Neural Network for Graphs: A Contextual Constructive Approach

Alessio Micheli, *Member, IEEE*

***Abstract*—This paper presents a new approach for learning in structured domains (SDs) using a constructive neural network for graphs (NN4G). The new model allows the extension of the input domain for supervised neural networks to a general class of graphs including both acyclic/cyclic, directed/undirected labeled graphs. In particular, the model can realize adaptive contextual transductions, learning the mapping from graphs for both classification and regression tasks. In contrast to previous neural networks for structures that had a recursive dynamics, NN4G is based on a constructive feedforward architecture with state variables that uses neurons with no feedback connections. The neurons are applied to the input graphs by a general traversal process that relaxes the constraints of previous approaches derived by the causality assumption over hierarchical input data. Moreover, the incremental approach eliminates the need to introduce cyclic dependencies in the definition of the system state variables. In the traversal process, the NN4G units exploit (local) contextual information of the graphs vertices. In spite of the simplicity of the approach, we show that, through the compositionality of the contextual information developed by the learning, the model can deal with contextual information that is incrementally extended according to the graphs topology. The effectiveness and the generality of the new approach are investigated by analyzing its theoretical properties and providing experimental results.**

***Index Terms*—Cascade correlation, contextual transductions, graph patterns, learning in structured domains (SDs), neural networks for structured data, recursive neural networks.**

I. Introduction

THE study of learning in structured domains (SDs) allows for a generalization of machine learning (ML) approaches to the treatment of complex data, offering both new impulses for theory and applications. In particular, extending the methodologies in order to deal with classes of data structures can be recognized as one of the most promising ways of increasing the possibility of successful new applications. Indeed, varying-size structures, e.g., sequences, trees, and graphs occur in many fields such as natural language processing (NLP), document and structured text analysis, image processing, object recognition, cheminformatics, and bioinformatics. Learning in SD raises theoretical issues considering first the need of new models for nonvectorial data, which must be able to directly represent relational data of variable size in their hypothesis space, up to the generalization of the ML basis induced by the extension of the data domains. Indeed, various methodologies approach the topic starting from different paradigms in the machine learning, statistical relational learning (see, e.g., [1] and [2]), and relational data mining area (see, e.g., [3]), with the aim of extending the class of input data that are traditionally restricted to a flat-vector form. Often the terms "structured domain learning" (SDL) or "relational learning" are used to refer the topic.

In this paper, we focus on subsymbolic supervised learning (classification/regression) for discrete data structures. We are recently witnessing an increasing attention in this area to the development of various kernel-based approaches for structured data (e.g., see [4]–[8] for overviews). Classical kernel approaches rely on the definition of a proper similarity measure on structures (expressed by the kernel function). An open issue is the problem of the design of a similarity measure on structures effective for specific tasks and/or of general validity, which poses practical and theoretical limitations [9], [10]. The point at issue is the concept of adaptivity, i.e., the capability that allows the model in principle to learn from the training data the similarity measure on structures for the task at hand. In this respect, neural network approaches based on adaptive processing of the structured input data constitute a powerful methodology in the area of subsymbolic approaches to SDL. In particular, advanced models such as recurrent neural networks for sequence domains and recursive neural networks (RNNs) for SDs have been introduced [11]–[13]. In the family of RNNs, constructive approaches have been introduced for sequences and structures as well, such are recurrent cascade correlation [14] and recursive cascade correlation (RCC) [12], [15], [16]. RNN models realize an adaptive processing (encoding) of recursive (hierarchical) data structures. A recursive traversal algorithm is used to process (encode) all the graph vertices, producing state variable values for each visited vertex.

The RNN approach is computationally characterized by the *causality* of the transition system [11]. The causality assumption imposes on a directed acyclic graph that the computation for a vertex depends only on the current vertex and vertices descending from it (see Section II-B). Such assumption is used by RNN (and RCC) to realize an encoding that matches the input hierarchical structures according to their topological order. However, such encoding process poses constraints on the transduction realized by the RNN models (see [17]) and on the classes of data. Indeed, sequences, rooted trees, or directed (positional) acyclic graphs (DAG[1] and DPAG) are subclasses of graphs for which a topological sort exists.[2]

Manuscript received May 02, 2007; revised January 19, 2008 and July 29, 2008; accepted August 18, 2008. First published February 03, 2009; current version published February 27, 2009.

The author is with the Dipartimento di Informatica, Università di Pisa, 56127 Pisa, Italy (e-mail: micheli@di.unipi.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2008.2010350

[1] The treatment of DAG by RNN is introduced in [18].

[2] A topological sort of a DAG is a linear ordering/numbering of all its vertices such that every directed edge from a vertex numbered i to a vertex numbered j satisfies $i < j$.

Various approaches are emerging, with different bases, to overcome the causality constraints and to treat more extended classes of graphs by transition systems. The contextual recursive cascade correlation (CRCC) [17] provides a contextual approach to deal with DPAG, showing the possibility to partially relax the causality assumption in a recursive model. Although the universality approximation results on classes of DPAG assure the power of the CRCC [19], the approach leaves open the problem to directly deal with general classes of graphs including cyclic and/or undirected graphs by neural networks.

In particular, processing cyclic graphs is another relevant issue. For RNN, extensions to cyclic graphs are studied both by introducing cycles in the state transition system or by rewriting the original graphs into hierarchical structures, e.g., [12] and [20], and more recently by a new recursive approach for graphs [21] and [22]. While in such approaches cyclic dependencies among states are allowed and treated by relaxation techniques and/or imposing constraints on the dynamical system, in this paper, we investigate, through the introduction of a new approach, the possibility to process general graphs relaxing the need of complex dynamical/recursive systems.

Another problem to consider is the difficulty in the learning for the recurrent/recursive neural models. Indeed, the design of fast and effective training algorithms for RNN, due to its complex dynamical properties, still constitutes an ongoing issue of research (e.g., [23] and, for very recent results, [24]–[26]). Moreover, the study of efficient solutions, leading to (neural networks) models able to deal with general graphs, including either directed/undirected, cyclic/acyclic, or ordered/unordered graphs still deserves further research. Hence, there is need to investigate simple approaches, which however retain the capability to directly treat (or to represent in its hypothesis space) large classes of structured data of variable size.

In this paper, we present a new supervised approach based on these assumptions, called *neural network for graphs* (NN4G), relaxing the well-known power of recursive approaches in favor both of a simpler architectural and learning design, and of the contextual processing of general classes of graphs (including both acyclic/cyclic and directed/undirected labeled graphs). The simplicity of the model stems from the use of neurons that have no recursive/feedback connections. Hence, in comparison with previous causal neural networks for structures, which are recursive models, NN4G possesses a feedforward architecture; it evolves (generalizing to graphs) the concepts of structures traversal and of contextual processing previously emerged in the area of causal (recursive) neural network approaches, while relaxing the constraints due to causality for general graphs. Moreover, NN4G exploits a constructive approach, and the contextual information of each vertex is incrementally used by the model without introducing cyclic dependencies in the definition of the system state variables. On the other hand, in contrast to standard feedforward architectures, the model generalizes from flat vectors to graphs the input domain by the means of a state transition system and of an incremental contextual processing.

The aim of this paper is to introduce the new neural network contextual approach to SDL and to analyze the effects of relaxing the causality assumption on the class of input data and on the shape of contextual information used by the model. A preliminary version of this paper was presented in [27] and [28]. Here we present an extended version including the generalization of the model and the first theoretical analysis of the model properties.

