Integration of Graphical Rules with Adaptive Learning of Structured Information

Paolo Frasconi[1], Marco Gori[2], Alessandro Sperduti[3]

[1] Dipartimento di Ingegneria Elettrica ed Elettronica, Università di Cagliari
Piazza d'Armi, 09123 Cagliari Italy
[2] Dipartimento di Ingegneria dell'Informazione, Università di Siena
Via Roma 56, 53100 Siena, Italy
[3] Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, I-56125 Pisa, Italy

Abstract. We briefly review the basic concepts underpinning the adaptive processing of data structures as outlined in [3]. Then, turning to practical applications of this framework, we argue that stationarity of the computational model is not always desirable. For this reason we introduce very briefly our idea on how a priori knowledge on the domain can be expressed in a graphical form, allowing the formal specification of perhaps very complex (i.e., non-stationary) requirements for the structured domain to be treated by a neural network or Bayesian approach. The advantage of the proposed approach is the systematicity in the specification of both the topology and learning propagation of the adopted computational model (i.e., either neural or probabilistic, or even hybrid by combining both of them).

1 Introduction

Several disciplines have taken advantage of structured representations. These include, among many other, knowledge representation, language modeling, and pattern recognition. The interest in developing connectionist architectures capable of dealing with these rich representations (as opposed to "flat" or vector-based representations) can be traced back to the end of the 80's.

Different approaches have been proposed. Touretzky's BoltzCONS system [15] is an example of how a Boltzman machine can handle symbolic structures using coarse-coded memories [11] as basic representational elements, and LISP's *car, cdr*, and *cons* functions as basic operations. The RAAM model proposed by Pollack [10] is based on backpropagation (Backpropagation is only one particular way to implement the concept underlying the RAAM model) to discover compact recursive distributed representations of trees with a fixed branching factor. Recursive distributed representations are an instance of the concept of a reduced descriptor introduced by Hinton [6] to solve the problem of mapping part-whole hierarchies into connectionist networks.

Also related to the concept of a reduced descriptor are Plate's holographic reduced representations [9]. A formal characterization of representations of structures in connectionist systems using the tensor product was developed by Smolensky [13].

Examples of application domains where structures are extensively used are medical and technical diagnoses (discovery and manipulation of structured dependencies, constraints, explanations), molecular biology (DNA and protein analysis), chemistry (classification of chemical structures, quantitative structure-property relationship (QSPR), quantitative structure-activity relationship (QSAR)), automated reasoning (robust matching, manipulation of logical terms, proof plans, search space reduction), software engineering (quality testing, modularization of software), geometrical and spatial reasoning (robotics, structured representation of objects in space, figure animation, layouting of objects), speech and text processing (robust parsing, semantic disambiguation, organizing and finding structure in texts and speech).

While algorithms that manipulate symbolic information are capable of dealing with highly structured data, adaptive neural networks are mostly regarded as learning models for domains in which instances are organized into *static* data structures, like records or fixed-size arrays. Recurrent neural networks, that generalize feedforward networks to sequences (a particular case of dynamically structured data) are perhaps the best known exception.

However, during the last few years, neural networks for the representation and processing of structures have been developed [14]. In particular, recursive neural networks are a generalization of recurrent networks for processing sequences (i.e., linear chains from a graphical point of view) to the case of directed acyclic graphs. These kind of networks are of paramount importance for both structural pattern recognition and for the development of hybrid neural-symbolic systems, since they allow the treatment of structured information very naturally and, in several cases, very efficiently.

The main motivations for using neural networks for the processing of structures are several: first of all, neural networks are universal function approximators, then they can perform automatic inference (learning), hold very good classification capabilities, and they can deal with noise and incomplete data. All the above properties are particularly useful when facing classification problems in a structured domain.

Let us consider, for example, the use of neural networks for classification of labeled graphs which represent chemical compounds. The standard approach with feedforward neural networks consists of encoding each graph as a fixed-size vector, which is then used for feeding the neural network. Unfortunately, the a priori definition of the encoding process has several drawbacks. For example, in chemistry, the encoding is performed through the definition of *topological indices* which are properly designed by means of a very expensive trial and error approach) and the resulting vectorial representations of graphs may be very difficult to classify. Recursive neural networks face this problem by simultaneously learning either how to represent or how to classify structured patterns. This is a very desirable property which has already been shown to be useful in practice.

It must be noted that automatic inference can also be obtained by using a symbolic representation, such as in *Inductive Logic Programming* [7]. Recursive neural networks, however, have their own specific peculiarity, since they can approximate functions from a structured domain (possibly with real valued vectors as labels) to the set of reals. To the best of our knowledge, this cannot be performed by any symbolic system.

