y}$ are graphically separated by the subset of nodes $\boldsymbol{Z}$ for each triplet such that $\boldsymbol{X} \perp \boldsymbol{Y}|\boldsymbol{Z}$. Graphical separation criteria for verifying conditional independence can be defined for undirected graphs (also known as Markov networks), directed acyclic graphs (DAG's) (also known as Bayesian networks) and chain graphs [31]. These criteria are referred to as u-separation, d-separation, and c-separation, respectively.

Belief networks, however, are not limited to qualitatively encoding conditional independencies, but they also quantitatively specify the parameters of the probability distribution over the universe of discourse. In particular, in the case of Bayesian networks (BN's), it can be shown that the table $P(\boldsymbol{A})$ on the universe of discourse $\boldsymbol{A}$ can be factored as

$$P(\boldsymbol{A}) = \prod_i P(A_i|\mathrm{Pa}_i) \tag{1}$$

where Pa_i denotes the parents of A_i. Hence, BN's are specified by a DAG with local density models $P(A_i|\mathrm{Pa}_i, \boldsymbol{\theta}_i)$ attached to each node, where $\boldsymbol{\theta}_i$ is a set of parameters for the local density. In the case of categorical variables, the simplest choice is the unrestricted multinomial model for the local densities, i.e., $P(A_i|\mathrm{Pa}_i, \boldsymbol{\theta}_i)$ is a conditional probability table (CPT) with entries $\theta_{ijk} = P(A_i = a_i^k|\mathrm{Pa}_i = \mathrm{pa}_i^j)$. In the case of numerical variables, a common choice for the local densities is based on the Gaussian model. Most of the theoretical results in graphical models (in particular, results concerning learning) hold in the more general case of local densities belonging to the exponential family [32].

Directed belief networks have an immediate interesting interpretation in terms of probabilistic causal relationships. The set of parents Pa_i of a variable A_i is the subset of the universe of discourse which has a *direct* causal impact on A_i. This notion of causality, however, need not to be restricted to probabilistic relationships, and directed graphical models can be extended to include deterministic causal relationships. Following [32], we shall consider *deterministic* nodes as computing a deterministic function of the state of their parents. Formally, $a_i = f(\mathrm{pa}_i)$ iff $P(A_i = a_i|\mathrm{Pa}_i = \mathrm{pa}_i) = \delta(a_i, f(\mathrm{pa}_i))$ where δ denotes the Dirac function $\delta(x,y) = 1$ if $x = y$ and $\delta(x,y) = 0$ if $x \neq y$. Artificial neural networks are one of the best known examples of graphical models involving deterministic nodes. Throughout this paper we shall tacitly assume that neurons in connectionist networks are modeled as deterministic nodes in a graphical model.

III. General Aspects of Structural Processing

Generally speaking, the problem of learning with data structures consists of making predictions based on the knowledge of a labeled graph. The framework assumed in this paper is essentially probabilistic. In our setting, the unsupervised learning problem is formulated as the estimation of the density $P(\boldsymbol{Y})$ over $\mathcal{Y}^{\#}$. The supervised learning problem is formulated as the estimation of the *conditional* distribution $P(\boldsymbol{Y}|\boldsymbol{U})$ of an *output* DOAG $\boldsymbol{Y}$ given an *input* DOAG $\boldsymbol{U}$, where the local universe domains $\mathcal{U}$ and $\mathcal{Y}$ are generally distinct.

We shall introduce two distinct but related classes of models for supervised learning. As detailed below, both classes rely on a hidden state-space representation, where states are random variables. Relationships among input, output, and state variables may be deterministic or probabilistic. The former kind of computation is typical of artificial neural networks, where neural activities are deterministic functions of random variables and the response of the output neurons is interpreted as a "position" parameter for the conditional density of the output given the input (i.e., the conditional expectation $E[\boldsymbol{Y}|\boldsymbol{U}]$). The latter kind of computation is typical of Bayesian networks, where nodes are labeled by random variables and $P(\boldsymbol{Y}|\boldsymbol{U})$ is directly obtained by running a probabilistic inference algorithm after having entered evidence into the input nodes. The distinction between these two classes of models is mostly a matter of interpretation. In particular, neural networks could also be interpreted as fully probabilistic directed graphical models, in which the logistic function of weighted sums is read as the conditional density of units, given their parents. The problem with this interpretation is that neural networks are densely connected graphs and, therefore, exact inference algorithms would quickly become intractable even for moderately large networks.

A. Structural Transductions

In order to describe some general properties of models for supervised learning on data structures, it is useful to abandon temporarily the probabilistic setting and to assume that data were generated according to a deterministic transduction on structured spaces.

Generally speaking, a deterministic transduction is a binary relation defined on $\mathcal{U}^{\#} \times \mathcal{Y}^{\#}$, where $\mathcal{U}^{\#}$ and $\mathcal{Y}^{\#}$ are two structured spaces. However, only those relations which are functions from $\mathcal{U}^{\#}$ to $\mathcal{Y}^{\#}$ will be considered in this paper. Learning general functions from $\mathcal{U}^{\#}$ to $\mathcal{Y}^{\#}$ is a challenging open research problem. In this section we further restrict our learning domain and characterize a subclass of transductions for which it is reasonably easy to build adaptive models. In particular, a general function from $\mathcal{U}^{\#}$ to $\mathcal{Y}^{\#}$ can modify the skeleton of the structure being processed, i.e., it may be $\mathrm{skel}(\boldsymbol{U}) \neq \mathrm{skel}(\tau(\boldsymbol{U}))$. In the case of sequences, allowing a transduction to modify the skeleton essentially means to allow the input and output sequences to have different lengths. Sequential transductions are called *synchronous* if, for each time step, an input label is consumed and an output label is emitted (i.e., inputs and outputs share the same time scale). The concept of synchronism can be generalized to transductions on structures, by asking that the input and the output structures share the same skeleton. Specifically, a transduction $\tau(\cdot)$ is *IO-isomorph* if

$$\mathrm{skel}(\tau(\boldsymbol{U})) = \mathrm{skel}(\boldsymbol{U}) \quad \forall \boldsymbol{U} \in \mathcal{U}^{\#}.$$

In this paper we restrict our attention to IO-isomorph transductions.

An IO-isomorph transduction $\tau(\cdot)$ is *algebraic*, or *unstructured*, if $\forall \boldsymbol{U} \in \mathcal{U}^{\#}$ and $\forall v \in \mathrm{vert}(\boldsymbol{U}), \tau(\boldsymbol{U})_v$ only depends on $\boldsymbol{U}_v$. Clearly, the problem of learning algebraic transductions can be reduced to a conventional learning problem in which $\mathcal{U}$ and $\mathcal{Y}$ are the input and the output instance spaces. By contrast, predictions in structural transductions which are not algebraic, depend on *contextual* information possibly stored throughout the whole input data structure. For example, transductions operated by temporal dynamical systems (such as recurrent neural networks) are not algebraic.

An IO-isomorph structural transduction $\tau(\cdot)$ is *causal* if $\forall v \in \mathrm{vert}\ (\boldsymbol{U})\tau(\boldsymbol{U})_v$ only depends on the subgraph of $\boldsymbol{U}$ induced by $\{v\} \cup \mathrm{de}[v]$. Causal transductions which are not algebraic need some kind of *memory* to store information about the input labels found in the descendants of a given node v. For example, let us consider the class of linear chains. In this case, the memory is a device that stores information about past events so that the output $\boldsymbol{Y}_t$ at time t does not only depend on the input $\boldsymbol{U}_t$ at time t, but also on the past inputs $\boldsymbol{U}_{t-1}, \boldsymbol{U}_{t-2} \cdots$. Sequential transductions of this kind are realized by dynamical systems and the memory in these systems is normally associated with the concept of *internal state*. Causality in dynamical systems is a necessary and sufficient condition for the existence of an internal state. These issues can be generalized and extended to data organized in the form of DOAG's.

To give an example of the above definitions, consider again the logical terms domain and suppose that the output label space is a binary alphabet indicating the class (positive or negative) of terms. As shown in Fig. 5, the transduction τ is IO-isomorph since input and output graphs share the same skeleton. The label $\boldsymbol{Y}_v$ at any generic node v can be thought of as the class of the subterm rooted at node v (i.e., the class of the subgraph induced by $v \cup \mathrm{de}[v]$). The label at the supersource is clearly the class of the whole term. Looking at the top of Fig. 5 we can notice that in this case $\tau$is not algebraic. In fact, for example, the output label for the subterm $\phi(\cdot,\cdot)$ depends also on the context associated with the labels found in other nodes (if τ were algebraic, then the outputs at the supersource and at the second child of the supersource would have been equal). Moreover, looking at the bottom of Fig. 5 we can see that τ is not causal. In fact, the class given to the subterm $\phi(\alpha, \beta)$ (the second child of the supersource) depends on a contextual information found in the supersource. If τ were causal, the class given to the term would have only depended on the bottom context associated with the arguments of ϕ.

We say that an IO-isomorph transduction $\tau(\cdot)$ from $\mathcal{U}^{\#^{(i,o)}}$ to $\mathcal{Y}^{\#^{(i,o)}}$ admits a *recursive state representation* if there exists a structure space $\mathcal{X}^{\#^{(i,o)}}$ such that for each $\boldsymbol{U}, \boldsymbol{Y} = \tau(\boldsymbol{U})$ there exist $\boldsymbol{X} \in \mathcal{X}^{\#^{(i,o)}}$ with $\mathrm{skel}(\boldsymbol{X}) = \mathrm{skel}(\boldsymbol{U}) = \mathrm{skel}(\boldsymbol{Y})$ and two functions

$$f: \mathcal{X}^K \times \mathcal{U} \times V(\#^{(i,o)}) \to \mathcal{X}$$
$$g: \mathcal{X} \times \mathcal{U} \times V(\#^{(i,o)}) \to \mathcal{Y}$$

such that for each $v \in \mathrm{skel}(U)$

$$\boldsymbol{X}_v = f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v, v) \tag{2}$$
$$\boldsymbol{Y}_v = g(\boldsymbol{X}_v, \boldsymbol{U}_v, v) \tag{3}$$

where $\boldsymbol{X}_{\mathrm{ch}[v]}$ is a fixed size array of labels (sets of variables) attached to the children of v in the internal state DOAG. In the above definition, f is referred to as the *state transition function* and g is referred to as the *output function*. It can be observed that in the special case of sequences, each node v corresponds to a discrete-time point and $\mathrm{ch}[v]$ contains a single node that corresponds to the previous time point. Hence, in the case of sequences, the above representation corresponds to the usual state-space representation of dynamical systems, as found for example in control systems. Intuitively, data generated by transductions that admit a recursive representation are explained by a *hidden* structure whose skeleton matches the input and the output skeletons. This representation will be used in the following for characterizing models (such as recursive neural networks) with deterministic relations between input, state, and output variables. In that context, it will be assumed that these functions depend on trainable parameters (such as connection weights).