The rest of the paper is organized as follows. Sections II-A and II-B introduce the preliminaries on the domains of data and on contextual transductions over SDs. The NN4G architecture (together with the concepts of state variables, graph traversal, and stationarity) is then introduced in Section II-C and its learning algorithm in Section II-E. In Section II-D, we introduce the analysis of the flow of the information used by the NN4G allowing us to show an instance of the compositional development of NN4G context over graphs. Note that this property is not trivially observable in the model equations while it is fundamental to characterize the new model for the realization of contextual transductions on graphs. We also show how these concepts allow the model to extend the context considered by recurrent and recursive approaches. Such analysis is formalized and completed in Section III by introducing the general form assumed by the context of NN4G state variables. Simulation results are presented in Section IV. Discussion and conclusions are contained in Sections V and VI, respectively.

II. Neural Network for Graphs (NN4G)

A. Structured Domain and Preliminaries on Graphs

Here we assume a class of input structured patterns as labeled graphs. Given a set of labeled graphs $\mathcal{G}$ and a graph $g \in \mathcal{G}$, we denote the set of the vertices of g as $\mathrm{Vert}(g)$ and the set of its edges as $\mathrm{Edg}(g)$. Each vertex has a label representing a multidimensional numerical attribute $\boldsymbol{l}(v)$ and $l_i(v)$ is the ith of L components. An edge is a connection between vertices (u and v) written as a pair (u, v). Given a vertex $v \in \mathrm{Vert}(g)$, we define $\mathrm{edg}(v)$ to be the set of the edges incident on v, whose cardinality is $\mathrm{degree}(v)$; it is assumed a maximum finite degree for the graphs in $\mathcal{G}$. The graph can be *directed* or *undirected* if the edge set consists of ordered or unordered pairs of vertices, respectively. In an *ordered* graph, for each vertex v, a total order on the edges incident on v is defined. Besides ordering, in a *positional* graph, a distinctive positive integer can be associated to each edge incident on v (where some positions can be absent). The *distance* between two vertices in a graph is the number of edges in a shortest path connecting them. In a undirected graph, we denote the set of adjacent vertices, or *neighbors*, of the vertex v as $\mathcal{N}(v)$, considering both the *open* form which does not include v itself and the *closed* form when v is also included. For an open neighborhood, it holds that

$$\mathcal{N}(v) = \{u \in \mathrm{Vert}(g) | (u, v) \in \mathrm{edg}(v)\}. \quad (1)$$

Denoting the cardinality of the set $\mathcal{S}$ by $|\mathcal{S}|$, for open neighborhoods, we have $\mathrm{degree}(v) = |\mathcal{N}(v)|$. In a directed graph g, the set of *neighbors* $\mathcal{N}(v)$ of v is given by the set of adjacent vertices obtained in the undirected version of g, with the possibility to distinguish the neighbors on the basis of entering edges (u, v) or leaving edges (v, u), defining the set of predecessors, or in-neighborhood ($\mathcal{P}(v) = \{u | (u, v) \in \mathrm{edg}(v)\}$) and

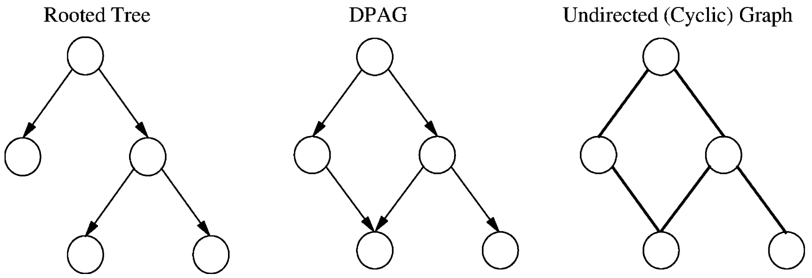

Fig. 1. Examples of structures: a rooted tree, a directed positional acyclic graph, and an undirected (cyclic) graph.

the set of successors, or out-neighborhood ($\mathcal{S}(v) = \{u | (v, u) \in \mathrm{edg}(v)\}$), respectively.

A subclass of structures is the class of *free trees*, which are connected acyclic undirected graphs. *Rooted trees* are free trees with a distinctive (supersource) vertex called the *root*. In a positional rooted tree (as a special case of directed positional graphs), the position is assigned to each edge leaving a vertex. Positional trees with bounded out-degree K are called K*-ary trees*. A "1-*ary*" tree is a path, i.e., the domain is restricted to *sequences*. The restricted subclasses of K*-ary* trees, directed acyclic graphs (DAG), directed ordered acyclic graphs (DOAG), and directed positional acyclic graphs (DPAG) are considered here because they play a major role in previous recursive neural network approaches to SD processing. In Fig. 1, examples of various classes of structures, i.e., a tree, a DPAG, and an undirected (cyclic) graph, are reported.

Note that in the context of recursive approaches the treatment of undirected graphs and the treatment of cycles are related: each pair of vertices connected by an undirected edges can be interpreted in terms of a directed graph with a cycle involving the two mutually connected vertices.

B. Contextual Transductions

In the following, we consider *transductions* over SDs (see, e.g., [11], [13], and [17]). A transduction is a mapping from an SD $\mathcal{G}$ (a set of input graphs) to an output domain. In a supervised transduction, the output domain, which is used to represent the targets of the relational learning task, can be either a set of scalar values, for classification or regression tasks (e.g., the transduction is a mapping $\mathcal{G} \rightarrow \Re$), or a structured data. The transductions considered in the following entail an encoding of the input structures into an inner representation space, composed by state variables associated with each vertex of the input graph, and an output mapping function that returns the predicted values.

Contextual processing can be referred to computations that yield for each vertex a response that depends on the whole information represented in the structured data [16]. To realize a contextual transduction, we assume that the encoding (state) for a vertex v of the input graph is a function of the information in the context of v according to the topological relationships among vertices represented by the edges of the graph.

Specific kinds of context are considered in causal neural networks. The context of recurrent neural networks summarizes the set of past inputs of the (temporal) input sequence. The context of recursive neural networks applied to directed structures summarizes information about the descending vertices of the current processed input vertex.

At the same time, the causality assumed in such models poses constraints on the traversing of the input structures. In particular, in a DAG, a *causal* transduction assumes that the computation for a state variable of a vertex depends only on the current vertex and vertices descending from it. Accordingly, the model traverses sequences from one end of the sequences to the other, and directed structures such as trees from the leaves to the root (following an inverse topological order). Hence, the state variables associated with each vertex depend either on a single-side context of a sequence or on the context of the children of trees. This assumption can be natural for special cases, such as modeling temporal dynamics, i.e., for transductions on time series, where the system response only depends on past events. However, the causality assumption can be an unsuitable constraint for tasks requiring contextual information, e.g., considering both possible directions in sequences or considering the neighborhood of each vertex in a structure (instead of only the descending vertices). For instance, among the cases occurring in language analysis, text analysis, and bioinformatics, we mention tasks and previous approaches developed for sequence domains whereas the task is the prediction of the classes of secondary structure of proteins from their sequence of amino acids [29]–[31]. More general solutions, relaxing the constraints on the traversal process, are needed for general classes of graphs. In particular, less restrictive dependencies for state variables extended to a symmetric context of each vertex can be considered. This is the case of the NN4G model that we introduce in the following.

C. A New Contextual Model

Following the line of contextual processing initiated in [16] and [17] for K*-ary* trees and DPAGs, it is possible to introduce a simple approach that extends the domain to general graphs by means of an incremental factorization of the state variables allowing to progressively access the state of contextual vertices. To this aim, here we introduce a constructive network, the NN4G. In a constructive approach, the learning algorithm progressively adds one new state variable (or hidden unit) at a time to a set of already trained (*frozen*) units. A concrete instance of a constructive approach of this type was introduced for flat-vector inputs by the family of cascade correlation models [32], [33]. For graph processing, where a state variable is associated to each vertex of the input graph, the basic idea is to exploit the content of frozen units according to the graph topology. In particular, the state values computed by the frozen units are available for all the vertices of the input graph when new units are introduced. Hence, the frozen state values of the neighbors of each vertex can be used as an input for the new units without introducing cycles in the dynamics of the state computing system.