In many applications, however, the assumption of stationarity of the computational model, which is typically done when using neural or Bayesian networks, does not fit with the nature of the problem at hand. This is especially true when considering complex problems which may involve heterogeneous (i.e., both symbolic and numerical) and structured data. In this chapter, after a brief review of the basic concepts underpinning the adaptive processing of structures as outlined in [3], we show by two examples from practical applications that the assumption of stationarity may not be adequate to the complexity of the application domains. Following this argument, in this chapter we discuss some ideas about how to define a graphical formalism for the systematic specification and processing of a priori knowledge on the application domain.

2 Structured Domains

In this chapter, we consider domains of DOAGs (Directed Ordered Acyclic Graphs) where vertices are marked by *labels*, i.e. subsets of domain variables. The meaning of an edge (v, w) is that the variables in the labels attached to v and w are related. A graph is *uniformly labeled* if: *i)* an equivalence relation is defined on the domain variables; *ii)* every label contains one and only one variable from each equivalence class. For example, in a pattern recognition domain, when classifying images containing geometrical patterns, *Perimeter*, *Area*, *Texture* are all possible equivalence classes since all geometrical patterns can be represented using these features.

Let us assume uniformly labeled graphs (with N equivalence classes). Let Y_1, Y_2, ..., Y_N be representative variables for the equivalence classes. Y_i is said categorical when the realizations belong to a finite alphabet $\mathcal{Y}_i$. Y_i is said numerical when the realizations are real numbers: $\mathcal{Y}_i = \mathbb{R}$. Moreover, the set of label realizations

$$\mathcal{Y} = \mathcal{Y}_1 \times \mathcal{Y}_2 \times \cdots \times \mathcal{Y}_N$$

is called *label space*.

The class of DOAGs in which we are interested in, is formed by directed acyclic graphs such that, for each vertex v, a total order $\prec$ is defined on the edges leaving from v.

$$\text{E.g.:} \qquad (v, w) \prec (v, u) \prec (v, t)$$

A trivial example of structure is a sequence (see Figure 1), which can be defined as either: *i)* an external vertex, or *ii)* an ordered pair (t, h) where the *head* h is a vertex and the *tail* t is a sequence.

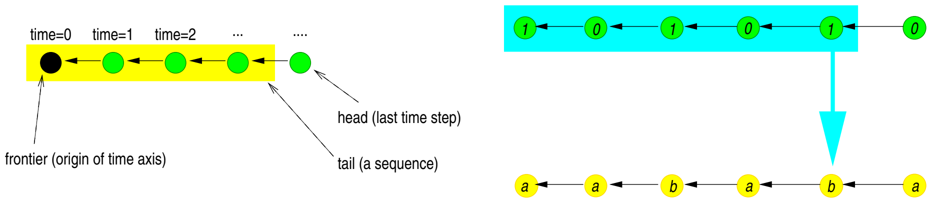


Fig. 1. *left:* Example of sequence; *right:* Causality for sequential transductions.

2.1 Sequential Transductions

Here we review very briefly the concept of *sequential transduction*. Let $\mathcal{U}$ and $\mathcal{Y}$ be the *input* and *output* label spaces. We denote by $\mathcal{U}^*$ the set of all sequences with labels[4] in $\mathcal{U}$.

A general transduction $\mathcal{T}$ is a subset of $\mathcal{U}^* \times \mathcal{Y}^*$. We shall limit our discussion to functions: $\mathcal{T} : \mathcal{U}^* \rightarrow \mathcal{Y}^*$. A transduction $\mathcal{T}(\cdot)$ is *causal* if the output at time t does not depend on *future* inputs (at time $t+1, t+2, \ldots$)(see Figure 1).

A particular form of transduction can be obtained by having the outputs to depend on *hidden state variables* (label space $\mathcal{X}$), i.e., for each time t:

$$\boldsymbol{X}_t = f(\boldsymbol{X}_{t-1}, \boldsymbol{U}_t, t) \text{ and } \boldsymbol{Y}_t = g(\boldsymbol{X}_t, \boldsymbol{U}_t, t)$$

where $f : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{X}$ is the *state transition function* and $g : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{Y}$ is the *output function*, and the initial state is associated with the external vertex (frontier).

A recursive state representation exists only if $\mathcal{T}(\cdot)$ is *causal* (see Figure 2). Moreover, $\mathcal{T}(\cdot)$ is *stationary* if $f(\cdot)$ and $g(\cdot)$ do not depend on t.

2.2 Transductions for Data Structures

Here we generalize causal and stationary transductions to structured domains. Let $\mathcal{U}^\#$ and $\mathcal{Y}^\#$ be two DOAG spaces. A transduction $\mathcal{T}$ is a subset of $\mathcal{U}^\# \times \mathcal{Y}^\#$.