States in (2) are updated following a recursive message passing scheme in which each state label $\boldsymbol{X}_v$ is updated after the state labels corresponding to the children of v. On a serial computer this can be achieved by traversing the graph skel$(\boldsymbol{U})$ according to the order defined by any reversed topological sort[5] of the nodes in skel$(\boldsymbol{U})$. On a parallel computer some states can be simultaneously updated, propagating from the frontier (the set of nodes v such that $\mathrm{ch}[v] = \emptyset$) to the supersource. Note that $\boldsymbol{X}_{\mathrm{ch}[v]}$ is a fixed size array of o elements, where o is the maximum outdegree of the input DOAG. If vertex v has $k < o$ children, then in order to apply (2) it is also necessary to specify the states associated with the missing children (i.e., elements of $\boldsymbol{X}_{\mathrm{ch}[v]}$ associated with the indexes ranging from $k+1$ to o). In particular, if v is a leaf then $\mathrm{ch}[v] = \emptyset$, i.e., all the children of v are missing. States associated with missing children are accessed at basis of induction that terminates a recursive traversal of the DOAG. The concept of basis of recursion has a direct correspondence to the concept of *initial state* X^0 in classic dynamical systems (such as recurrent neural networks) that deal with sequentially ordered data. Initial states in these models may be assumed to be fixed, or may be learned from data [33]. In the case of general DOAG's, it is necessary to specify a set of state variables $\boldsymbol{X}^F$ associated with the basis of recursion. This set will be referred to as the *frontier label*. Frontier labels are used in (2) in correspondence of missing children in the data structure being processed. In other words, the array $\boldsymbol{X}_{\mathrm{ch}[v]}$ is filled in with the frontier state label $\boldsymbol{X}^F$ whenever one of the children of v is missing.[6]

Equations (2) and (3) can be also interpreted in terms of causal dependencies among input, state, and output variables. Causal dependency in deterministic models is related to conditional independency in probabilistic belief network. The main difference is that in (2) and (3) variables are a *functions* of their parents and, therefore are conditionally independent of the *rest* (the remaining variables), given their parents. In belief networks (which is a more general case, since functions may be thought of as degenerate conditional densities in which all the probability mass is concentrated on a single value), a variable is conditionally independent of the rest given its Markov blanket, namely the set of nodes formed by joining children, parents, and parents of the children [27]. Note that noncausal transductions cannot admit a recursive state representation and, therefore, the causality property of the global map $\tau(\cdot)$ can be explained in terms of local causal dependencies in the recursive representation.

The state transition function f and the output function g in (2) and (3) are dependent on v. A causal IO-isomorph transduction $\tau(\cdot)$ is said to be *stationary* if these functions are independent on node v. Stationarity defined in this way generalizes the concept of time-invariance which applies to dynamical systems operating on the class of linear chains.

In some cases, structural processors are used to produce outputs which are not themselves structured. For example, in a problem of data structure classification, only a single categorical variable is commonly associated with the whole input structure. Since this is a remarkable type of prediction, we give a specialized definition for transductions whose output space is not structured. An *supersource transduction* $\tau(\cdot)$ is a function from $\mathcal{U}^{\#}$ to $\mathcal{Y}$ defined through the following recursive representation:

$$\boldsymbol{X}_v = f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v, v) \tag{4}$$
$$\boldsymbol{Y} = g(\boldsymbol{X}_s) \tag{5}$$

where s denotes the supersource of the input graph. Supersource transductions map an input structure $\boldsymbol{U}$ into an output structure $\boldsymbol{Y}$ whose skeleton is always made of a single node (i.e., a trivial graph). Alternatively, supersource transductions can be thought of as IO-isomorph transductions in which all the output labels are empty sets of variables, except for the label attached to the supersource of $\boldsymbol{U}$.

It is worth mentioning that stationary transductions that admit a recursive representation cannot compute any function. For example, consider the two DOAG's shown in Fig. 6. Any given supersource stationary transduction described by (4) will necessarily map these two graphs into the same output, regardless of the form of functions f and g. In fact, it can be seen that the state variables on the leaves must be equal (i.e., $\boldsymbol{X}_4 = \boldsymbol{X}_8 = \boldsymbol{X}_9$) since $\boldsymbol{U}_4 = \boldsymbol{U}_8 = \boldsymbol{U}_9 = d$. Therefore, when propagating the state from the frontier, it can be easily seen that the state variables at the supersource are equal and so must be the predicted output.

Example 3.1 Tree Automata: Perhaps, the best known stationary transductions that operate on structures more complex that linear chains are those realized by tree automata [34], that we briefly recall here. For the sake of simplicity we limit our discussion to binary tree automata. Let us first introduce accepting automata. A *frontier to root* automaton (FRA) is a five-tuple $\{\overline{x}, \mathcal{X}, \mathcal{U}, \delta, A\}$ in which the transition function δ maps a triple in $\mathcal{X}^2 \times \mathcal{U}$

[5]A topological sort of a DAG D is any linear order $\prec$ on the vertices of D such that $v \prec w$ whenever D has an edge (v, w).

[6]A more general setting might be conceived, in which different frontier states are associated with different children. For example, in the case of binary trees, the left and the right frontier states might take on different values $\boldsymbol{X}_L^F$ and $\boldsymbol{X}_R^F$.

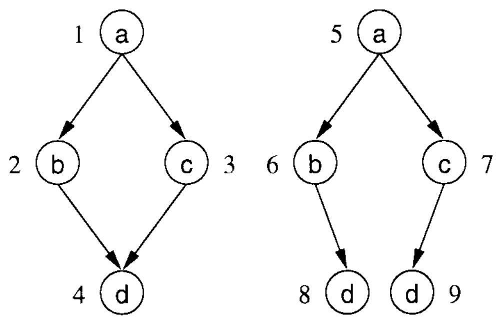

Fig. 6. Two different DOAG's necessarily mapped into the same output by *any* causal stationary supersource transduction.

into a "next state" $X_v = \delta(q_L^{-1}X_v, q_R^{-1}X_v, U_v) \in \mathcal{X}$, being U_v a symbol in $\mathcal{U}$, and q_L^{-1}, q_R^{-1} the shift operators pointing to the left and right child, respectively. $\overline{x}$ is the *frontier* state. The computation of the automaton proceeds from the external nodes toward the root r. A binary tree is accepted iff $X_r \in A$, the set of accepting states. Accepting tree automata fit the state-space representation (4) and thus realize supersource transductions. Translating tree machines can easily be defined by introducing an output alphabet $\mathcal{Y}$ and replacing A with an output function g that maps a pair in $\mathcal{X} \times \mathcal{U}$ into $\mathcal{Y}$. Translating tree machines defined in this way clearly realize causal IO-isomorph transductions from $\mathcal{U}^{\#^{(1,2)}}$ to $\mathcal{Y}^{\#^{(1,2)}}$. In fact, the pair of functions (δ, g) specifies a recursive representation for such transductions.

B. Probabilistic Transductions

A probabilistic transduction is a joint density $P(\boldsymbol{U}, \boldsymbol{Y})$ defined over $\mathcal{U}^{\#} \times \mathcal{Y}^{\#}$. All the concepts we have defined for qualifying deterministic transductions can be easily generalized to probabilistic transductions by constraining $P(\boldsymbol{U}, \boldsymbol{Y})$. We shall say that a probabilistic transduction has a generic property F (such as IO-isomorphism, causality, etc.) if $P(\boldsymbol{U}, \boldsymbol{Y}) = 0$ whenever there exists a deterministic transduction $\tau(\cdot)$ such that $\boldsymbol{Y} = \tau(\boldsymbol{U})$ and $\tau(\cdot)$ does not have the property F. For example, a probabilistic transduction is IO-isomorphic if $P(\boldsymbol{U}, \boldsymbol{Y}) = 0$ whenever $\mathrm{skel}(\boldsymbol{U}) \neq \mathrm{skel}(\boldsymbol{Y})$. Similarly, we say that an IO-isomorph probabilistic transduction on $\mathcal{U}^{\#^{(i,o)}} \times \mathcal{Y}^{\#^{(i,o)}}$ admits a *recursive state representation* if there exists a structure space $\mathcal{X}^{\#^{(i,o)}}$ such that for each $\boldsymbol{U}, \boldsymbol{Y}$ there exists $\boldsymbol{X} \in \mathcal{X}^{\#^{(i,o)}}$ with $\mathrm{skel}(\boldsymbol{X}) = \mathrm{skel}(\boldsymbol{U}) = \mathrm{skel}(\boldsymbol{Y})$ such that

$$P(\boldsymbol{U}, \boldsymbol{Y}) = P(\boldsymbol{U}) \prod_{v \in \mathrm{vert}(\boldsymbol{U})} P(\boldsymbol{Y}_v | \boldsymbol{X}_v, \boldsymbol{U}_v) P(\boldsymbol{X}_v | \boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v) \tag{6}$$

where $P(\boldsymbol{X}_v | \boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v)$ are the *state transition densities* and $P(\boldsymbol{Y}_v | \boldsymbol{X}_v, \boldsymbol{U}_v)$ are the *emission densities*. In the case of stationary transductions, these densities are the same at every node v. As we did for deterministic transductions, the set $\boldsymbol{X}_{\mathrm{ch}[v]}$ is assumed to be filled with frontier states whenever v has missing children. A probabilistic transduction is said to be stationary if state transition and emission densities are independent of the vertex v. Finally, supersource probabilistic transductions can be defined by assuming that $\boldsymbol{Y}$ is a trivial graph and that

$$P(\boldsymbol{U}, \boldsymbol{Y}) = P(\boldsymbol{U}) P(\boldsymbol{Y} | \boldsymbol{X}_s) \prod_{v \in \mathrm{vert}(\boldsymbol{U})} P(\boldsymbol{X}_v | \boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v) \tag{7}$$

where s is the supersource of $\boldsymbol{U}$.