We start presenting the hidden units component of NN4G, first by an abstract formulation with a parametrization of graph components in a general form (i.e., without fixing any specific stationarity assumption), and then proposing a specific realization that fits the case of data in the class of unordered undirected graphs.

1) Hidden Units (State Variables): Denote by $x_i(v)$, where $v \in \mathrm{Vert}(g)$, the state variable computed by the ith hidden neural unit of the NN4G for the vertex v. In the NN4G, as an initial condition, the first unit $(i = 1)$, which has no previous layers, computes its output only on the basis of the label of each vertex, i.e.,

$$x_1(v) = f\left(\sum_{j=0}^{L} \bar{w}_{1j} l_j(v)\right) \tag{2}$$

where f is a linear or sigmoidal function, $\bar{w}_{ij}$ is the weight of the jth entry of the label to the ith hidden unit $(i = 1)$, $l_j(v)$ is the jth component of the vertex label (Section II-A), and the bias is included in the model by setting $l_0(v) \doteq 1$. The functional dependencies of the (hidden neuron) state variables for all the other units, i.e., $x_i(v)$ for $i \geq 2$, can be expressed in a general and abstract formulation of NN4G as

$$x_i(v) = f\left(\sum_{j=0}^{L} \bar{w}_{ij} l_j(v) + \sum_{j=1}^{i-1} \sum_{u \in \mathcal{N}(v)} \hat{w}_{ij}^{(v,u)} x_j(u)\right), \quad i = 2, \ldots, N \tag{3}$$

where $\hat{w}_{ij}^{(v,u)}$ is the weight of the connection between the current unit i and the jth preceding frozen hidden unit associated with the edge (v, u), which brings the state value of vertices u in the neighborhood of v; see (1). The abstract (hidden unit) system in (3) is applied to each vertex of each input graph through a graph traversal algorithm (see Section II-C3), producing a state value for each visited vertex. Each unit i visits all the vertices of the input graphs set. The problem of variance in size and of the topology of the input graph is faced by the two following concepts:

- local unit configuration: the shape of the input configuration of NN4G units is composed according to the local vertex connections [the neighborhood of v in (3)];
- stationarity assumption: the abstract system in (3) is uniformly applied to all the vertices of all the graphs in the input data set, i.e., it is made independent of the vertex v, according to an appropriate weight-sharing technique among parameters used for different vertices or within other subsets of graph components [edge connections in (3)]. In order to realize the abstract framework in (3), a specific stationarity form must be assumed determining the relationship between parameters (weights) and types of edges, with both belonging to a finite set. The form of stationarity controls which weights are shared (i.e., which edges are treated equally). Then, this model is uniformly applied/replicated for all the vertices. As an instance, in the following, we introduce a specific realization tailored for general unordered undirected graphs with no other positional or labeling assumption for edges.

Hence, given a maximum degree for vertices and a finite set of edges types, it is possible to treat different vertices by a unit model with a number of free parameters that is independent of the size of the structures, in particular, independent of the number of vertices.

Note that (v, u) is intended to be an unordered pair. The formulation can be specialized for oriented edges by distinguishing the pairs on the basis of entering and leaving edges. Both the case of open and closed neighborhood can be considered in the concrete realizations of the model.

The NN4G state value of a vertex is computed as a nonlinear function of the vertex label and of the state values indexed by $j < i$ of the adjacent vertices. In particular, in (3), the first sum corresponds to the label information and the second sum corresponds to the (local) "contextual" information taken from each preceding hidden unit, i.e., the state values computed for the neighbors of v.

Fig. 2 shows two steps of the hidden unit traversal process: it shows the application of a general hidden unit i to two different vertices of an input graph (assuming that the state values $x_j(\cdot)$ for $j < i$ and $i > 1$ were computed for all the vertex of the input graph). Note that the values of $x_j(\cdot)$ state variables computed for each vertex of the input graph are graphically shown by a graph isomorphic to the original input graph in the two examples in the lower part of the Fig. 2. The unit i uses these values to compute the new value $x_i(\cdot)$. The same schema is applied to all the other vertices of the input graph, according to the graph traversing process, and to all the graphs of the input data set. The state values computed for each vertex will be used by the output layer to compute the predictive values of the model (Section II-C2).

This computation is *not recursive* because $x_i(\cdot)$ does not depend on itself but it depends on the output $x_j(\cdot)$, $j < i$ and $i > 1$, of the other hidden units that are already frozen. Moreover, no cyclic dependencies among state variables are introduced in the state transition system, even for cyclic graphs (which instead induce cyclic configurations for the unfolding of recursive approaches). This holds because, by construction, the computation of $x_i(v)$ does not depend on the state values computed by the unit i for the other vertices, and therefore, no cyclic circuits, or mutual functional dependences among states, can be established in the information flow defined in (3) independently of the topology of the input graph. In particular, note that differently from an unfolding process of a recurrent/recursive network, due to the absence of recursive connections, there are no connections among unit replica (or state variables $x_j(v)$ values) at the same hidden level for an NN4G. As shown in Fig. 2, only the connections among different state variables (with different indexes) are made according to the topology of the input structure.

Finally, in NN4G, each computation is local (vertex and its neighbors) and there is no constraint to follow any topological order on the vertices to compute the state variables. Hence, although the concept of stationarity is inherited from the RNN approaches [11], the concept of recursive unfolding of the RNN

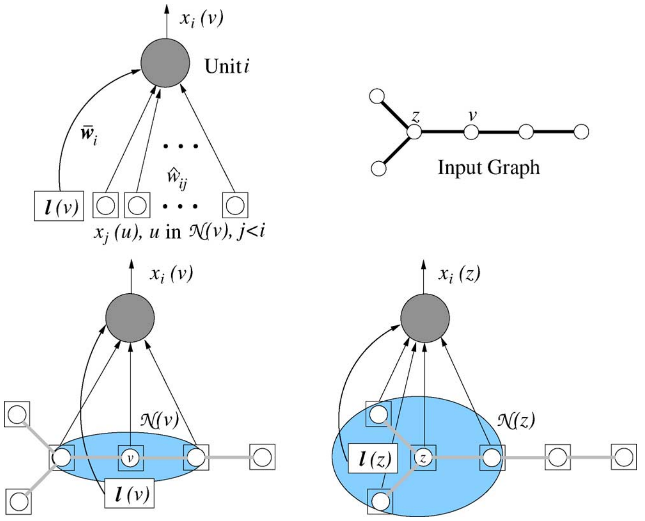

Fig. 2. Generic NN4G (hidden) unit, an example of input graph, and the application (visit) of the unit to vertices v and z for the computation of $x_i(v)$ and $x_i(z)$.

approaches is substituted by a general traversing of the input graph that is not constrained to any specific topological order. As a result, the class of graphs that can be treated can be quite general including cyclic/acyclic and directed/undirected graphs.

In Sections II-D and III, we show how the NN4G includes contextual information beyond the local neighborhood extending the context to the other vertices. This aspect is fundamental to distinguish the model from algebraic (unstructured) "sliding window" approaches, which are restricted to only local processing, and from previous recursive approaches.

The framework in (3) is still an abstract formulation. To realize concrete models, the association of the free parameters (weights) of the model with the components of the structure (edges in particular) can be made according to a specific weight-sharing (or *stationarity*) assumption. Various possibilities can be modeled according to the information associated to the edges, considering either the position, the orientation, or other labels of the edges (given that they belong to a finite set). Hence, specific classes of graphs, such are directed and positional graphs (e.g., K-*ary* trees, DOAGs, and DPAGs) can be considered in this framework as well.