As in the sequential case, we impose the following restrictions: *i)* $\mathcal{T}$ is a function $\mathcal{T} : \mathcal{U}^\# \rightarrow \mathcal{Y}^\#$; *ii)* $\mathcal{T}$ is *IO-isomorph*, i.e., $\mathrm{skel}(\mathcal{T}(\boldsymbol{U})) = \mathrm{skel}(\boldsymbol{U})$ ($\mathrm{skel}(\boldsymbol{U})$ returns a graph with the same topology as $\boldsymbol{U}$ but without labels attached to the vertices), where skel() is an operator returning the skeleton of the graph, i.e., the graph obtained by ignoring all node labels; *iii)* # is an ordered class of DAGs.

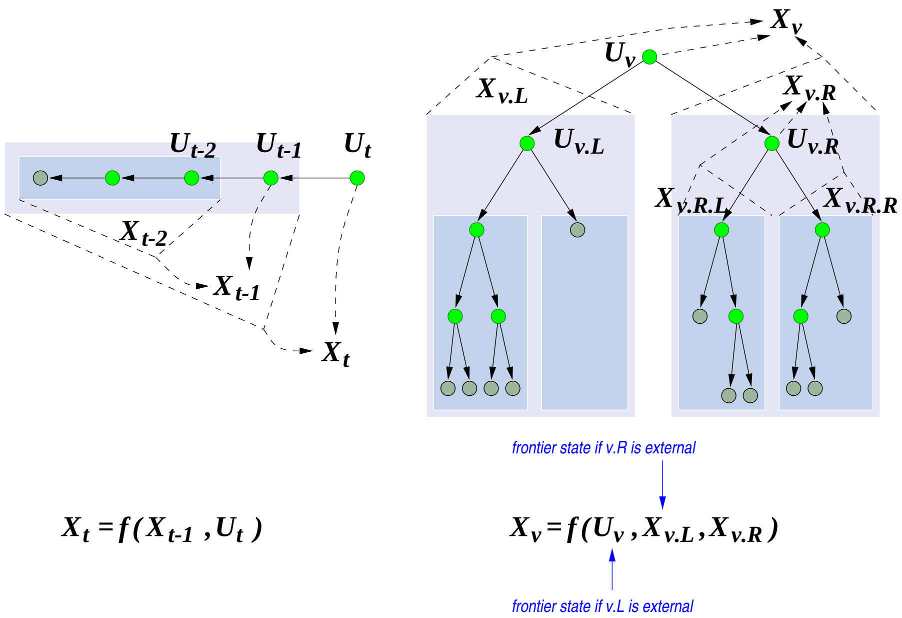
Given an input graph $\boldsymbol{U}$, a recursive state representation for a structural transduction can be defined so that, for each vertex v:

$$\boldsymbol{X}_v = f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v) \text{ and } \boldsymbol{Y}_v = g(\boldsymbol{X}_v, \boldsymbol{U}_v) \tag{1}$$

[4] Cf. the classical definition in language theory: $\mathcal{U}$ is a set of terminal symbols and $\mathcal{U}^*$ is the free monoid over $\mathcal{U}$.

where $\mathrm{ch}[v]$ are the (ordered) children of v, $f : \mathcal{X}^m \times \mathcal{U} \rightarrow \mathcal{X}$ is the *state transition function* and $g : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{Y}$ is the *output function*.

Also for structural transductions a recursive state representation exists only if $\mathcal{T}(\cdot)$ is *causal*, and $\mathcal{T}(\cdot)$ is *stationary* if $f(\cdot)$ and $g(\cdot)$ do not depend on v (see Figure 2).



(a) (b)

Fig. 2. *(a):* Sequential transduction with hidden recursive state representation; *(b):* Structural transduction with hidden recursive state representation.

Note that the update of equations (1) may follow any *reverse topological sort* of the input graph (see Figure 3). Specifically, some vertices can be updated in parallel according to the preorder defined by the topology of the input graph.

We are particularly interested in *supersource transductions*, which can be characterized by the following points: *i)* the input graph $\boldsymbol{U}$ is a DOAG with supersource s; *ii)* the output is a single label $\boldsymbol{Y}$, which is either a categorical variable, in case of classification, or a (multivariate) numerical variable, in case of regression; *iii)* the hidden state $\boldsymbol{X}_v$ is updated by $f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v)$, while the output label is "emitted" at s: $\boldsymbol{Y} = g(\boldsymbol{X}_s)$.

In order to graphically describe structured transductions, it is convenient to define *generalized shift operators*. In case of sequences, the standard shift operator is defined as $q^{-1}\boldsymbol{Y}_t = \boldsymbol{Y}_{t-1}$ (unitary time delay). This definition can be extended for DOAGs as follows: $q_k^{-1}\boldsymbol{Y}_v$ is the label attached to the k-th child