As shown in the following section, (7) and (6) are identical to the factorization of $P(\boldsymbol{U}, \boldsymbol{Y})$ in Bayesian networks whose universe of discourse contains input, state, and output variables.

C. Graphical Models for Structural Processors

We now give a graphical notation for the recursive representation of causal transductions. The proposed formalism describes both probabilistic and deterministic transductions.

Let $\tau(\cdot)$ be an IO-isomorph causal stationary transduction from $\mathcal{U}^{\#}$ to $\mathcal{Y}^{\#}$. The canonical *recursive network* of $\tau(\cdot)$ is a directed (possibly cyclic) labeled graph $N(\tau, \#)$ defined as follows.

- $N(\tau, \#)$ has $|\mathcal{U}| + |\mathcal{X}| + |\mathcal{Y}|$ nodes and each node is labeled by one distinct variable chosen from the sets of input, state, and output variables.
- Edges in $N(\tau, \#)$ describe causal dependencies among the input, state, and output variables in the recursive representation of $\tau(\cdot)$. Edges are labeled by shift operators. Let A, B be generic variables. An edge $(A, B, 1)$ labeled by the neutral shift operator indicates that B_v is causally dependent on A_v. Similarly, an edge (A, B, q_k^{-1}) indicates that B_v is causally dependent on $q_k^{-1}A_v$. Edges incident to output nodes must always be labeled by the neutral shift operator 1.
- Edges incident to either output or state nodes cannot leave from output nodes. Node labeled by input variables must have zero indegree. Cycles in $N(\tau, \#)$ such that all edges are labeled by the neutral shift operator are not allowed.

Noncanonical recursive networks can be defined by labeling the edges of $N(\tau, \#)$ with a composition of shift operators $q_{k_1}^{-1} q_{k_2}^{-1}, \cdots, q_{k_n}^{-1}$ that will be abbreviated by $L(\boldsymbol{q}^{-1})$.

The whole set of variables involved during the processing of a given data structure $\boldsymbol{U}$ can be graphically represented using the recursive network associated with the transduction. Essentially, the recursive network is used as a template which is unfolded (i.e., expanded) according to the skeleton of the input data structure (that, by IO-isomorphism assumption, matches the skeleton of the output structure). The resulting labeled graph (function of both the recursive network $N(\tau, \#)$ and the input–output data structures $\boldsymbol{U} \in \mathcal{U}^{\#}, \boldsymbol{Y} \in \mathcal{Y}^{\#}$) is called *encoding network* of the transduction, denoted $\mathcal{E}(N, \boldsymbol{U}, \boldsymbol{Y})$. Nodes and edges of $\mathcal{E}(N, \boldsymbol{U}, \boldsymbol{Y})$ are constructed as follows.

- The vertex set of $\mathcal{E}$ is the Cartesian product of the vertex sets of N and $\mathrm{skel}(\boldsymbol{U})$. For each $v \in \mathrm{vert}(\mathrm{skel}(\boldsymbol{U})), U_{i,v}$ denotes the ith input variable at node $v, X_{i,v}$ denotes the

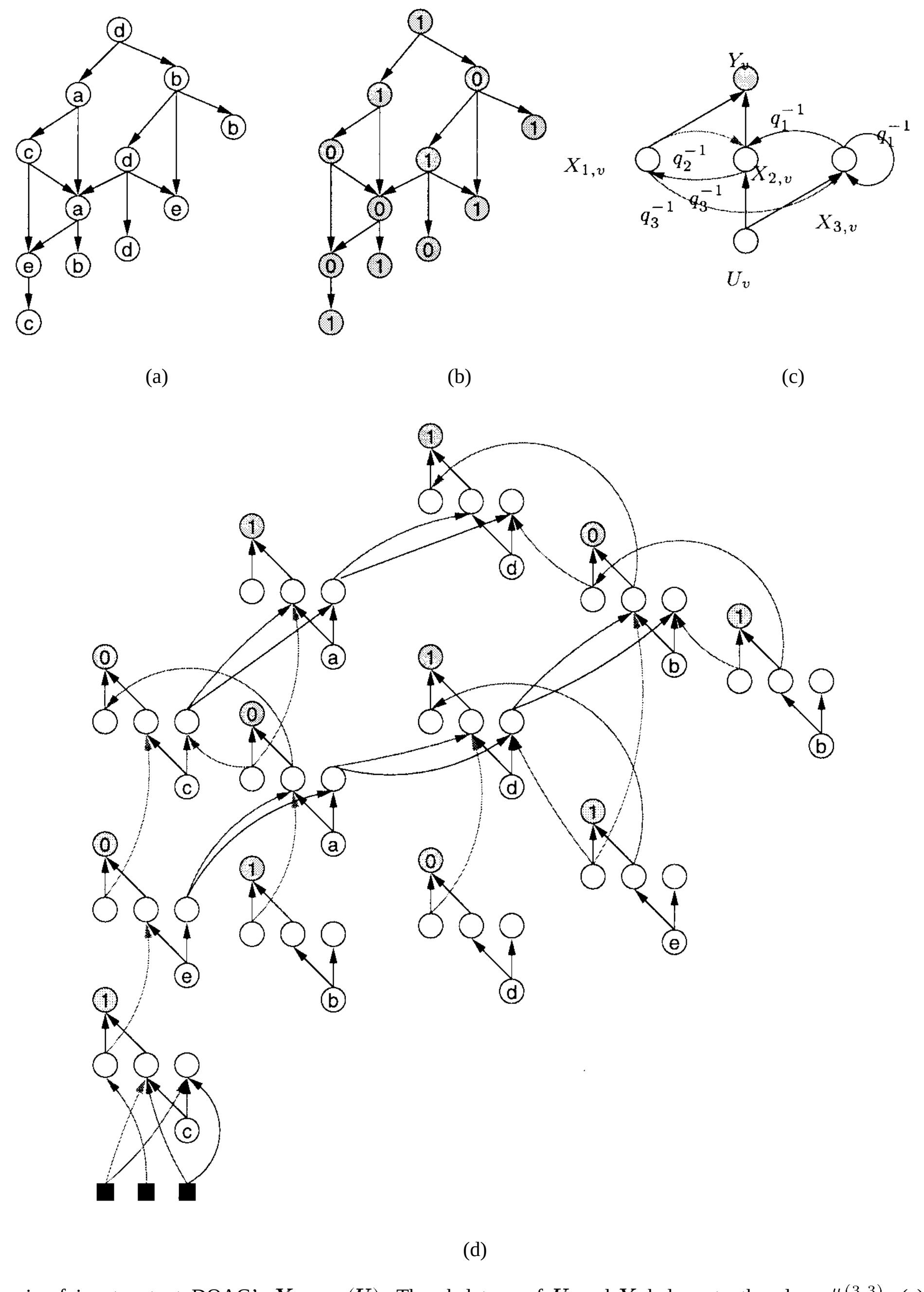

Fig. 7. (a) and (b) A pair of input–output DOAG's $\boldsymbol{Y} = \tau(\boldsymbol{U})$. The skeletons of $\boldsymbol{U}$ and $\boldsymbol{Y}$ belong to the class $\#^{(3,3)}$. (c) Recursive network for the IO-isomorph transduction $\tau(\cdot)$. In this example, the state space has three components X_1, X_2, X_3. (d) Encoding network for the given input–output DOAG's and the recursive network of $\tau(\cdot)$. For the sake of simplicity, edges leaving the frontier state variables are only drawn when incident on the nodes located on the bottom left portion of the encoding network.

ith state variable at node v, and $Y_{i,v}$ denotes the ith output variable at node v.

- The edge set of $\mathcal{E}$ is obtained as follows. Let $A_{i,v}B_{j,w}$ denote two vertices of $\mathcal{E}$. The directed edge $(A_{i,v}, B_{j,w})$ is present in $\mathcal{E}$ if and only if the edge (A_i, B_j) is present in N and is labeled by an expression $L(\boldsymbol{q}^{-1})$ such that $L(\boldsymbol{q}^{-1})\boldsymbol{U}_w = \boldsymbol{U}_v$.

Algorithm 1 builds the skeleton of the encoding network associated with the recursive network N and the skeleton $S = \text{skel}(\boldsymbol{U}) = \text{skel}(\boldsymbol{Y})$ of the input–output DOAG's.

```
Algorithm 1 BUILD-ENCODING-NETWORK($N, S$)
vert($\mathcal{E}$) $\leftarrow \emptyset$
edg($\mathcal{E}$) $\leftarrow \emptyset$
**foreach** $v \in$ vert($S$) **do**
    **foreach** $A_i \in$ vert($N$) **do**
        vert($\mathcal{E}$) = vert($\mathcal{E}$) $\cup$ $\{A_{i,v}\}$
**foreach** $(A_i, A_j, L(\boldsymbol{q}^{-1})) \in$ edg($N$) **do**
    **foreach** $v \in$ vert($S$) **do**
        **if** $L(\boldsymbol{q}^{-1})v \neq nil$ **then**
            edg($\mathcal{E}$) = edg($\mathcal{E}$) $\cup$ $\{(L(\boldsymbol{q}^{-1})A_{i,v}, A_{j,v})\}$
        **else**
            edg($\mathcal{E}$) = edg($\mathcal{E}$) $\cup$ $\{(\boldsymbol{X}_i^F, A_{j,v})\}$
**return** $\mathcal{E}$
```

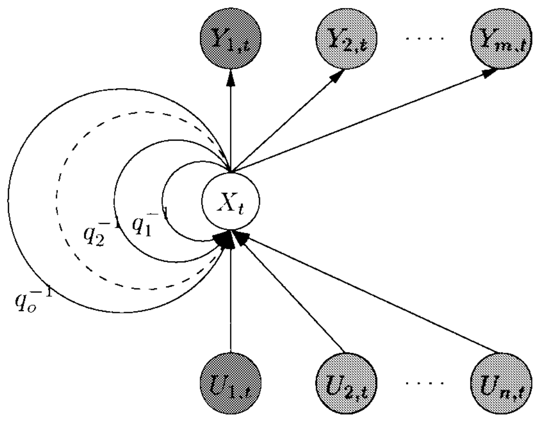

Fig. 8. Recursive network of an HRM.