In particular, we focus on the further simplification of this abstract system based on a full stationarity assumption where a single parameter $\hat{w}_{ij}$ is associated with each incident edge (which are therefore assumed as all components of the same type). This allows both the treatment of unordered (and/or non-positional) undirected graphs and a strong parameters reduction. In this instance of NN4G, the state variables are defined as

$$\begin{cases} x_1(v) = f\left(\sum\limits_{j=0}^{L} \bar{w}_{1j} l_j(v)\right) \\ x_i(v) = f\left(\sum\limits_{j=0}^{L} \bar{w}_{ij} l_j(v) + \sum\limits_{j=1}^{i-1} \hat{w}_{ij} \sum\limits_{u \in \mathcal{N}(v)} x_j(u)\right), \\ \qquad\qquad\qquad\qquad\qquad\qquad i = 2, \ldots, N \end{cases} \tag{4}$$

where $\hat{w}_{ij}$ is the weight associated with the connection between the current unit i and the jth preceding frozen hidden unit, which brings state values of all the adjacent vertices to v and v itself (i.e., the closed neighborhood was used in the experimental setting). This is a challenging simplified version of the NN4G with efficiency benefits, whose efficacy deserves to be experimentally evaluated (Section IV).

Before presenting the global NN4G transduction algorithm, let us complete the architecture of the model.

2) Output Layer: The systems specified in (3) or (4) can emit a value for each vertex of the given graph g. These values can be used to implement an input–output (IO)-isomorphic transduction [11] with a structured output denoted by $\boldsymbol{x}(g)$ [$\boldsymbol{x}_i(g)$ will denote the output graph for a single $x_i(\cdot)$]. Alternatively, a scalar value for a whole graph can be emitted, i.e., the input pattern is a graph and the response will be calculated in the form of a single scalar value associated with the whole graph. In the latter case, the state values (whose number depends on number of vertices of the input graph) can be collected by an average operator applied to $\boldsymbol{x}_i(g)$ for each unit i. In particular, in this instance of NN4G, we calculate a value denoted by $X_i(g)$ according to the following definition:

$$X_i(g) = \frac{1}{k} \sum_{v \in \mathrm{Vert}(g)} x_i(v) \tag{5}$$

where we can fix k either as $k = 1$ to define a sum, or $k = |\mathrm{Vert}(g)|$ to define an arithmetic average, or $k = \max_{h \in \mathcal{G}}(|\mathrm{Vert}(h)|)$ to define a sum normalized in the interval $[-1, 1]$ for sigmoidal activation functions.

The output y_o $(o = 1 \ldots M)$ of the NN4G network is given by M output units that are computed distinguishing two cases, i.e., vertex patterns (IO-isomorphic transductions with a target value associated with each vertex) or graph patterns (the target value is associated with each input graph)

$$y_o(p) = \begin{cases} f\left(\sum\limits_{i=0}^{N} w_{oj} X_i(p)\right), & \text{if } p \in \mathcal{G} \\ f\left(\sum\limits_{i=0}^{N} w_{oj} x_i(p)\right), & \text{if } p \in \mathrm{Vert}(g), \qquad g \in \mathcal{G} \end{cases} \tag{6}$$

where N is the number of hidden units, and f can be a linear or a sigmoidal function according to regression or classification task, respectively.

3) Transduction and Global Algorithm: Summarizing, the transduction performed by the NN4G for each input graph $g \in \mathcal{G}$ when patterns are graphs g and a single real value is produced in an output for each input graph, can be written as in the following succession of mappings:

$$g \rightarrow \boldsymbol{x}(g) \rightarrow \boldsymbol{X}(g) \rightarrow y(g) \tag{7}$$

where $\boldsymbol{x}(g)$ is a graph isomorphic to the input graph g annotated for each vertex with the state values computed by (4), $\boldsymbol{X}(g)$ is a vector computed for each element (unit) by (5), and $y(g)$ is computed by (6). Note that the $g \rightarrow \boldsymbol{x}(g)$ mapping in (7) is the *encoding* function and the rest composes the *output* mapping function of the whole transduction.

We summarize in Fig. 3 the algorithm used to compute the output of the NN4G (with one output unit and N hidden units) when each pattern is a graph $g \in \mathcal{G}$. The computation of $\boldsymbol{x}(g)$ follows a very general traversing algorithm where, for each unit

```
- For $i = 1$ to $N$
-   For each $g$ in $\mathcal{G}$
-     For each $v$ in $Vert(g)$
-         Compute $x_i(v) =$ Eq. (4)
-     Compute $X_i(g) =$ Eq. (5)
- For each $g$ in $\mathcal{G}$
-   $y(g) = f\left(\sum_{j=0}^{N} w_j X_j(g)\right)$
```

Fig. 3. Algorithm for the computation of the NN4G graph transduction (graph patterns).

i, the value of $x_i(v)$ is computed for every vertex v of every graph $g \in \mathcal{G}$. Note that, since no topological order is required for the vertices, the states $x_i(v)$ can be computed without imposing any order on the graph traversing and they can even be computed in parallel for each vertex of g. Moreover, the algorithm shows that if the maximum degree is bounded over the set of the input graphs, the method scales linearly with the number of vertices in the graphs.

4) Example (Graph Transduction): As an example, the "input graph" in Fig. 2 is encoded into $\boldsymbol{x}(g)$ by the application of the hidden NN4G units to all the vertices of g (where each hidden unit i computes a state value $x_i(v)$ for each visited vertex). According to (4), both topological information and vertex labels are encoded in $\boldsymbol{x}(g)$. Then, according to (7), the state values of each vertex of the input graph are collected by $\boldsymbol{X}(g)$ and mapped by the output unit to a predictive value. For a different graph of the input set, we use the same units and the process is the same as before. Supposing that the input graph represents a chemical compound, the mapping in (7) can be used to relate the molecular structure to an output value representing a property value of such molecule. The free parameters of the model are used to adapt the encoding and output mappings to the given training set (made of known compound-property samples). Globally, the NN4G allows the supervised learning of a transduction over graph domains that can be used, for instance, to learn a direct correlation between input molecular structures and property values for a given set of samples. In Section II-D, we show how the encoding part of the graph transduction can capture the global topology of each input graph.

D. NN4G Contextual Processing

In the following, we discuss the characteristics of the NN4G as a contextual learning system for graphs. We show by an example, before the formalization given in Section III, how the model can treat a contextual information that is extended from local neighbors to the other vertices.

The traversing of an input graph made by any NN4G unit (with index i) according to the algorithm in Fig. 3 defines a "virtual" layer $\boldsymbol{x}_i(g)$ with a state value for each vertex of the input graph g (*encoding* of g). According to (7), the units are also connected to the output units through the operator $\boldsymbol{X}(g)$ (*output* mapping for a graph pattern with singular scalar value response). We can graphically visualize the flow of the encoding and output process, showing such states for each vertex of the input graph and the dependencies of the state values from the output of previous frozen units according to the local graph topology [on $\boldsymbol{x}_i(g)$]. Fig. 4 shows both architectural details of NN4G and an example illustrating the concept of the context for the vertex v (in black) and its evolution in the network construction. For the same input graph shown in Fig. 2, we show the state values computed for each vertex of the input structure by three hidden units [to this aim, the input structure is replicated three times corresponding to $\boldsymbol{x}_1(g), \boldsymbol{x}_2(g), \boldsymbol{x}_3(g)$], the operators $X_i(g)$, and the neural unit used to compute the output variable y. Note that in Fig. 4 dashed lines are used to show the flow of information in the encoding mapping "virtual" layer. These flows occur within the hidden units state variable values (for the context development) or from the state values (toward the output mapping), which are computed, adding a unit at time, by the traversal process over the input graph shown in the left part. The solid lines are used to show the connections at the output (unit) layer, which computes the final mapping of the graph transduction after the encoding performed by the hidden units.