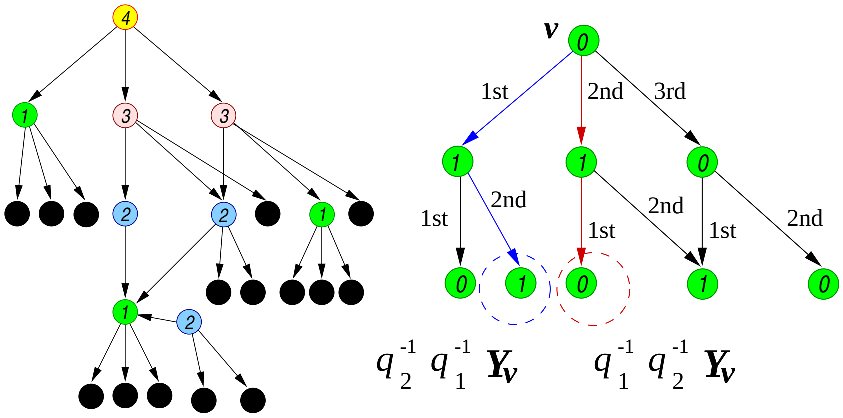


Fig. 3. *left:* Example of reverse topological sort for a DOAG: any total order on the nodes consistent with the numbering of the nodes is admissible (nodes with same number can be permuted); *right:* Example showing the non-commutativity of the generalized shift operators.

of vertex v. Notice that the composition of these operators is not commutative (see Figure 3).

Having defined generalized shift operators, we can now define a class of *graphical models* called *recursive networks*, used to represent structural transductions. We assume that: *i)* $\mathcal{T}()$ admits a recursive state space representation; *ii)* input, state, and output DOAGs are uniformly labeled.

The recursive network of $\mathcal{T}()$ is a directed graph where: *i)* vertices are marked with representative variables; *ii)* edges are marked with generalized shift operators; *iii)* an edge (A_v, B_v), with label q_k^{-p}, means that for each v in the vertex set of the input structure, B_v is a function of $q_k^{-p} A_v$.

Given a graph $\boldsymbol{U} \in \mathcal{U}^{\#}$ and a recursive transduction $\mathcal{T}$, it is possible to compute the output of the transduction $\mathcal{T}$ by resorting to *encoding networks*: the *encoding network* associated with $\boldsymbol{U}$ and $\mathcal{T}$ is formed by unrolling the recursive network of $\mathcal{T}$ through the input graph $\boldsymbol{U}$. Note that *time-unfolding* is a special case of the above concept for the class of sequences (see Figure 4).

The encoding network is a graphical representation of the functional dependences between the variables in consideration *given* a *specific* DOAG in input. This encoding network can be implemented either by a *neural network* or a *Bayesian network* [3]. For example, let us consider neural networks. In this case, both the state transition function $f(\cdot)$ and the output function $g(\cdot)$ are realized by feedforward neural networks, leading to the parametric representation

$$\boldsymbol{X}_v = f(\boldsymbol{X}_{\mathrm{ch}[v]}, \boldsymbol{U}_v, \boldsymbol{\theta}_f), \text{ and } \boldsymbol{Y}_v = g(\boldsymbol{X}_v, \boldsymbol{U}_v, \boldsymbol{\theta}_g),$$

where $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_g$ are connection weights. In the special case of linear chains, the above equations exactly correspond to the general state space equations

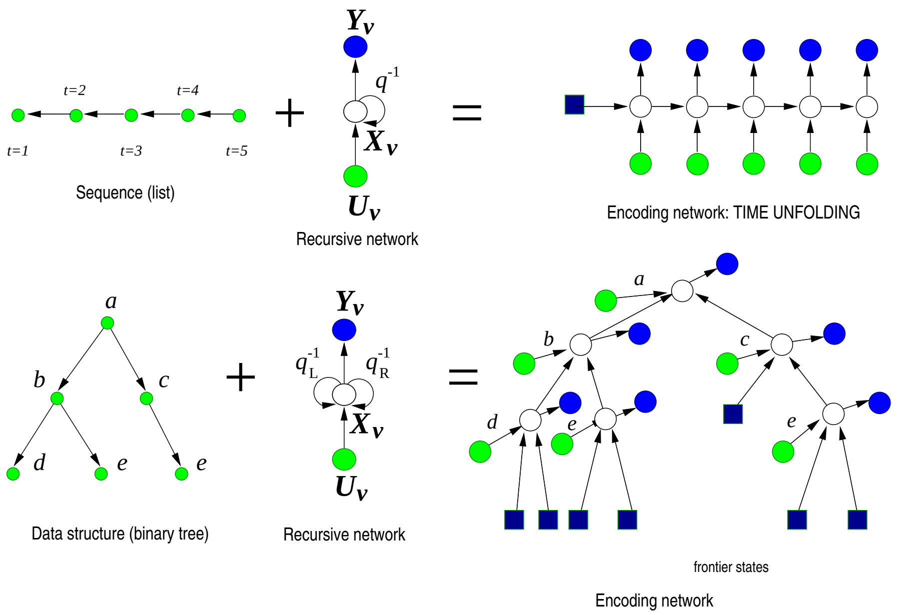


Fig. 4. Example of generation of encoding networks for sequential and structural transductions.

of *recurrent* neural networks. Moreover, the above representation is stationary, since both $f(\cdot)$ and $g(\cdot)$ do not depend on the node v.