An example of encoding network construction is shown in Fig. 7. Note that arrows in the encoding networks are reversed with respect to the direction of arrows in the input data structure. This is because computation in recursive causal transductions proceeds from the frontier to the supersource. Shift operators can be applied also to nodes of the encoding network. In particular, $q_k^{-1}\boldsymbol{X}_v$ returns the state label attached to the kth child of v in the skeleton of $\boldsymbol{U}$ (which is the kth parent of $\boldsymbol{X}_v$ in the encoding network).

It is straightforward to recognize that in the case of probabilistic transductions, the encoding network constructed by the above algorithm is a Bayesian network for the density $P(\boldsymbol{U}, \boldsymbol{Y})$ factored according to (6). As detailed in the next section, when the transduction is implemented by means of recursive neural networks, the encoding network is a feedforward neural network.

IV. Models for Adaptive Structure Processing

A. Hidden Recursive Models

Hidden recursive models (HRM's) are a class of probabilistic models for structure processing with hidden discrete states. The recursive network for a general HRM is shown in Fig. 8. Besides input and output nodes, the recursive network contains a single node X_v, corresponding to a hidden discrete state variable[7] at node v. In a model for the class of skeletons $\#^{(i,o)}$, the node X_v has o recursive connections (self loops) labeled by $q_1^{-1}, q_2^{-1}, \cdots, q_o^{-1}$. We denote by $\mathcal{X} = \{x^1, \cdots, x^{r_x}\}$ the hidden state space. The size of the state space is usually a design choice. Alternatively, model selection techniques may be employed for learning r_x from data. In a Moore model, edges are such that input variables $U_{i,v}$ are parents of X_v and X_v is a parent of output variables $Y_{j,v}$. In a Mealy model, additional edges from $U_{i,v}$ to $Y_{j,v}$ are present (see Fig. 8). The encoding network associated with HRM's is a Bayesian network in which the conditional probability tables are shared among the replicas of the basic cell.

Example 4.1 Hidden Markov Models: The hidden Markov models (HMM's) [23] are a well-known device for learning

[7]Extensions to include multiple state variables can be conceived but are not described in this paper.

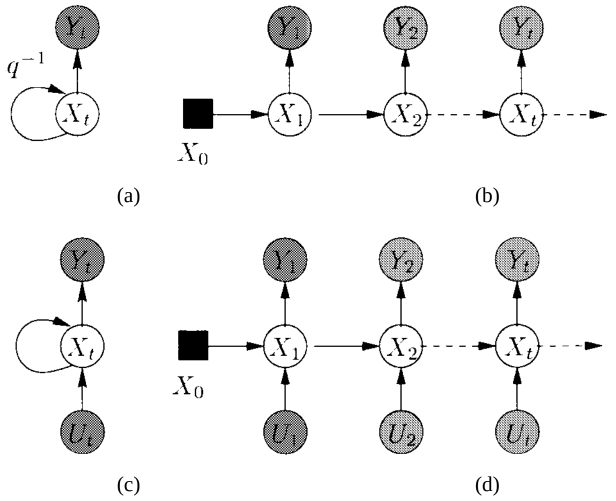

Fig. 9. (a) and (c) Recursive networks for standard HMM's and IOHMM's, respectively. (b) and (d) Encoding networks for standard HMM's and IOHMM's, respectively.

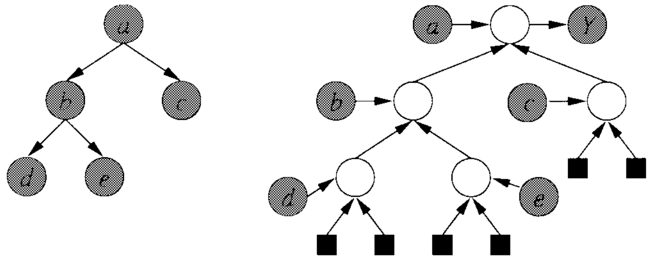

Fig. 10. A binary tree and a corresponding HRM (dark square nodes denote the frontier states).

distributions of temporal sequences (i.e., linear chains, in our framework). A standard HMM is a parametric model of a stochastic process generated by an underlying finite state Markov chain, with an output distribution associated with each state or to each state transition. State variables in HMM's satisfy the Markov conditional independency $X_t \perp X_0, \cdots, X_{t-2} | X_{t-1}$, graphically represented in Fig. 9(b). Fig. 9(a) shows the recursive network for standard HMM's.[8] Input–output HMM's (IOHMM's) [24] are recent extension of HMM's for supervised learning on temporal domains. The main difference is that the recursive network for IOHMM's [shown in Fig. 9(c)] also contains a node labeled by an input variable. HMM's and IOHMM's correspond to the simplest form of HRM's, for dealing with the class of linear chains $\#^{(1,1)}$. Intuitively, IOHMM's and recurrent neural networks are in the same relationship as HRM's and recursive neural networks described in Section IV-B.

If the network is employed for classification of structures then only one output node $\boldsymbol{Y}_s$ is present, where s is the supersource of the input structure $\boldsymbol{U}$. Fig. 11 shows an example of encoding network for an HRM that classifies binary trees.

[8]Often, HMM's are depicted in a different graphical form, known as *state transition diagram*. The state transition diagram is a directed (possibly cyclic) graph, where nodes are labeled by *states* (in contrast to recursive nets, where nodes are labeled by *variables*) and where the absence of an arc from state x^i to state x^j indicates that the probability of making a transition from x^i to x^j is zero.

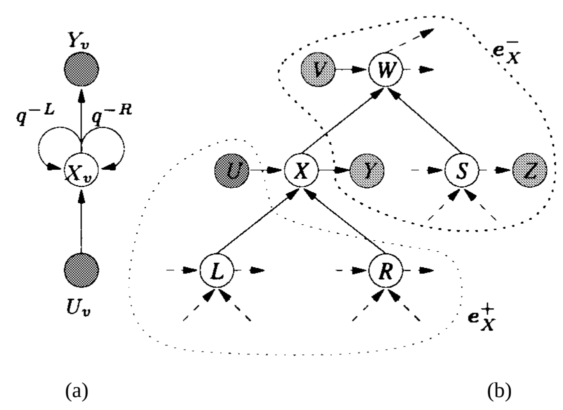

Fig. 11. (a) Recursive diagram for a binary-tree HRM. (b) Slice of the unfolded network; only variables involved in (8) are shown. Shaded variables receive evidence during learning.

An important difference with respect to recursive neural networks is that HRM's can naturally deal with both supervised and unsupervised learning problems. In the case of unsupervised learning, there are no input variables and the model has a generative form in which hidden states causally affect the labels in the observed structure $\boldsymbol{Y}$. Another interesting advantage of HRM's with respect to recursive neural networks is that missing input data can be dealt with by simply entering evidence into the encoding network whenever labels are actually observed.

Evidence is normally entered through the visible nodes of the encoding network. In the case of supervised learning, input and output nodes are visible during training, where output labels play the role of targets. Probabilistic inference assesses the probability of hidden states given the observed data, thus solving a structured credit assignment problem. When making predictions, input nodes only are instantiated and probabilistic inference yields $P(\boldsymbol{Y}|\boldsymbol{U})$. In the case of unsupervised learning, there are no input nodes and evidence is entered into output nodes. Prediction are made by assessing the probability of the evidence $P(\boldsymbol{Y})$.

Frontier variables may be instantiated with a known state, or alternatively the frontier state distributions may be thought of as additional parameters that can be learned from data.

1) Parameters: The parameters in any Bayesian network specify, for the generic node A_v, the conditional probabilities $P(A_v|\mathrm{pa}_v)$. The simplest statistical model for these conditional probabilities is the unrestricted multinomial model. In this case, $P(A_v|\mathrm{pa}_v)$ is a parameter table $\boldsymbol{\theta}_v$ that specifies the probabilities for each state of A_v given each configuration of A_v's parents. In particular, $\theta_{v,i,j}$ is the probability $P(A_v = a_v^i|\mathrm{pa}_v = \mathrm{pa}_v^j)$ where pa_v^j denotes the jth configuration of A_v's parents.

The particular topology we have described above is such that the nodes form a recursive structure which is locally connected according to the same pattern. This feature essentially depends on the fact that the network was obtained by unfolding a basic triplet (U_v, X_v, Y_v) through the skeleton S. A simple consequence is that the conditional distributions $P(X_v|\mathrm{pa}_{X_v})$ and $P(Y_v|\mathrm{pa}_{Y_v})$ all have the same size and similar meanings. Thanks to the stationarity assumption these tables are independent of v and thus we can achieve a significant reduction in terms of model complexity. It can be noticed that even in the simple case of HMM's and IOHMM's, stationarity is a very common assumption.

2) Inference: The theory for inference and learning in HRM's is relatively simple once one recognizes that HRM's are just a special case of Bayesian networks in which the topology is known.

Research concerning inference in Bayesian networks dates back to the 1980's, when the principal concern was the construction of probabilistic expert systems, and is now relatively mature, making available general and well-understood algorithms [35]. Nevertheless, if computing resources are a concern, straightforwardly calling these algorithms as subroutines may not be appropriate in the case of HRM's.

A first basic distinction is between inference algorithms for singly connected DAG's (or polytrees) and general (multiply connected) DAG's. In the former case, inference can be performed directly on the Bayesian network using a local message passing algorithm often referred to as π-λ propagation [27]. In the latter case, the DAG must be first "compiled" into a new structure, called junction tree, whose nodes contain clusters of variables; then inference relies on a local message propagation algorithm between the nodes of the junction trees [35].

Now, although the description on the model is fixed, the topology of the encoding network (the Bayesian network on which inference must be performed) changes with each training example. Hence a different junction tree needs to be constructed for each training example. Maintaining a junction tree for each example may be costly in terms of memory. Recompiling a junction tree each time a new example is presented to the network may be computationally costly since, as discussed below, training algorithms in the presence of hidden variables are iterative and several presentations of the same example are needed before convergence. An interesting solution consists of merging the training examples into an optimally compressed supergraph (as done in [36]), so that only one junction tree has to be built for the entire training set. Alternatively, *ad hoc* propagation algorithms for the class of DOAG's being considered may be derived. An example for the case of binary trees is presented in Section IV-A4. Another problem is that standard inference algorithms are intractable for densely connected networks. Hence, one must resort to approximate methods when dealing with complex classes of DOAG's. A recent interesting solution for approximate inference relies on mean field theory from statistical physics and can be shown to be more and more accurate as the density of connections increases [37].