The development of the context in the NN4G follows the incremental approach of the architectural growing, inducing a nested context development. In particular, the only information available to unit 1 is the label of each vertex (not shown in Fig. 4). Unit 2 gets the v label and the context stored in the preceding unit, i.e., the state value of unit 1 for each adjacent vertex to v (central ellipse in layer 1). For unit 2, we also show the context for the two adjacent vertices of v. By this construction, the state $x_3(v)$ can be computed on the basis of the nested context extended to all the vertices of the input structure (see all the ellipsoids in the first layer). This holds because $x_3(v)$ takes a local context from $\boldsymbol{x}_2(g)$, and indirectly, the context of all the other vertices (from $\boldsymbol{x}_1(g)$ in Fig. 4).

Note that the same schema is used for learning: each state value is frozen after the unit training and each virtual layer corresponds to a subtask splitting (each new unit is fed by different contextual information).

The major aspect of this computation is that the hidden unit for each layer takes a *local* neighborhood $\mathcal{N}(v)$ of each vertex v as input and progressively, by *composition* of context developed in the previous steps, it extends the context of influence to the other vertices. Hence, the context of the NN4G in such frame enlarges its meaning to consider all the vertices of the graph according to the structure topology. In such way, we match the general definition of contextual processing introduced in Section II-B. Moreover, the size of the context window can grow and we do not need to fix it prior to learning. The formalization and the generalization of the concepts proposed here is introduced in Section III.

1) Further Remarks on the Comparison With Recursive Approaches: In addition to the differences and analogies discussed in Section II-C for the architectural and input domain aspects, the shape of the context intuitively shown by the example in Fig. 4 allows us to explain further aspects concerning the information used by NN4G when compared to recursive approaches. These are related to the effects of relaxing causality constraints. In particular, the compositional process presented in Fig. 4 is a way to process contextual information for structures different from the approach used by RNN and RCC, which consider only

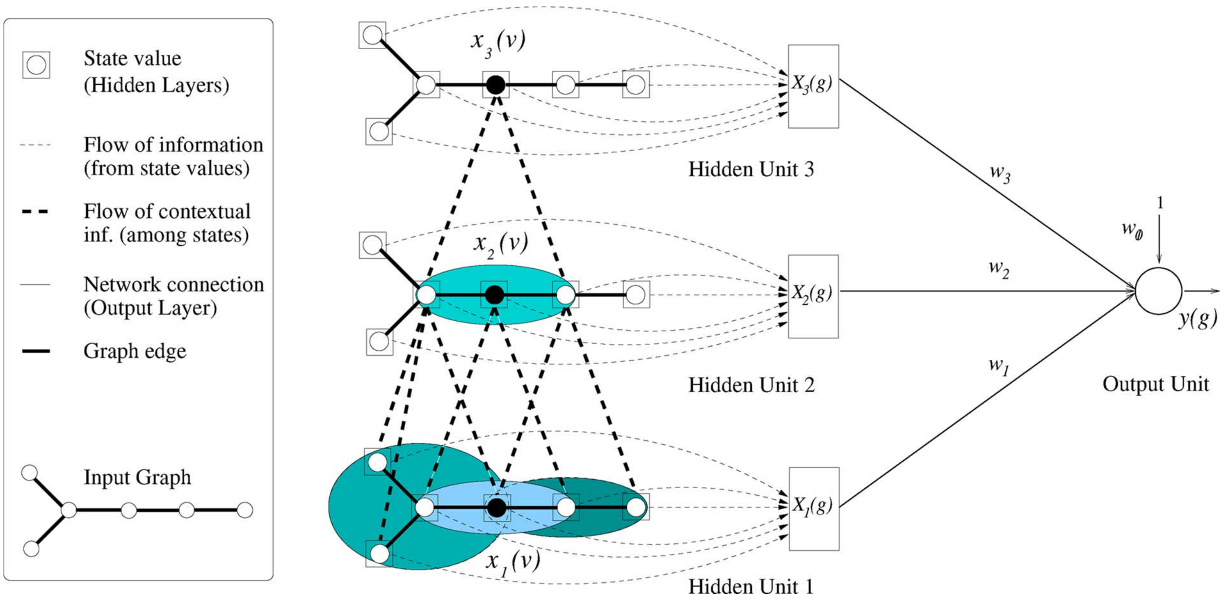

Fig. 4. NN4G encoding of an input graph $(\boldsymbol{x}_1(g), \boldsymbol{x}_2(g), \boldsymbol{x}_3(g))$, the (scalar value) output mapping, and the progressive composition of the context for a vertex v.

predecessors of a vertex according to a causal assumption. In this light, the concept of "context" in recurrent neural networks, as introduced by Elman [34], is to be understood in the restrictive sense as the recursive code/state computed by the *causal* recurrent/recursive dynamics to store "past" information. On the contrary, restricting the domain to sequences (or rooted trees), it is possible to observe that NN4G can treat both predecessors and successors of the current processed vertex. The extension of such concept to graph processing is peculiar to NN4G and does not apply to recurrent neural networks (and also obviously does not apply to flat-vector inputs feedforward neural networks, such as standard CC).

Moreover, NN4G offers a different way to process contextual information for graphs with respect also to *contextual* models, such as CRCC [17]; in contrast to CRCC, which distinguishes between predecessors and successors of each vertex of DPAGs, the context processed by the NN4G is centered in each vertex and grows symmetrically for general graphs including undirected and cyclic graphs (see details in Section III). Finally, with respect to works initiated in [21] and [22], in the NN4G, the contextual information is incrementally added through the insertion of new units. Hence, to compute the contextual information, there is no need to solve and to stabilize a dynamical system. In particular, the use of standard neuron (with no feedbacks) leads to the use of a simple standard strategy for NN4G learning, as summarized in Section II-E.

E. NN4G Learning Algorithm

The proposed method provides a new simple approach for *adaptive* transduction on SD. Note that, as for other neural network approaches, both classification and regression tasks can be addressed by using different output functions (Section II-C). A considerable advantage of the NN4G approach to SDL relays in the adaptive nature of the encoding and mapping functions that compose the graph transduction. More specifically, through the NN4G learning algorithm, both free parameters of the encoding functions (i.e., $\hat{w}$ and $\bar{w}$ parameters) and of the output mapping functions (i.e., w parameters) are estimated to minimize the error function. Hence, the transduction realized by the NN4G can be optimized with respect to the training examples, which can be seen as an approach where the similarity measure on structures can be learned for the task at hand.

The neural realization of the model uses an automatic construction of the network, which can be implemented extending to graph processing the basic approach used in the flat-vector input cascade correlation (see [32] and [33] for various architecture proposals): the NN4G learning algorithm starts with a minimal configuration and adds hidden units during training. Each unit is frozen (no weights are allowed to change) after it is trained. Since the building of the network continues until a desired error is reached, the method allows the automatic determination of the network dimension. The learning is performed by interleaving the minimization of the total error function at the output layer (e.g., by a gradient-descent training), and the maximization of the correlation of the new inserted hidden unit i with the residual error of the output unit o, $E_o(g)$, using a gradient ascent training on

$$S_i = \sum_{o=1}^{M} \left| \sum_{g \in \mathcal{G}} \left(E_o(g) - \bar{E}_o\right) \left(X_i(g) - \bar{X}_i\right) \right| \tag{8}$$

where $\bar{E}_o$ is the mean residual error of the output unit o, $\bar{X}_i$ is the mean value of $X_i(g)$ computed over all graphs, and M is the number of output units. In (8), the patterns are graphs $g \in \mathcal{G}$. If the pattern are the vertices $v \in g$, $X_i(g)$ becomes $x_i(v)$ and we compute the error for each vertex (IO-isomorphic transductions). The weight update for the hidden units is computed by the standard approach of the gradient ascent (extended to consider all the state values of graph vertices). The weight variation for the input from the state variables is computed as

$$\begin{aligned}\Delta\hat{w}_{ij} &= \eta\frac{\partial S_i}{\partial\hat{w}_{ij}}\\ &= \eta\sum_{o=1}^{M}\sigma_o\left(\sum_{g\in\mathcal{G}}\left(E_o(g)-\bar{E}_o\right)\sum_{v\in\mathrm{Vert}(g)}\frac{f'}{k}\sum_{t\in\mathcal{N}(v)}x_j(t)\right)\end{aligned}\tag{9}$$

where σ_o is the sign of the correlation used in (8) and f' is the first derivative of the function f. The derivation with respect to $\bar{w}$ is straightforward just considering $l_j(v)$ as input signal. A batch algorithm is used for training.