The encoding network, in this case, will result in a feedforward neural network obtained by replicating and connecting the feedforward neural networks implementing $f(\cdot)$ and $g(\cdot)$, according to the topology of the input DOAG. Standard learning algorithms for neural networks can be used to train the neural encoding network, taking care of the fact that $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_g$ are replicated across the network.

3 Need for Non-stationary Transductions

The basic ingredients of the framework briefly described in the previous section can be summarized by the following pseudo-equation:

$$\text{Encoding\_Network}(i) = \text{Recursive\_Network} + \text{Input\_Graph}(i)$$

This framework assumes stationarity as long as the Recursive Network is stationary. In the following, we argue that in several application cases it is useful to assume non-stationarity. Non-stationary solutions are also sometimes needed because of the recursive nature of the model.

Here we briefly discuss two practical cases where non-stationarity of the model will clearly improve the efficiency and performance of the model.

3.1 A Chemical Application

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 Figure 5).

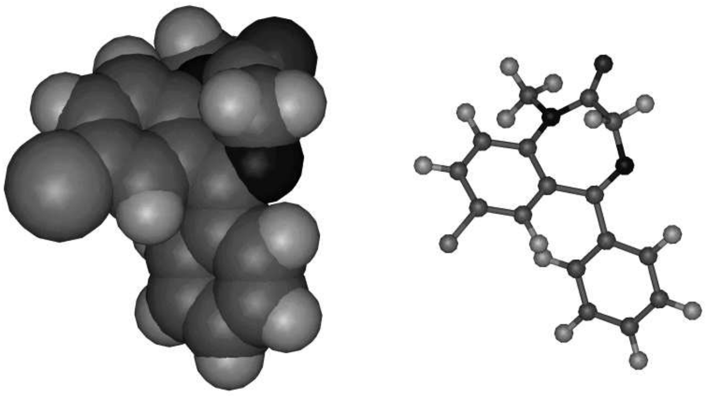


Fig. 5. Typical chemical compound, naturally represented by an undirected graph.

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 compounds' 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.

The main requirement for the use of recursive networks consists in finding a representation of molecular structures in terms of DOAGs. The candidate representation should retain the detailed information about the structure of the compound, atom types, bond multiplicity, chemical functionalities, and finally a good similarity with the representations usually adopted in chemistry. The main representational problems are: how to represent cycles, how to give a direction to edges, how to define a total order over the edges.

Let us consider a specific problem: the prediction of the non-specific activity (affinity) towards the Benzodiazepine/GABA$_A$ receptor [1]. An appropriate description of the molecular structures of benzodiazepines can be based on a labeled tree representation. In fact, concerning the first problem, since cycles mainly constitute the common shared template of the benzodiazepines compounds, it is reasonable to represent a cycle (or a set of connected cycles) as a single node where the attached label carries information about its chemical nature. The second problem can be solved by the definition of a set of rules based on the I.U.P.A.C. nomenclature system[5]. Finally, the total order over the edges follows a set of rules mainly based on the size of the sub-compounds. An example of representation for a Benzodiazepine is shown in Figure 6.

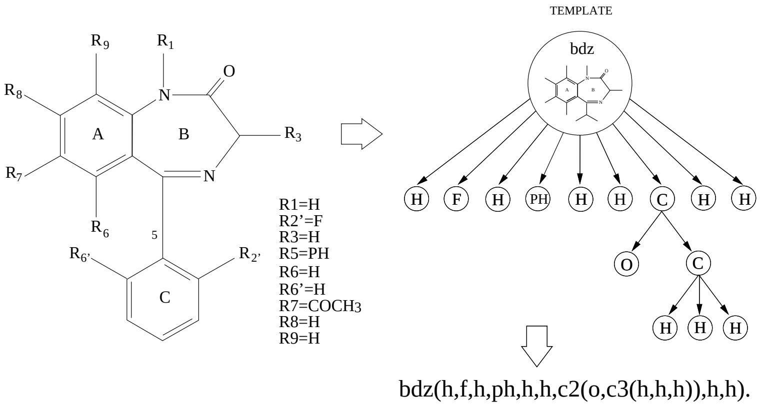


Fig. 6. Example of representation for a benzodiazepine.

This representation of Benzodiazepines implies that, once a given recursive network has been defined, the set of parameters corresponding to the pointers are used with different roles: at the level of the root, they are used to implement the target regression, while at the remaining levels they are used to perform the encoding of the substituents (which do not have a target value associated). Since there is no strong evidence about the fact that these two functionality should be implemented by the same set of parameters, it is clear that it would have been better to have a different set of parameters for each functionality. Moreover, in this particular case, since the substituents just exploit 3 out of 9 pointers, it turns out that only these 3 pointers are "overloaded", thus introducing a clear arbitrary bias in the model.