3) Learning: Since the topology of the network is deterministically known, the learning problem is simply reduced to estimating the parameters of the model given a dataset. The problem is complicated by the presence of hidden variables and exact full Bayesian methods are intractable.

A common approach for learning in the presence of missing data is to estimate the parameters according to the maximum *a posteriori* (MAP) principle. In the MAP framework, parameters are supposed to obey a probability distribution that tends to a delta function centered at a value $\boldsymbol{\theta}^*$ that maximizes $P(\boldsymbol{\theta}|\mathcal{D})$. Thus, instead of learning about $P(\boldsymbol{\theta}|\mathcal{D})$, we only learn about a single value $\boldsymbol{\theta}^*$ and we pretend that $P(\boldsymbol{\theta}|\mathcal{D})$ is negligible for $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$. Now $P(\boldsymbol{\theta}|\mathcal{D}) \propto P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})$, where $P(\boldsymbol{\theta})$ is a *prior* about the parameters. As for large datasets the effect of the prior becomes negligible, $\boldsymbol{\theta}^*$ can be approximated by the *maximum likelihood* (ML) value, i.e., $\boldsymbol{\theta}^* \approx \arg\max_{\boldsymbol{\theta}} = P(\mathcal{D}|\boldsymbol{\theta})$. In this way, learning is cast into an optimization procedure applied to the likelihood function $P(\mathcal{D}|\boldsymbol{\theta})$. Common approaches for solving this optimization problem are gradient ascent and the expectation-maximization (EM) algorithm, which can be applied provided that the local conditional distributions belong to the exponential family. Both methods are iterative and, under certain regularity conditions, they converge to a local maximum of the likelihood. Specializations of HRM's to the case of sequentially structured data (e.g., HMM's and IOHMM's) are typically trained using the EM algorithm. EM iteratively fills in missing values in the data by effectively assigning credit to the hidden state variables given the observed data. An initial value $\boldsymbol{\theta}_0$ is assigned to the parameters. The generic tth iteration of EM consists of an expectation step followed by a maximization step. The expectation step consists of computing the expected sufficient statistics $\boldsymbol{\tau}$ for the parameters, given the observed data and the previous parameters $\boldsymbol{\theta}^{(t-1)}$. In the case of unrestricted multinomial distributions, the sufficient statistics are simply the counts N_{ijk}: how many times the generic variable A_i was found in state k while its parents were found in the jth configuration. The intuition behind EM is that these counts are not available because the state of hidden variables is unknown; however their expectations can be computed

$$\begin{aligned}\tau_{ijk}^{(t)} &= E[N_{ijk}|\mathcal{D}, \boldsymbol{\theta}^{(t-1)}] \\ &= \sum_{\boldsymbol{s}\in\mathcal{D}} P(A_i = a_i^k, \mathrm{pa}_i = \mathrm{pa}_i^j | \boldsymbol{s}, \boldsymbol{\theta}^{(t-1)}).\end{aligned}$$

The above probabilities can be easily computed by solving a probabilistic inference problem with evidence $\boldsymbol{s}$ (i.e., instantiating the label nodes and the target nodes from the data structure $\boldsymbol{s}$ and propagating this evidence into the hidden nodes of the unfolded network with parameters $\boldsymbol{\theta}^{(t-1)}$). The maximization step consists of updating the parameters using the expected sufficient statistics

$$\theta_{ijk}^{(t)} \leftarrow \frac{\tau_{ijk}^{(t)}}{\sum_k \tau_{ijk}^{(t)}}.$$

4) Inference in Binary Tree HRM's: The model we consider in this section is a generalization of IOHMM's for processing binary trees of categorical variables. The recursive network of the model and a slice of the unfolded network are shown in Fig. 11. We derive an evidence propagation algorithm specialized for this topology. The algorithm does not require compilation of the unfolded network into a junction tree and can be seen as a specialization of Pearl's π-λ propagation.

Denote by v a generic node in the data structure S_U and let $X = X_v, U = U_v$, and $Y = Y_v$ denote the hidden variable, the input variable, and the output variable at node v, respectively (we drop the node subscripts to simplify notation). Assuming v is not the root node of S_U, let W be a variable such that $q_L^{-1}W = X$ or $q_R^{-1}W = X$ (i.e., depending whether v is the left or the right child of his parent in S_U) and let V be the input variable connected to W. Moreover, let $S = q_R^{-1}W$ if $q_L^{-1}W = X$, or $S = q_L^{-1}W$ if $q_R^{-1}W = X$ and let Z be the output variable connected to S. Let $\boldsymbol{e} = (\boldsymbol{u}, \boldsymbol{y})$ denote the evidence entered through the input and the output nodes. The aim of probabilistic inference is to assess the table $P(X|\boldsymbol{e})$ for a generic hidden node X. Denote by $\boldsymbol{e}_X^+ = (\boldsymbol{u}_X^+, \boldsymbol{y}_X^+)$ the evidence connected to X through X's ancestors and by $\boldsymbol{e}_X^- = (\boldsymbol{u}_X^-, \boldsymbol{y}_X^-)$ the evidence connected to X through X's descendants. Then we have $P(X|e) \propto P(\boldsymbol{y}_X^-|X, \boldsymbol{u}_X^-)P(X|\boldsymbol{y}_X^+, \boldsymbol{u}_X^+)$. Introduce the "inward" table $\pi(X) = P(X|\boldsymbol{y}_X^+, \boldsymbol{u}_X^+)$ and the "outward" table $\lambda(X) = P(\boldsymbol{y}_X^-|X, \boldsymbol{u}_X^-)$. These tables can be recursively computed by the following inward–outward equations (that generalize the well-known Baum–Welch's forward–backward propagation for HMM's):

$$\begin{aligned}\pi(X) &= \sum_L \sum_R \theta_{X,L,R,u}\pi(L)\pi(R), \\ \lambda(X) &= \vartheta_{y,X} \sum_W \lambda(W) \sum_S \vartheta_{z,S}\theta_{W,S,X,v}\pi(S)\end{aligned} \tag{8}$$

where the parameters θ are transition probabilities and the parameters ϑ are emission probabilities. Finally, the expected sufficient statistics for θ and ϑ are recursively computed (using Bayes' theorem) as

$$\begin{aligned}E[X, L, R|\boldsymbol{u}, \boldsymbol{y}] &\propto \lambda(X)\theta_{X,L,R,u}\pi(L)\pi(R) \\ E[X|\boldsymbol{u}, \boldsymbol{y}] &= \lambda(X)\pi(X).\end{aligned} \tag{9}$$

It is interesting to note a formal resemblance between (8) and the recurrences in the inside–outside algorithm for learning stochastic context free grammars (SCFG's) [38]. However, SCFG's are intended for learning in sequential domains and the tree structures in SCFG's correspond to admissible parse trees, which explain how a given string is generated by the grammar.

B. Recursive Neural Networks

The adaptive processing of data structures based on (2) takes place once we introduce a parametric representation in which the weights can be estimated from examples. In this section, we show that IO-isomorph transductions can be naturally implemented by *recursive neural networks*, a generalized form of recurrent networks where the parameters can be learned from examples by gradient descent algorithms. A recursive neural network is based on the recursive (2), where the state transition function f and the output function g are approximated by feedforward neural networks, leading to the parametric representation

$$\begin{aligned}\boldsymbol{X}_v &= f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v; \boldsymbol{\theta}_f) \\ \boldsymbol{Y}_v &= g(\boldsymbol{X}_v, \boldsymbol{U}_v; \boldsymbol{\theta}_g)\end{aligned}$$

where $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_g$ are connection weights. Note that in the special case of linear chains, the above equations exactly

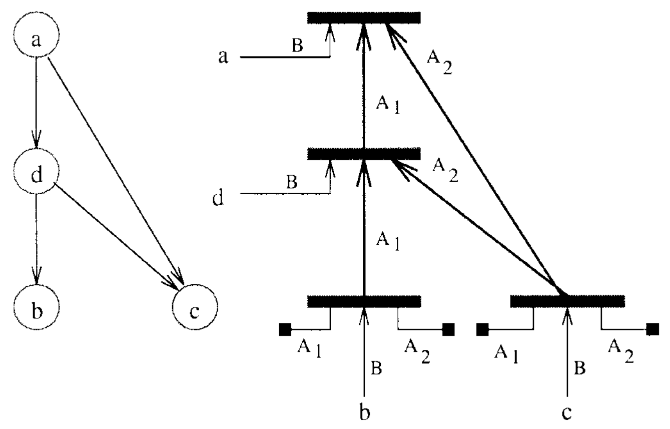

Fig. 12. The *encoding network* associated with a given DOAG. The recursive network is *unfolded through the structure* of the given DOAG.

correspond to the general state-space equations of *recurrent* neural networks. The above representation is stationary, since f and g are independent of node v. Nonstationary transductions could be obtained, for example, by using *node-variant* connection weights $\boldsymbol{\theta}_{f,v}$ and $\boldsymbol{\theta}_{g,v}$. The state variables of recursive representation (2) are continuous, but like in others neural networks, can also be used for coding discrete entities.

The parametric representations $f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v; \boldsymbol{\theta}_f)$, and $g(\boldsymbol{X}_v, \boldsymbol{U}_v; \boldsymbol{\theta}_g)$ can be implemented by a number of different feedforward neural-network architectures. In the following, we present most significant architectures.