The weight update for the output unit is made by the standard approach of the gradient descent. In our experiments, we have also applied the direct computation of the weights by pseudoinverse using singular value decomposition (SVD) instead of normal equation [35].

The learning procedure constitutes a further element of distinction of the NN4G with respect to previous approaches for graphs learning. First, with respect to recurrent/recursive approaches, which require a backpropagation of error through the unfolding network (e.g., backpropagation through time (BPTT) [36], [37] and backpropagation through structure (BPTS) [12] training algorithms), in NN4G, there is no propagation of error signal through unfolded layers. This contributes to the simplicity and to the generality of the approach. Indeed, though the realization by a cascade correlation technique is quite natural, the NN4G can also be implemented with other training algorithms for standard neural units as well. Moreover, the constructive nature of the learning algorithm allows for a splitting of the global training process. Each unit is trained by itself and receives a different input and a different error signal (residual error of the global output) with respect to the other units. In other words, each unit solves a different subtask of the contextual transduction. It is also very likely that such hierarchical architecture of tasks eases the regression/classification problem by decomposing the task into several simpler ones at each layer. Note that layers use different information (from the input graph), with growing depth of their context over the graph topology. Hence, the relevance of the contextual information can be "weighted" according to the task by the learning procedure.

III. Formal Analysis of the Context

In this section, we characterize the information used by NN4G for contextual transductions. We formulate the "shape" of the context exploited by a state variable in terms of the set of state variables that contribute directly or indirectly to its determination. The main aim is to give abstraction and proof of the properties graphically shown in Section II-D. In particular, the derivation of compact expression for the context allows us to abstract the process used by the model to incrementally integrate contextual information with respect to the addition of state variables. Indeed, we believe that, following the line initiated in [17] for recursive models, a formal analysis that is independent from the specific neural realization of the models can be very useful in order to characterize in term of contextual information the properties of various graphical models defined on SDs.

First, to formalize the context concept, let us introduce an extension of the notation introduced in Section II-A.

Let the set $\mathcal{N}(\cdot)$ be defined also when the argument is a subset of vertices $V\subseteq\mathrm{Vert}(g)$: $\mathcal{N}(V)=\bigcup_{v\in V}\mathcal{N}(v)$.

We consider also the repeated application (by a composition in the closed form) of $\mathcal{N}(V)$ for i times denoted with $\mathcal{N}^i(V)$ such that

$$\mathcal{N}^i(V)=\mathcal{N}\left(\mathcal{N}^{i-1}(V)\right)\cup\mathcal{N}^{i-1}(V)$$

where the base case is $\mathcal{N}^1(V)\equiv\mathcal{N}(V)$.

It may be useful to consider the neighborhood also in terms of induced subgraph formed by vertices adjacent to v (including v for the closed form). $\mathcal{N}^d(v)$ is the extension of the neighborhood of v considering the vertex *up to* distance d from v. If we say that v is a d-neighbor of u, for u and v at distance d, then $\mathcal{N}^d(v)$ is the set of k-neighbors of v with $k\le d$, and $\mathcal{N}^d(V)$ is the set of k-neighbors, with $k\le d$, of vertices in V. Moreover, with $x_j.V$, we refer to the set of state variables $\{x_j(v)|v\in V\}$

Let us formally define the concept of context window (as in [16] and [17]).

Definition 1 (Context Window): The context window of a state variable $x_i(v)$, denoted as $\mathcal{C}(x_i(v))$, is the set of all the state variables that (directly or indirectly) contribute to the determination of $x_i(v)$.

The context in Definition 1 can be described in terms of the vertices of the state value graph $\boldsymbol{x}_i(g)$ introduced in Section II-C. In this meaning, the vertices included in the "context of a vertex" v, $\mathcal{C}(x_i(v))$, form an induced subgraph on $\boldsymbol{x}_i(g)$, which corresponds to an induced subgraph on g. Hence, with the notion of the context, we can characterize the graph information used by the model in terms of the set of vertices of g that are considered by the context window on $\boldsymbol{x}(g)$. In other words, it is useful to reason in terms of the vertices of g involved in the context as the graph information that become available to the model. For these reasons, to simplify the interpretation of the formal analysis, in the following, we informally refer the context also as a set of vertices.

We can now introduce the formal expression of the context in terms of $\mathcal{N}^i(v)$ for any state variable of NN4G.

Theorem 1: Given a graph g, $\forall v\in\mathrm{Vert}(g)$, and $i\ge 2$, the NN4G context is given by

$$\mathcal{C}\left(x_i(v)\right)=\bigcup_{j=1}^{i-1}x_j.\mathcal{N}^{i-j}(v)=\bigcup_{j=1}^{i-1}x_{i-j}.\mathcal{N}^{j}(v).\tag{10}$$

Proof: It can be proved by induction on the order of indexes of variables. For the sake of simplicity, let us consider the case of neighborhood in the closed form.

By definition of context window (direct and indirect contribution), and by (3) for any v in $\mathrm{Vert}(g)$, the context window for the state variable $x_i(v)$, $i \geq 2$, can be expressed as

$$\mathcal{C}\left(x_i(v)\right) = \bigcup_{j=1}^{i-1} \bigcup_{u \in \mathcal{N}(v)} x_j(u) \cup \mathcal{C}\left(x_j(u)\right) \tag{11}$$

where we consider the direct contribution of the state variables $x_j(u)$ united to the indirect contributions given by the state variables that contribute to its determination, i.e., $\mathcal{C}(x_j(u))$.

Base Case: $\mathcal{C}(x_1(u)) = \emptyset$ because by definition of x_1 in (2), no state variables contribute to the computation of $\mathcal{C}(x_1(u))$.

For $i = 2$

$$\mathcal{C}\left(x_2(v)\right) = \bigcup_{u \in \mathcal{N}(v)} x_1(u) \cup \mathcal{C}\left(x_1(u)\right) = x_1.\mathcal{N}(v) \tag{12}$$

which is the thesis for $i = 2$.

Inductive Step: We first observe that for any vertex v and for any $i \geq 2$

$$\mathcal{C}\left(x_i(v)\right) \supseteq \mathcal{C}\left(x_{i-1}(v)\right). \tag{13}$$

This is true because, using (11), it holds

$$\begin{aligned}
\mathcal{C}\left(x_i(v)\right) &= \bigcup_{j=1}^{i-1} \bigcup_{u \in \mathcal{N}(v)} x_j(u) \cup \mathcal{C}\left(x_j(u)\right) \\
&= \bigcup_{u \in \mathcal{N}(v)} \bigcup_{j=1}^{i-2} x_j(u) \\
&\quad \cup \mathcal{C}\left(x_j(u)\right) \cup \bigcup_{u \in \mathcal{N}(v)} x_{i-1}(u) \cup \mathcal{C}\left(x_{i-1}(u)\right) \\
&= \mathcal{C}\left(x_{i-1}(v)\right) \cup \bigcup_{u \in \mathcal{N}(v)} x_{i-1}(u) \cup \mathcal{C}\left(x_{i-1}(u)\right).
\end{aligned}$$

Hence, $\mathcal{C}(x_i(v)) \supseteq \mathcal{C}(x_{i-1}(v)) (i \geq 2)$.