3.2 Logo 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 [4, 8, 5, 12], that are based on the premise that the *structure* of an entity is very important for both classification and description.

[5] The root of a tree representing a benzodiazepine is determined by the common template.

Figure 7 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.

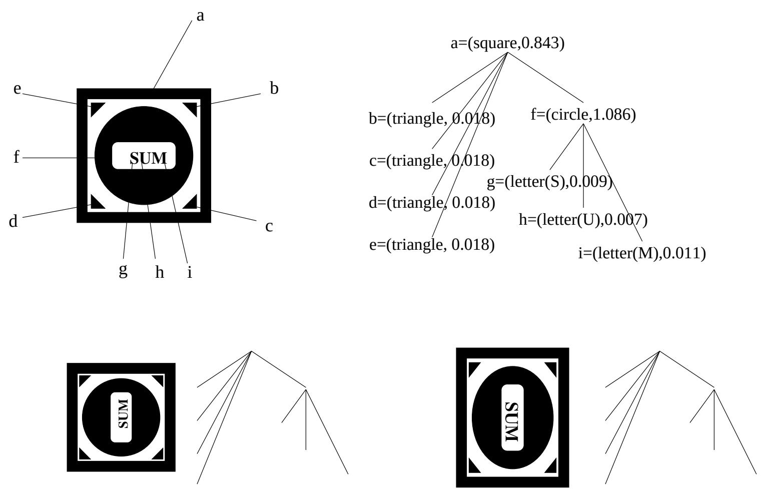


Fig. 7. A logo with the corresponding representation based on both symbolic and sub-symbolic information.

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 affect significantly 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 sub-symbolic information. For example, the logo shown in Figure 7 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 word as sub-symbolic information collected in a single node.

In general, however, the symbolic information could be exploited as a selector for different sets of parameters: tree nodes with the same symbolic part can be processed by using the same set of parameters, while tree nodes with different symbolic part can be processed by a different set of parameters. This would allow the reduction of the variance in input, thus permitting the learning algorithm to focus only on relevant aspects of the problem. Some work in this direction has already been presented in [2].

4 Representing Non-stationary Transductions

In the previous section, we have argued about the advantages of "using" non-stationary transductions. Non-stationary transductions can be defined within the proposed formalism by implementing $f(\cdot)$ and $g(\cdot)$ through non-stationary models, e.g., non-stationary neural networks. Very often, however, a priori knowledge about this non-stationarity could conveniently be represented in an explicit way at the level of the graphical model, i.e., at the level of the recursive network, independently of the implementation mechanism. The basic idea is to define a sort of graphical language for describing explicitly the desired non-stationarity. The advantage of this approach is to avoid the introduction of distinct parameters for each vertex in the input structures. In fact, the graphical formalisms allows the user to exactly specify under which specific conditions a new set of parameters must be used.

For example, let us consider the case of a transduction for binary trees where both $f(\cdot)$ and $g(\cdot)$ depend on: *i)* the distance of the node in the tree from the frontier; *ii)* the value associated to a numerical variable ($\boldsymbol{U}$) within the labels. A specific instance of this situation could be described by the pseudo-code reported in Figure 8.

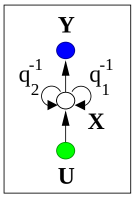
The first statement declares a variable of type sequence, while the second statement declares a variable of type vertex. The variable of type sequence is then used to contain the sequence of vertices returned by the statement `sort_vertices_by(dist_from(frontier),<)`. This statement returns one of the admissible topological orders (there may be many) where the vertices of the input DOAG are sorted by increasing ($<$) distance from the frontier. The next `foreach (v,seq){ ... }` command iterates the value of `v` through each element of the sequence `Seq`, executing the commands contained in the body { ... } after each assignment.

Within the `foreach`, there is an `if-then-else` statement which tests how far the current vertex is from the frontier. If the distance of the current vertex from the frontier is less than 3, then an additional test is performed on the associated numerical variable $\boldsymbol{U}$: if the value of $\boldsymbol{U}$ is within the range $[0.3, 0.55]$, then the recursive network corresponding to the second occurring `then` branch is used, otherwise the recursive network corresponding to the first occurring `else` branch is used. If the distance of the current vertex from the frontier is not less than 3, the recursive network corresponding to the second occurring `else` branch is used.

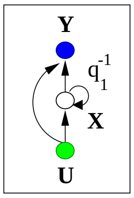
Please note that the three recursive networks which appear in the "program" should be intended as distinct models, i.e., each network has a different set of parameters associated with its state transition and output functions.