- *First-Order Recursive Networks*

Let o be the maximum outdegree of the given DOAG's. The dependence of node v on its children ch$[v]$ can be expressed by means of weight matrices $A_r \in \mathcal{R}^{n,n}, r = 1, \cdots o$. Similarly, the information attached to the nodes can be propagated by a weight matrix $B \in \mathcal{R}^{n,m}$, being $m = |\mathcal{U}|$ and $n = |\mathcal{X}|$. Hence, the parameters of the adaptive model are $\boldsymbol{\theta}_f \doteq \{A_1, \cdots, A_o, B\}$. The state is updated according to

$$\boldsymbol{X}_v = \vec{\sigma}\left(\sum_{k=1}^{o} A_k \cdot q_k^{-1}\boldsymbol{X}_v + B \cdot \boldsymbol{U}_v\right) \qquad (10)$$

where $\vec{\sigma}$ is a vectorial sigmoidal function. This equation is quite a straightforward extension of first-order recurrent neural networks, the only difference being in the generalized form of processing taking place in the "pseudotime" dimension v. Note that the weight matrices A_r and B are the same for every node v and, therefore, the resulting transduction is stationary. Finally, the output at every node can be obtained either directly from some state neurons (through an identity map) or by placing another layer on the top of the state neurons. Fig. 12 is a pictorial representation of the computation taking place in the recursive neural network. According to the graphical formalism developed in Section III-C, a given graph is mapped into the output thanks to the associated encoding network. In Fig. 12, all the recursive neurons are represented by the layer they belong to, and dark squares are used to denote frontier states.

- *Recursive Radial Basis Functions*

Recurrent radial basis functions [39] can be extended to the more general computation needed to process structures by relying on the following parametric representation, for $i = 1, \cdots, p$:

$$X_{i,v}^r = \boldsymbol{\rho}\left(\sum_{k=1}^{o} ||q_k^{-1}\boldsymbol{X}_v - A_{i,k}||^2 + ||\boldsymbol{U}_v - B_i||^2\right) \qquad (11)$$

where $\boldsymbol{\rho}$ is often chosen as an exponential function. $X_{i,v}^r$ in the above equation denote the output of the ith radial basis function unit, and $A_{i,k} \in \mathcal{R}^n$ and $B_i \in \mathcal{R}^m, i = 1, \cdots, p$ are the position parameters. A more complex model can be obtained by adding dispersion parameters to adaptively control the widths of the radial functions. The state vector $\boldsymbol{X}_v$ is obtained using an additional layer of sigmoidal units on the top of radial basis function units, with weight matrix $W \in \mathbb{R}^{n \times p}$ (see [39] for more details). In matrix notation, $\boldsymbol{X}_v = \vec{\sigma}(W\boldsymbol{X}_v^r)$. The parameters controlling the state transition function are therefore $\boldsymbol{\theta}_f \doteq \{A_1, \cdots, A_o, B, W\}$.

- *High-Order Recursive Recurrent Neural Networks*

High-order neural networks, proposed mainly by Giles and associates for both static networks [40] and recurrent networks [41], are very interesting models especially for dealing with symbolic tasks. One can easily conceive high-order networks for processing data structure as an extension of second-order recurrent networks. For instance, in the special case of binary trees, one can introduce third-order networks based on $\boldsymbol{\theta} \doteq \{w_{ijkl}\}$ as follows:

$$X_{i,v} = \sigma\left(\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{m} w_{ijkl} \cdot q_L^{-1}X_{j,v} \cdot q_R^{-1}X_{k,v} \cdot U_{l,v}\right). \qquad (12)$$

The extension to the general case of $(o+1)$-dimensional networks is straightforward.

The processing of data structures according to the scheme defined in Section III and the connectionist specification of functions $f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v; \boldsymbol{\theta}_f)$ and $g(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v; \boldsymbol{\theta}_g)$ make it possible to calculate $\boldsymbol{X}_v$. As for the computation of the outputs, in the case of stationary supersource-based transductions, one simply needs to compute $g(\boldsymbol{X}_s)$, thus producing an unstructured output value whereas, in general, function $g(\boldsymbol{X}_v, \boldsymbol{U}_v)$ must be calculated which produces an output graph with the same skeleton as skel$(\boldsymbol{U})$. Using neural networks for approximating these functions makes it possible to carry out their computation and learning process very effectively. For instance, these functions can be regarded as multilayer perceptrons having either sigmoidal or locally tuned processing units.

Like for the formulation of supervised learning adopted in the case of sequences for recurrent network an error function $E(\cdot)$ can be created, which gives a measure of the fitting of examples of the training set.

No matter what architecture we are considering, learning the parameters associated with the recursive function $f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v; \boldsymbol{\theta}_f)$ can be carried out by relying on the encoding network associated with a given graph. The formalism based on graphical models developed in Section III makes it possible to regard the adaptive computation in the case of graphs as a natural extension of the time-unfolding process for recurrent neural networks.

Like for any connectionist model based on function optimization, learning the parameters can be hard in practice if the error function contains many local minima. A detailed analysis on optimal convergence issues is proposed in [42].

V. Examples of Applications

In the following, we discuss some examples of applications where the proposed framework can fruitfully be applied. First of all, we recall the standard feature-based approach to processing of structures and we discuss the drawbacks of such approach. Then we give a closer look at learning of heuristics and tree-grammars inference. Specifically, we summarize existing work in the context of the proposed framework for applications where it is present, and we motivate why the proposed framework should be applied when considering applications for which the proposed framework has yet to be applied systematically.

A. The Feature-Based Approach to Classification of Structures

Consider a structured domain $\mathcal{Y}^{\#}$, and a training set $T = (X, \xi(X))$ representing a classification task. The standard feature-based approach encodes each graph X as a fixed-size vector of predefined features. Examples of trivial features of graphs are the number of vertices, the number of edges, the mean number of outgoing edges, end so on. These features are usually determined by experts of the application domain and they are used as input to a classifier, e.g., a feedforward neural network. Thus, the encoding process is defined *a priori* and does not depend on the classification task. For example, in molecular biology and chemistry, where chemical compounds are represented as labeled graphs, the encoding process is performed through the definition of *topological indexes* [43], which are designed by a very expensive trial and error approach. The problem of selecting the most relevant features can be partially solved by using the same special selection criterion devised for the classification task at hand. The definition of this criterion, however, can be very difficult when no *a priori* knowledge on the problem is available and it can be useless if the candidate features are not encoding the relevant information for a correct classification of the input graphs, i.e., graphs which must be classified differently are represented by the same feature vector.

Summarizing, we can conclude that the *a priori* definition of the encoding process has two main drawbacks. First, the definition of the candidate features is independent on the classification task. Second, the selection of the relevant features must use *a priori* knowledge. This means that no *general* scheme for the definition and selection of features can be devised. In fact, the relevance of different features of the graphs may change dramatically for different classification tasks, and since the encoding process has to work for any classification task, this implies that such a general scheme would assign a unique feature vector to each graph, thus making the classification task very difficult.

One advantage of the framework we propose in this paper is that, in general, the encoding process is parametric and, apart for some general assumptions such as causality and stationarity, it fully depends on the specific classification task at hand. In fact, by using one of the proposed adaptive processing scheme, the specific encoding procedure is learned on the training data by adapting the set of parameters $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_g$. Thus, for example, given classification problem and a structure $\boldsymbol{U}$ with supersource s, the encoding vector representing $\boldsymbol{U}$ is the state vector $\boldsymbol{X}_s = (\boldsymbol{X}_{\mathrm{ch}[s]}, \boldsymbol{U}_s; \boldsymbol{\theta}_f)$, that depends on the information contained in the whole data structure $\boldsymbol{U}$. This means that, given an application domain where examples of the desired function are available, there is no need to have explicit *a priori* knowledge for encoding a dynamic data structure into an array containing the most relevant features. This encoding process is performed by the recursive state transition function, whose parameters are tuned according to the available training examples. Of course, the quality of the resulting encoding will depend on the amount and the representativity of the training data.

B. Learning Heuristics

As we mentioned in the introduction, a serious problem in symbolic problem solving systems is the combinatorial explosion of the search space. This problem is usually faced by devising heuristics for the selection of the most promising part of the search tree. In this context, by heuristic we mean an *evaluation function* $\mathrm{eval}(\cdot)$ which returns a cost for each branch of the search tree. This cost is expected to be representative of the computational burden and usefulness associated with the exploration of that part of the search tree. The aim is to reach a solution leaf as soon as possible.

Unfortunately, heuristics are domain specific, i.e., a heuristic which is very effective for one domain is usually almost useless for another domain. Moreover, in most cases, heuristics are not known in advance, and they are very difficult to formalize since the automated system usually works on a level of abstraction which is different from that of the expert devising the heuristic. A solution to these problems is to use machine learning techniques to extract control information to be used for devising $\mathrm{eval}(\cdot)$. Specifically, this can be done in the context of supervised learning by collecting positive and negative examples of searches across the search space and then training an inductive system on this set of examples.

As depicted in Fig. 13, the search space in symbolic problem solving systems is represented as a tree whose nodes correspond to search states and whose edges correspond to inference steps. Learning heuristic functions is therefore the problem of learning a real-valued function having the set of admissible search states as domain. While in some applications search states can be conveniently represented in a static form, problems such as automated reasoning yield search states that contain dynamically structured information. For example, states in model-elimination provers and PROLOG-systems consist of logical expressions [44], which are suitable for a labeled DOAG representation, as explained in Example 1.3. In these cases, the evaluation function $\mathrm{eval}(\cdot)$ actually depends on

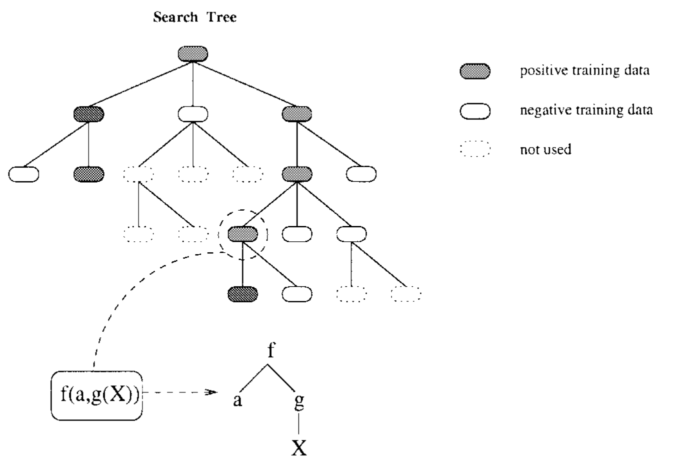

Fig. 13. Example of search-tree. Shaded nodes represent positive training data, while white nodes constitute negative training data. Dashed nodes are not used for training. Note that each node of the tree contains a set of subgoals which can be represented as trees.

structured information (see Fig. 13). In fact, finding a rating for the applicable rules according to the current context, selecting the next subgoal, or realizing a generalizing lemma/failure store, can be regarded as problems of learning a function of symbolic structures of arbitrary size. So, $\mathrm{eval}(\cdot)$ can be defined as

$$\mathrm{eval}(\cdot) \colon \mathcal{U}^{\#} \to \mathbb{R}. \tag{13}$$

For this reason, both recursive neural networks and HRM's are natural candidates for performing the inductive task.