Since, according to (13), $\mathcal{C}(x_{i-1}(u))$ occurring in (11) for $j = i-1$ also includes the cases for $j = 1, \ldots, i-2$, then (11) can be rewritten as

$$\mathcal{C}\left(x_i(v)\right) = \bigcup_{j=1}^{i-1} \bigcup_{u \in \mathcal{N}(v)} x_j(u) \cup \mathcal{C}\left(x_{i-1}(u)\right). \tag{14}$$

By inductive hypothesis $\mathcal{C}(x_{i-1}(u)) = \bigcup_{j=1}^{i-2} x_{i-1-j}.\mathcal{N}^j(u)$, therefore

$$\begin{aligned}
\mathcal{C}\left(x_i(v)\right) &= \bigcup_{j=1}^{i-1} \bigcup_{u \in \mathcal{N}(v)} x_j(u) \cup \bigcup_{u \in \mathcal{N}(v)} \bigcup_{j=1}^{i-2} x_{i-1-j}.\mathcal{N}^j(u) \\
&= \bigcup_{j=1}^{i-1} x_j.\mathcal{N}(v) \cup \bigcup_{j=1}^{i-2} x_{i-(j+1)}.\mathcal{N}\left(\mathcal{N}^j(v)\right) \\
&= \left(\bigcup_{j=1}^{i-2} x_j.\mathcal{N}(v) \cup x_{i-1}.\mathcal{N}(v) \right) \\
&\quad \cup \bigcup_{j=1}^{i-2} x_{i-(j+1)}.\mathcal{N}^{j+1}(v) \\
&= \bigcup_{j=1}^{i-1} x_{i-j}.\mathcal{N}^j(v).
\end{aligned}$$

■

This theorem formally shows that the context grows one step ahead, for each added unit, in the extended neighborhood of each vertex: the dimension of the context is proportional to the number of units, as intuitively shown in Section II-D and in Fig. 4. In other words, the context of a vertex v for NN4G is composed by the neighbors $\mathcal{N}^d(v)$, where d is proportional to the number of units and it can grow one step ahead for each added unit.

It is worth noting that the context expressed in (10) is composed, according to the model equations, by a set of smaller and local context developed by the previous units. The structure of the composition is given by the topology of the input graph as expressed in (10) by the composition of the neighbors $\mathcal{N}(v)$ of each vertex v. The aim of the formulation expressed in Theorem 1 is to synthesize and to generalize the possible complex information flows occurring on different possible input graphs topologies.

In order to show that NN4G can compute contextual transduction (in the meaning given in Section II-B), we provide the following property.

Theorem 2: Given a finite size graph g, there exists a finite number of h state variables such that for each $v \in \mathrm{Vert}(g)$ the context $\mathcal{C}(x_h(v))$ involves all the vertices of g. In particular, any index $h > d$ satisfies the proposition if d is the greatest distance between any two vertices of g (or the "diameter" of the graph).

Proof: For $h = d+1$, $\mathcal{C}(x_h(v)) \supseteq \bigcup_{j=1}^{h-1} x_{i-j}.\mathcal{N}^j(v) = \bigcup_{j=1}^{d} x_{i-j}.\mathcal{N}^j(v)$. Hence, the context of v involves all the vertices in the extended neighborhood of v up to a distance d, which assures to involve all the vertices of the graph (i.e., all the vertices for which there exists a path from v in g or in the undirected version of g). ■

Theorem 2 formalizes the intuitive reasoning stated in Section II-D, showing that although each unit exploits local information, the context developed by NN4G can be extended to a global information over the graph.

The class of graph used in the Theorem 1 is a general class, which corresponds to a neighborhood defined on the undirected version of the graph (Section II-A). The different cases of directed/undirected, ordered/unordered graphs give rise to specific views of the theorem according to different possible instance of the NN4G (see Section II-C).

Such cases can be derived looking to the neighborhood for a DAG (or a DPAG) in terms of the decomposition with respect to predecessors and successors (as introduced in Section II-A), i.e.,

$$\mathcal{N}(v) = \mathcal{P}(v) \cup \mathcal{S}(v). \tag{15}$$

Other contextual models such as CRCC (which can be applied restricting the domain to DAG and DPAG) show an expansion of the context that follows a recursive dynamics for successors of a vertex, hence separating the grow of the context window for successors and predecessors of each vertex [17]. These approaches are particularly suited for causal processing. However, it can be observed that for some problems it occurs that there is no canonic direction of influence (for instance, in the case of bidirectional approaches to sequential transductions). In such cases, the context to various sides should not in principle be treated differently among them (i.e., with a different dynamics in the expansion). Thus, even restricting the input domain to DAG and DPAG, Theorem 1 allows us to characterize NN4G with respect to previous contextual neural networks approaches showing that for NN4G, the context, because of (15), can grow symmetrically on the neighbors of each vertex.

IV. Experimental Results

As a first proof-of-principle application of the current model proposal, we present in the following an experimental assessment devoted to investigating the behavior of the model with respect to the extension of the causality assumption and of the class of data domain in graph pattern problems.

For the sake of introduction of the new methodology in the SDL field, it is worth first testing the generality of the model on a real-world problem already treated by connectionist related models for SD, with emphasis on causal RNN, CRCC, and kernel methods (using a restricted class of tree structures). In addition, this allows us to assess whether the relaxing of the recursive/dynamical system full-power assumed in NN4G is to be paid in terms of predictive accuracy. The second experiment is devised to show that the new model can, at least in principle, tackle directly with classes of data that cannot be treated directly by standard causal RNN models, specifically undirected cyclic graphs.

A. QSPR Analysis of Alkanes

We report the first assessment of NN4G applied to an SD involving chemical compounds for quantitative structure-property relationship (QSPR) analysis.

QSPR/QSAR (quantitative structure-activity relationship) are fundamental aspects in cheminformatics, where the aim is to correlate chemical structure of molecules with their properties (or biological activity) in order to achieve prediction. The high pressure in searching for new computational methods that can reduce the burden of developing new materials and new drugs posed by the current research in chemistry and pharmacology is a strong motivation to study such problems. Since molecules are naturally described via varying size structured representation, tools for direct structure domain learning seem to provide a new perspective in QSPR/QSAR analysis, with respect to traditional features-based approaches.

The set of *alkanes* (saturated hydrocarbons) molecules provides a concrete example of real objects belonging to an SD. It is interesting to note that, as a historical fact, the mathematical notion of *tree* structure has been introduced by the mathematician A. Cayley in 1857 [38] when he was trying to model and to enumerate alkanes compounds. The study of the QSPR for *alkanes* aimed at predicting the boiling point property (e.g., measured in Celsius degrees), provides a simple and clear example of the application goal of the structure transduction: given an alkane compound, we would like to predict the value of the boiling point temperature just looking at its chemical structure. Using a set of compounds for which we already knew the boiling point values (training set), the model learns to map the compound structure to the real value that measures the boiling point property. The boiling temperature of alkanes is frequently used as a benchmark property in testing QSPR models. Due both to its high relevance in the design of industrial process and to its well-known characteristics, many *ad hoc* methods have been developed for the prediction of boiling point of organic compounds since the field origins. At the same time, it is still a currently studied problem, e.g., [39]. For these reasons, we consider this QSPR problem in the following to test the proposed model on a real-world problem characterized by an SD. The presence of previous approaches allows us also to compare our method with *ad hoc* and structure-based methods for the problem.