All the three recursive networks involved in the definition of the above description exploit the same set of variables (i.e., $\boldsymbol{U}$, $\boldsymbol{X}$, $\boldsymbol{Y}$), which assures the consistency of the description. In Figure 9, an example of application of the "program" is reported. In the right side of the figure, the encoding network obtained by applying the program to the input tree in the left-side of the figure is shown. Note that vertices 1, 2, 4, 5, and 6 satisfy both the first and second `if-then-else` conditions, while vertices 3 and 7 only satisfy the first `if-then-else` condition. Finally, vertex 8 is the only node for which the last recursive network is applied.

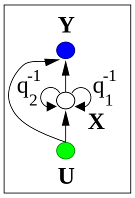
```
Sequence_of_vertices Seq;
Vertex v;
Seq <- sort_vertices_by( dist_from(frontier), <);
foreach(v, Seq) {
  if (dist_from(frontier)<3) then {
      if ( U  in [0.3,0.55] ) then
```




```
;
      else
```




```
;
  }
  else
```




```
;
}
```

Fig. 8. Example of non-stationary recursive network.

It must be observed that the resulting encoding network is not connected since the subnetwork corresponding to vertices 3 and 6 is not connected to the remaining part of the network. As a consequence of that, neither vertex 3, nor vertex 6 will contribute to the definition of the state and output of vertices 7 and 8.

The processing flow, in this case, is sequential, since the vertices of the input graph are sorted according to a given topological order. In general, however, the preorder defined by the topology of the input DOAG admits a degree of parallelism. This form of parallelism can be expressed by a data flow model, where the nodes of the input DOAG are visited in parallel in a bottom-up fashion, from the frontier to the supersource. For example, the nodes of the input DOAG of Figure 9 can be processed according to the following sequence of sets, S_1={1,2,3,4}, S_2={5,6}, S_3={7}, S_4={8}, where nodes in the same set

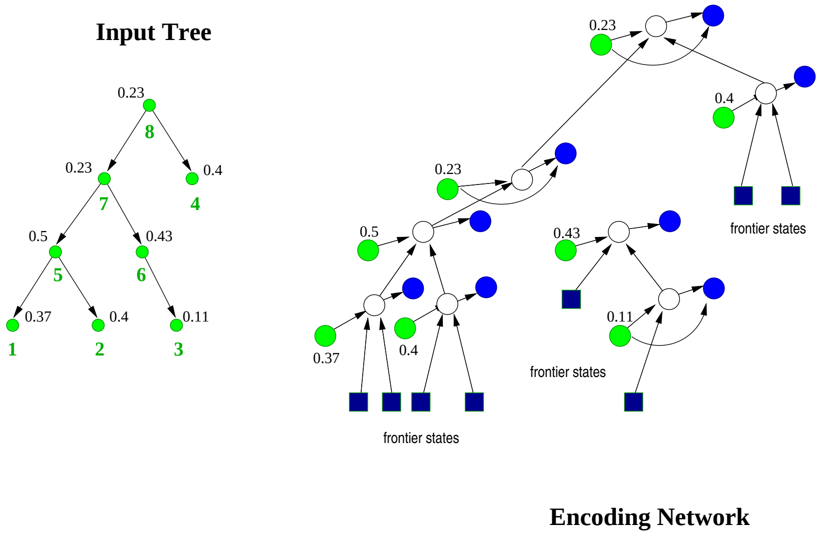


Fig. 9. Example of generation of the encoding network by using the input in the left side of the figure and the non-stationary recursive network shown in Figure 8.

are processed in parallel but set S_i must be processed strictly after set S_{i-1}.

Such parallelism could be expressed by adopting an event-based environment where an event can be understood as a condition satisfied by a given vertex of the input DOAG. This event-based environment could be introduced by the statement `deal_with` { ... } which in general contains a set of (non contradictory) `event(condition):` {...} statements and a `default:` {...} statement applied when no condition of the `event` statements is satisfied.

It must be observed that a `condition`, besides to be a hard logical condition on the topology of the DOAG, i.e., whether a node is at a given level of the DOAG or the node has more than 2 children, can also be defined as a fuzzy condition, implemented, for example, by a neural network. So the test on the condition can either activate a symbolic procedure or a pattern recognition subsystem.

One interesting aspect of this approach is that, since the "program", together with the input DOAG, is used to define the encoding network, learning is driven by the "program" as well. In fact, the correspondence between parameters and connections within the encoding network is established by the "program". In general, it may be feasible to define statements concerning the learning modality for each recursive network involved in the "program", even considering different (but "compatible") learning algorithms for each recursive network, i.e., different gradient descent algorithms.