In the context of neural networks, a good deal of work as been performed on the SETHEO system. SETHEO is a theorem prover for full first-order logic. It uses model elimination for proof calculus. More details on SETHEO can be found in [45]. A first attempt to learn an evaluation function in SETHEO was performed by training a backpropagation network on feature vectors representing predefined features for the structures encountered during the search, like the proof-context and the applicable inference steps [46]. Examples of *static* features, i.e., features which can be computed at compilation time, are the number of literals in a clause, or the number of distinct variables in the head of a clause. Examples of *uniform dynamical* features, i.e., features computed at run time but uniform for all branches of a logical OR branch-point, are the current depth in the search tree or the number of instantiated variables in the calling subgoal. Finally, there are *distinct dynamical* features, i.e., which depend on the actual clause occurrence at run time, like the current total number of uses of a given clause in the current search or the number of variables of calling clause becoming instantiated.[9] The results obtained with this network and other standard neural-network models [47] improved the search time of one order of magnitude, however, as is clear from the discussion in Section V-A and the examples of features given above, the encoding of structural information in a fixed-size vector is not able to capture all the relevant information gathered by symbolic structures of arbitrary size. In fact, it is important to include the selection of the most relevant features as part of the learning task. A preliminary move toward this direction is reported in [48], where LRAAM-based networks were successfully used to perform classification of symbolic recursive structures encoding logical terms. A refinement of this work led to the definition of the backpropagation through structure algorithm [47]. The experimental comparison between the LRAAM-based approach and the backpropagation through structure algorithm reported in [47] shows that the latter algorithm obtains slightly better results for all examples solvable by the LRAAM-based approach, but with smaller networks and training times. Moreover, the backpropagation through structure algorithm was able to successfully learn classification tasks that could not be solved in a reasonable amount of time by LRAAM-based networks. Another improvement in this direction has been the development of the cascade-correlation network for structure [48], [19] (a generalization of recurrent cascade-correlation for sequences [50]) which obtained better results with respect to the LRAAM-based networks on a subset of the classification problems reported in [48]. One advantage of the cascade-correlation network for structure over backpropagation through structure networks is that the necessary number of hidden units is automatically determined by the learning algorithm. The integration of backpropagation through structure networks within the SETHEO system is discussed in great detail in [5], where a more sophisticated formulation of the learning goal is given. The basic idea is that a heuristic is good even if it enables the system to find a single solution, as long as the length of the path leading to this solution is reasonable with respect to the length of shortest path leading to a solution. The merging of this new formulation of the learning goal with the use of neural networks for structures resulted both in a seed-up of the search and the discovering of a solution for problems for which no solution was found (in a reasonable amount of computation) before.

C. Inference of Tree Grammars

In classical grammatical inference, a learner is presented a set of labeled strings and is requested to infer a set of rules that define a formal language [51]. Stated in this classical way, the domain for grammatical inference consists of learning sets of strings, i.e., sets of sequentially ordered pieces of information. In particular, assuming the language associated with the grammar is a regular set, grammatical inference consists of identifying a (possibly small) finite accepting automaton. During the past years, research was carried out to generalize conventional automata theory by changing the type of inputs and outputs, from strings to labeled trees. Recently, numerous researchers have approached grammatical inference using adaptive models such as recurrent neural networks [22][10] or IOHMM's [24].

[9] Note that this feature is different from the number of instantiated variables in the calling subgoal, since the subgoal is the same for each clause for which unification is tried.

[10] It is well known that recurrent neural networks can simulate any finite state automata [52] as well as any multistack Turing machine in real time [53]. However, the ability of representing finite automata is not sufficient to guarantee that a given regular grammar can be actually learned from examples [54]. A detailed discussion of the language identification problem on connectionist grammatical inference systems can be found in [55].

In this section, we discuss the relationships of the proposed framework with inference of tree grammars.

A *tree grammar* is defined as a four-tuple $G_t = (V, r, P, S)$ where $V = N \cup \Sigma$ is the grammar alphabet (nonterminals and terminals); (V, r) a ranked alphabet; productions in P are of the form $T_i \rightarrow T_j$, where T_i and T_j are trees; and S in T_V is a finite set of "starting trees," where T_V denotes the set of trees with nodes labeled by elements in V. A tree grammar is in *expansive form* if all its productions are of the form

$$\begin{tikzcd}[column sep=small] X \arrow[r] & & x \arrow[dl, dash] \arrow[dr, dash] & \\ & X_1 & \cdots & X_n. \end{tikzcd}$$

Tree automata generalize finite state automata and it is known (see, for example, [8]) that languages of trees generated by tree grammars in expansive form can be recognized by *frontier to root tree automata* (FRA) (as defined in Example 3.1). In [56] computational results on three different recursive neural-network models (for structures), namely Elman-style networks, cascade-correlation networks, and neural trees, were reported. Specifically, it is shown that Elman-style networks can simulate any FRA, while neither cascade-correlation networks for structures nor neural trees for structures can simulate any FRA. From these results, and observing that neural trees for structures are generalizations of neural trees for sequences, it is also shown that neural trees for sequences cannot simulate any finite state machine. Note how results derived on structures are also true on sequences since sequences are special cases of labeled graphs. This means that a fully developed theory on structures will automatically cover sequences. Moreover, it can be observed that HRM's are equivalent to FRA's in the limit case of $0-1$ probabilities (in that case, the state transition function of the FRA is simply obtained by taking the arguments of the nonzero parameters in the CPT associated with hidden state variables in the HRM). In fact, experimental results have been reported in [36] showing that binary-tree HRM's are able to learn regular grammars on binary tree domains.

VI. Conclusions

The framework discussed in this paper opens a novel way of adaptively dealing with pieces of information among which relations are expressed in the form of directed acyclic graphs. Our formalism can be intuitively explained by extending models for serially ordered sequences to models capable of dealing with partial orders among the variable described by DOAG's. Therefore, it seems quite natural to expect that many theoretical analyzes for recurrent networks and HMM's can be generalized to the recursive models presented in this paper. In particular, one interesting issue is the introduction of symbolic prior knowledge into adaptive structural processors. Several prior knowledge injection algorithms have been introduced for recurrent networks, assuming that the available prior knowledge can be expressed in terms of state transition rules for a finite automaton (see [52] for a very detailed theoretical analysis). Known transitions are injected into the recurrent architecture, while unknown rules are supposed to be filled in by data-driven learning. The approach aims to reduce the complexity of learning and to improve generalization. After learning, one can also extract rules from the trained network, in order to complete the refinement process in the symbolic domain [57], [22], [39], [58]. Because of the strict relations between recurrent and recursive networks pointed out in this paper, most of the methods adapted to incorporate and extract symbolic knowledge for the case of linear lists are likely to be extended to the case of graphs. For instance, the natural relationship between automata and second-order recurrent neural networks can be extended to a relationship between $m$ary tree automata and $(m+1)$th order recursive networks. Unlike recurrent networks used to process sequences where the partial specification of the transduction can only involve the information attached to the nodes of the list, in the case of recursive networks, the graph topology can be an additional rich source of prior knowledge. Because of the hybrid nature of the data, there are cases in which the graph topology can itself be used to create a partition of the training set. The prior knowledge on the the graph topology can be exploited for designing a *modular recursive network*, in which the single modules are learned separately on the basis of the partition of the training set operated by the prior topology information.

Finally, we would like to remark that the extension of the framework we have proposed to manage transductions which are not IO-isomorph is an open research problem. In the case of non IO-isomorph transductions, defining encoding networks for structured processing is highly nontrivial and a quite difficult task that need to be solved during learning is the structural alignment of the input and output DOAG's (i.e., to determine which output subgraphs are generated in correspondence of which input subgraphs). In the particular case of temporal sequences, solutions to the problem of learning asynchronous transductions have been proposed with recurrent networks [59] and IOHMM's [60].

Acknowledgment

The authors thank P. Baldi, C. Goller, A. Küchler, and A. C. Tsoi for very fruitful discussions and comments on an earlier version of this paper.

References

[1] P. Frasconi, M. Gori, and G. Soda, "Recurrent neural networks and prior knowledge for sequence processing: A constrained nondeterministic approach," *Knowledge-Based Syst.*, vol. 8, no. 6, pp. 313–332, 1995.

[2] T. J. M. Cabe, "A software complexity measure," *IEEE Trans. Software Eng.*, vol. 2, pp. 308–320, 1976.

[3] R. Prather, "Design and analysis of hierarchical software metrics," *ACM Computing Surveys*, vol. 27, no. 4, pp. 497–518, 1995.

[4] N. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.

[5] C. Goller, "A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems," Ph.D. dissertation, Dept. Comput. Sci., Tech. Univ. Munich, 1997.

[6] K. S. Fu, *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, N.J: Prentice-Hall, 1982.

[7] T. Pavlidis, *Structural Pattern Recognition*. New York: Springer-Verlag, 1977.

[8] R. C. Gonzalez and M. G. Thomason, *Syntactical Pattern Recognition*. Reading, MA: Addison-Wesley, 1978.

[9] R. J. Schalhoff, *Pattern Recognition: Statistical, Structural and Neural Approaches*. New York: Wiley, 1992.

[10] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, pp. 157–166, Mar. 1994.
[11] G. E. Hinton, "Mapping part-whole hierarchies into connectionist networks," *Artificial Intell.*, vol. 46, pp. 47–75, 1990.
[12] J. B. Pollack, "Recursive distributed representations," *Artificial Intell.*, vol. 46, no. 1–2, pp. 77–106, 1990.
[13] T. A. Plate, "Holographic reduced representations," *IEEE Trans. Neural Networks*, vol. 6, pp. 623–641, May 1995.
[14] A. Sperduti, "Labeling RAAM," *Connection Sci.*, vol. 6, no. 4, pp. 429–459, 1994.
[15] ——, "Encoding labeled graphs by labeling RAAM," in *Advances in Neural Information Processing Systems*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. San Mateo, CA: Morgan Kaufmann, vol. 6, pp. 1125–1132, 1994.
[16] ——, "Stability properties of labeling recursive autoassociative memory," *IEEE Trans. Neural Networks*, vol. 6, pp. 1452–1460, Nov. 1995.
[17] A. Sperduti and A. Starita, "A memory model based on lraam for associative access of structures," in *IEEE Int. Conf. Neural Networks*, 1996, pp. 543–548.
[18] V. Cadoret, "Encoding syntactical trees with labeling recursive autoassociative memory," in *Proc. ECAI*, Amsterdam, The Netherlands, 1994, pp. 555–559.
[19] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Trans. Neural Networks*, vol. 8, 1997.
[20] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *J. Complex Syst.*, vol. 1, pp. 145–168, Feb. 1987.
[21] J. L. Elman, "Finding structure in time," *Cognitive Sci.*, vol. 14, pp. 179–211, 1990.
[22] C. L. Giles and C. W. Omlin, "Extraction, insertion, and refinement of symbolic rules in dynamically driven recurrent neural networks," *Connection Sci.*, vol. 5, no. 3, pp. 307–337, 1993.
[23] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, 1989.
[24] Y. Bengio and P. Frasconi, "Input–output HMM's for sequence processing," *IEEE Trans. Neural Networks*, vol. 7, pp. 1231–1249, Sept. 1996.
[25] B. de Vries and J. C. Principe, "The gamma model—A new neural-net model for temporal processing," *Neural Networks*, vol. 5, pp. 565–576, 1992.
[26] A. D. Back and A. C. Tsoi, "A comparison of discrete-time operator models and for nonlinear system identification," in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds. Cambridge, MA: MIT Press, vol. 7, 1995, pp. 883–890.
[27] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.
[28] J. Whittaker, *Graphical Models in Applied Multivariate Statistics*. Chichester, U.K.: Wiley, 1990.
[29] R. M. Neal, "Asymmetric parallel Boltzmann machines are belief networks," *Neural Comput.*, vol. 4, no. 6, pp. 832–834, 1992.
[30] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Clarendon, 1995.
[31] S. Lauritzen, *Graphical Models*, no. 17 in Statistical Science Series. Oxford, U.K.: Clarendon, 1996.
[32] W. L. Buntine, "Operations for learning with graphical models," *J. Artificial Intell. Res.*, vol. 2, pp. 159–225, 1994.
[33] M. L. Forcada and R. C. Carrasco, "Learning the initial state of a second-order recurrent neural network during regular-language inference," *Neural Comput.*, vol. 7, no. 5, pp. 923–930, 1995.
[34] J. Thacher, "Tree automata: An informal survey," in *Currents in the Theory of Computing*, A. Aho, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1973, pp. 143–172.
[35] F. V. Jensen, S. L. Lauritzen, and K. G. Olosen, "Bayesian updating in recursive graphical models by local computations," *Comput. Statist. Quarterly*, vol. 4, pp. 269–282, 1990.
[36] P. Frasconi, M. Gori, and A. Sperduti, "Hidden recursive models for probabilistic learning of structured information," preprint, Dipartimento di Sistemi e Informatica, Università di Firenze, Firenze, Italy, 1997.
[37] L. K. Saul, T. Jaakkola, and M. I. Jordan, "Mean field theory for sigmoid belief networks," *J. Artificial Intell. Res.*, vol. 4, pp. 61–76, 1996.
[38] K. Lari and S. J. Young, "The estimation of stochastic context-free grammars using the inside–outside algorithm," *Comput. Speech and Language*, vol. 4, no. 1, pp. 35–56, 1990.
[39] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite-state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, pp. 5–32, 1996.
[40] C. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Appl. Opt.*, vol. 26, no. 23, p. 4972, 1987.
[41] C. B. Miller and C. L. Giles, "Experimental comparison of the effect of order in recurrent neural networks," *Int. J. Pattern Recognition Artificial Intell.*, (Special issue on *Applications of Neural Networks to Pattern Recognition*), I. Guyon, Ed., 1993.
[42] P. Frasconi, M. Gori, and A. Sperduti, "Optimal learning of data structures," in *Proc. Int. Joint Conf. Artificial Intell.*, Nagoya, Japan, August 23–29, 1997, pp. 1066–1071.
[43] L. H. Hall and L. B. Kier, "The molecular connectivity chi indexes and kappa shape indexes in structure-property modeling," in *Reviews in Computational Chemistry*, K. B. Lipkowitz and D. B. Boyd, Eds. New York: VCH, 1991, pp. 367–422.
[44] L. Wos, R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning: Introduction and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1984.
[45] R. Letz, J. Schumann, S. Bayerl, and W. Bibel, "Setheo: A high-performance theorem prover," *J. Automated Reasoning*, vol. 8, no. 2, pp. 183–212, 1992.
[46] C. B. Suttner and W. Ertel, "Using backpropagation for guiding the search of a theorem prover," *Int. J. Neural Network Res. Applicat.*, vol. 2, no. 1, pp. 3–16, 1991.
[47] C. Goller and A. Küchler, "Learning task-dependent distributed structure-representations by backpropagation through structure," in *IEEE Int. Conf. Neural Networks*, 1996, pp. 347–352.
[48] A. Sperduti, A. Starita, and C. Goller, "Learning distributed representations for the classification of terms," in *Proc. Int. Joint Conf. Artificial Intell.*, 1995, pp. 509–515.
[49] A. Sperduti, D. Majidi, and A. Starita, "Extended cascade-correlation for syntactic and structural pattern recognition," in *Advances in Structural and Syntactical Pattern Recognition*, P. Perner, P. Wang, and A. Rosenfeld, Eds., vol. 1121 of *Lecture notes in Computer Science*. Berlin, Germany: Springer-Verlag, 1996, pp. 90–99.
[50] S. E. Fahlman, "The recurrent cascade-correlation architecture," in *Advances in Neural Information Processing Systems*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, vol. 3, 1991, pp. 190–196.
[51] D. Angluin and C. H. Smith, "A survey of inductive inference: Theory and methods," *ACM Comput. Survey*, vol. 15, pp. 237–269, Sept. 1983.
[52] C. Omlin and C. L. Giles, "Constructing deterministic finite-state automata in recurrent neural networks," *J. ACM*, vol. 43, no. 6, pp. 937–972, 1996.
[53] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *J. Comput. Syst. Sci.*, vol. 50, no. 1, pp. 132–150, 1995.
[54] E. M. Gold, "Complexity of automaton identification from given data," *Inform. Contr.*, vol. 37, pp. 302–320, 1978.
[55] S. C. Kremer, "A Theory of Grammatical Induction in the Connectionist Paradigm," Ph.D. dissertation, Dept Comput. Sci., Univ. Alberta, Canada, 1996.
[56] A. Sperduti, "On the computational power of recurrent neural networks for structures," *Neural Networks*, to appear, 1997.
[57] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee, "Learning and extracting finite state automata with second-order recurrent neural networks," *Neural Comput.*, vol. 4, no. 3, pp. 393–405, 1992.
[58] G. G. Towell and J. W. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine Learning*, vol. 13, pp. 71–101, 1993.
[59] R. Neco and M. L. Forcada, "Beyond mealy machines: learning translators with recurrent neural," in *Proc. WCNN'96 (World Congr. Neural Networks)*, San Diego, CA, pp. 408–411.
[60] Y. Bengio and F. Gingras, "Recurrent neural networks for missing or asynchronous data," in *Advances in Neural Information Processing Systems*, M. Mozer, D. Touretzky, and M. Perrone, Eds., vol. 8. Cambridge, MA: MIT Press, 1996.

Paolo Frasconi (S'91–M'94) received the M.Sc. degree in electronic engineering in 1990 and the Ph.D. degree in computer science in 1994, both from the University of Florence, Italy.

He is currently Assistant Professor with the Dipartimento di Sistemi e Informatica at the University of Florence. In 1992 he was a Visiting Scholar in the Department of Brain and Cognitive Science at the Massachusetts Institute of Technology, Cambridge. In 1994 he was a Visiting Scientist at Centro Studi e Laboratori Telecomunicazioni (CSELT), Turin. His current research interests include neural networks, Markovian models, and graphical models, with particular emphasis on problems involving learning about sequential and structured information.

Marco Gori (S'88–M'91–SM'97) received the Laurea degree in electronic engineering from Università di Firenze, Italy, in 1984, and the Ph.D. degree in 1990 from Università di Bologna, Italy.

He was also a Visiting Student at the School of Computer Science, McGill University, Montreal, Canada. In 1992, he became an Associate Professor of Computer Science at Università di Firenze and, in November 1995, he joined the University of Siena, Italy. His main research interests are in pattern recognition, neural networks, and artificial intelligence.

Dr. Gori was the general chairman of the Second Workshop of Neural Networks for Speech Processing held in Firenze in 1992, organized the NIPS'96 postconference workshop on "Artificial Neural Networks and Continuous Optimization: Local Minima and Computational Complexity," and coorganized the Caianiello Summer School on "Adapting Processing of Sequences" held in Salerno on September 1997. He coedited the volume *Topics in Artificial Intelligence* (Berlin, Germany: Springer-Verlag, 1995) which collects the contributions of the 1995 Italian Congress of Artificial Intelligence. He serves as a Program Committee member of several workshops and conferences mainly in the area of Neural Networks and acted as Guest Coeditor of the *Neurocomputing Journal* for the special issue on recurrent neural networks (July 1997). He is an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS, *Neurocomputing*, and *Neural Computing Survey*. He is the Italian chairman of the IEEE Neural Network Council (R.I.G.) and is a member of the IAPR, SIREN, and AI*IA Societies.

Alessandro Sperduti received the laurea and Doctoral degrees in 1988 and 1993, respectively, both in computer science, from the University of Pisa, Italy.

In 1993 he spent a period at the International Computer Science Institute, Berkley, CA, supported by a postdoctoral felloship. In 1994 he returned to the Computer Science Department at the University of Pisa, where he is presently Assistant Professor. His research interests include data sensory fusion, image processing, neural networks, and hybrid systems. In the field of hybrid systems his work has focused on the integration of symbolic and connectionist systems.

Dr. Sperduti has contributed to the organization of several workshops on this subject and also served on the program committee of conferences on neural networks.