In Figure 10, an example of a simple `if-then-else` graphical rule which solves our chemical problem, as stated in Section 3.1, is shown. If the current vertex is the root of the input DOAG, then all the subgraphs (substituents) are considered and an output (predicted biological activity) is generated, otherwise only three set of parameters are used for the encoding of the subgraphs (chemical

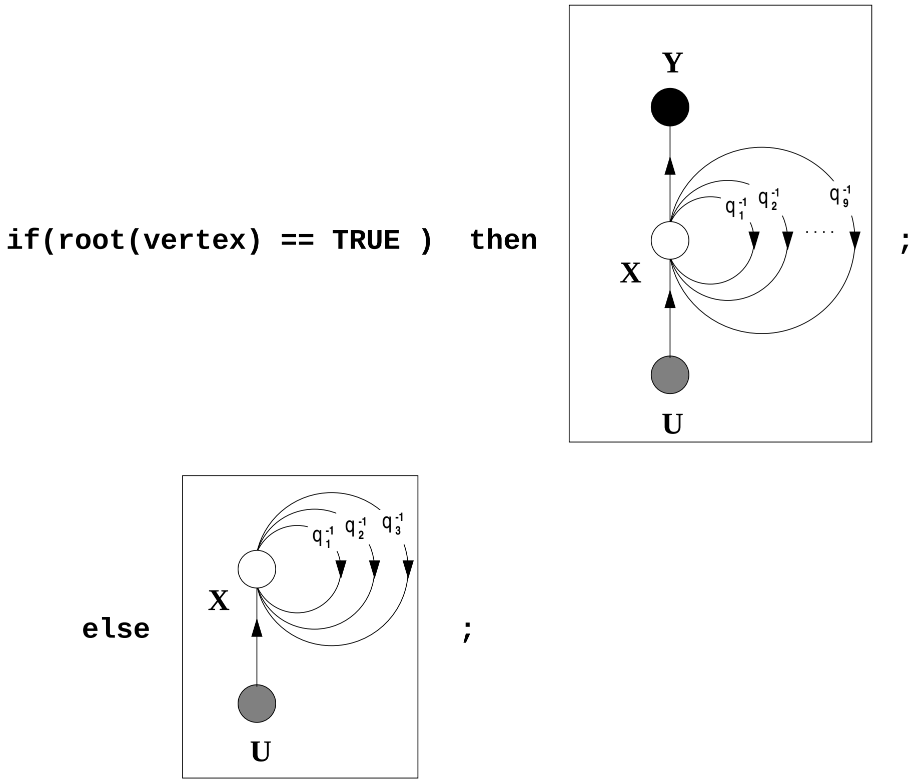


Fig. 10. Example of non-stationary rule for the chemical application.

fragments) and no output (prediction) needs to be generated.

5 Conclusion

We have briefly reviewed the basic concepts underpinning the adaptive processing of data structures as outlined in [3]. Concepts such as causality and stationarity have been adapted to the context of learning data structures. In particular, we have argued that in practical applications stationarity in not always desiderable. In order to explain the role of non-stationarity, we have discussed two practical examples, i.e., predicting the biological activity of chemical compounds and automatic classification of logos. Finally, we have introduced very briefly our idea on how a priori knowledge on the domain can be expressed in a graphical form, allowing the formal specification of eventually very complex (i.e., non-stationary) requirements for the structured domain to be treated by a neural network or a Bayesian approach. The advantage of the proposed approach is the systematicity in the specification of both the topology and learning propagation of the adopted computational model (i.e., either neural or probabilistic, or even hybrid by combining both of them). Moreover, the proposed approach would allow the easy definition and reuse of basic modules, besides to the possibility to modify the computational model with a few changes in the formal specification.

References

1. A.M. Bianucci, A. Micheli, A. Sperduti, and A. Starita. Quantitative structure-activity relationships of benzodiazepines by recursive cascade correlation. In *IEEE International Joint Conference on Neural Networks*, pages 117–122, 1998.
2. M. Diligenti, M. Gori, M. Maggini, and E.Martinelli. Adaptive graphical pattern recognition: the joint role of structure and learning. In *Proceedings of the International Conference on Advances in Pattern Recognition*, pages 425–432. Springer, 1998.
3. P. Frasconi, M. Gori, and A. Sperduti. A framework for adaptive data structures processing. *IEEE Transactions on Neural Networks*, 9(5):768–786, 1998.
4. K. S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs, N.J, 1982.
5. R. C. Gonzalez and M. G. Thomason. *Syntactic Pattern Recognition.* Addison Wesley, Reading, Massachusettes, 1978.
6. G. E. Hinton. Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46:47–75, 1990.
7. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
8. T. Pavlidis. *Structural Pattern Recognition.* Springer-Verlag, Heidelberg, 1977.
9. Tony A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, May 1995.
10. J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–106, 1990.
11. R. Rosenfeld and D. S. Touretzky. Four capacity models for coarse-coded symbol memories. Technical Report CMU-CS-87-182, Carnegie Mellon, 1987.
12. R. J. Schalhoff. *Pattern Recognition: Statistical, Structural and Neural Approaches.* John Wiley & Sons, 1992.
13. P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216, 1990.
14. A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.
15. D. S. Touretzky. Boltzcons: Dynamic symbol structures in a connectionist network. *Artificial Intellicence*, 46:5–46, 1990.