In particular, we compare NN4G with respect to CRCC [17], which improves previous results obtained by standard QSPR approaches, causal recursive models, and "kernel for trees" methods (see [15]–[17], [40], and the references therein). In fact, through different applications to these data, a set of results for recursive causal models was obtained: RCC models achieved a test set mean absolute error on a tenfold cross validation (tenfold CV TS mean AE) of 2.87 °C [17]. Previously, the same model was used in [15] and [41]. In [15] and [41], the original aim was the assessment of recursive methods by comparison with standard multilayer feedforward neural networks using an *ad hoc* vectorial representation of alkanes that yields a results of 3.01 °C for tenfold CV TS mean AE [42], which in turns improved previous traditional approaches to the problem. Competitive results (with an error of around 2.93 °C using the training stop criterion used for RCC) were also obtained through support vector machine with tree kernel method [8], [40]. Such results yield the potential to work directly with structured data rather than to preprocess the molecular data using an encoding of the structures into flat representations (typically by vectors of *ad hoc* descriptors). In particular, the results show that generality and expressivity of a structure-based approach are not at the expense of predictive accuracy. Other results have been recently presented in [43] using a convolution kernel ("kernels on prolog ground terms") approach, which however does not improve the previous mentioned results. Further improvements (tenfold CV TS mean AE of 2.56 °C) were obtained by a CRCC approach, showing the advantage of a contextual approach, even if restricted to K*-ary* trees representation in [17].

Alkanes are free trees. Basing on a hydrogens suppressed representation, carbon-hydrogens groups are associated with vertices, and bonds between carbon atoms are represented by edges. This is a regression task: the target is the boiling point expressed in Celsius degrees into the range $[-164, 174]$ for this benchmark set based on all the 150 alkanes with up to ten carbon atoms ($\mathrm{C}_n\mathrm{H}_{2n+2}$). There is a total of 1331 vertices in the data set.[3]

The most significant aspect of this experiment is that NN4G is able to treat alkanes in their natural form of (undirected) free trees instead of imposing a root, a direction, and an order on the structure by conventional rule to obtain K*-ary* trees, as needed for the RNN models used in [15] and [17] (see the different representations in Fig. 5). Hence, we avoid the use of conventional representation rules aimed at defining rooted ordered/positional

[3] The data set is fully reported in [15] and also available at http://www.di.unipi.it/~micheli/dataset

TABLE I
TENFOLD CV RESULTS FOR CRCC AND NN4GS

Model	TR Max AE	TR Mean AE		TS Max AE		TS Mean AE		#Units
		Mean	Var.	Mean	Var.	Mean	Var.	Mean
CRCC QP	8	1.68	0.056	8.29	16.28	**2.56**	0.639	137.8
NN4G QP	8	1.95	0.045	8.53	23.46	**2.35**	0.234	114.5
NN4G SVD	8	2.05	0.036	6.86	2.40	**2.34**	0.098	13
NN4G SVD	5	1.33	0.008	5.03	2.54	**1.74**	0.054	23.1

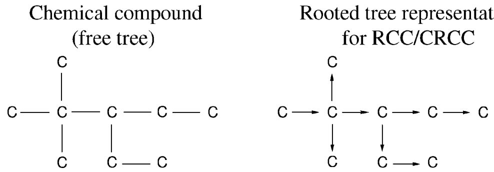

Fig. 5. Hydrogens suppressed representations for 2,2-dimethyl-3-ethylpentane.

trees for the alkanes [15], and we relax the causality assumption, enhancing the advantages deriving by contextual processing.

In the original RCC/CRCC experiments [15], [17], learning was stopped when the maximum absolute error for a single compound in the training set was below 8 °C (TR max AE), or when a maximum number of hidden units (150) was reached. A tenfold cross validation on the data set has been performed. For each fold, five different learning trials were performed and averaged to collect the results.

In Table I, we report the mean together with the variance of the training mean absolute error (TR mean AE) and of the test maximum and mean absolute error (TS max/mean AE) on a tenfold cross validation. We report also the number of hidden units added to the network (#Units) and the error threshold used for the training (TR max AE) (as originally assumed for the RCC/CRCC experiments [15], [17]). QP and SVD in the first column indicate the use of quick propagation and of pseudoinverse algorithm on the output unit, respectively. The mean absolute error on test set is shown in bold.

In particular, it is possible to observe an improvement in average of the test mean error by the new model using the same linear output unit used for CRCC model, with a slight reduction of the number of hidden units. Further improvements in the efficiency are achieved using a direct computation of the output unit parameters by a pseudoinverse algorithm (SVD). The value of k in (5) was chosen following the normalized sum, however no major differences were observed using $k = 1$. Since no overtraining was observed, and due to the low number of units obtained with SVD training, we explored the possibility to fix the maximum absolute error tolerance to 5 °C, showing the possibility to predict the boiling point of alkanes with an average mean error under 2 °C. The current improvement of the results accuracy extends the progress introduced through the relaxing of the causality initiated by the CRCC. Moreover, the result shows that also the weight sharing assumed in the instance of NN4G does not hamper the efficacy while yielding an efficient approach. In fact, the numbers of parameters used by the NN4G model are around 6850 for TR max AE 8, 118 for TR max AE 8 (SVD), and 325 for TR max AE 5 (SVD) experiments. These numbers are very small if compared with those obtained in the CRCC experiments, i.e., around 38 500 [17].

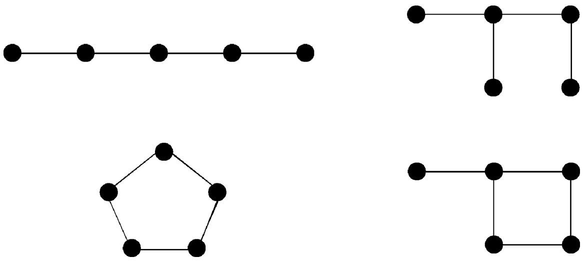

Fig. 6. Examples of two acyclic and two cyclic graphs.

B. Classification of Cyclic/Acyclic Graphs

This artificial task allows testing the capability of NN4G to learn a relevant topological feature, i.e., the occurrence of cycles in the input undirected graphs, which cannot be directly treated by RNN. The data set is made up of 150 cyclic graphs and 150 acyclic ones (Fig. 6), with three up to ten vertices and a total of 2670 vertices. The task is a binary classification of acyclic (1) and cyclic (0) graphs.

For this task, NN4G achieves 100% classification accuracy both for training and test set over all the folds of tenfold cross validation with five trials performed for each fold. The model solves the task adding just two hidden units. In fact, the second unit is able to distinguish the ratio between the number of edges and vertices in the graph, which is a sufficient feature to discriminate the input graphs on the basis of the occurrence of cycles in its topology.

More in general, the experiments allow us to show, at least in principle, the new characteristics of the approach in terms of extension of the class of input graphs to undirected cyclic graphs. The empirical results are a clear evidence of this possibility.

V. DISCUSSION AND FUTURE DIRECTIONS

Through the NN4G model, we showed that simple solutions are possible for dealing with fairly general classes of graphs by subsymbolic approaches. Comparing with RNN approaches, both the aspects of simplicity and of extension of the class of data are related to the relaxation of the causality constraints in the encoding function of NN4G. The flexibility of the framework defined for NN4G allows also to deal with transductions on hierarchical graph subclasses data such are DPAG and K-*ary* trees, including supersource transduction for causal tasks (which is a special case of IO-isomorphic transductions with output focused on the supersource/root vertex). Of course, if the causality holds for the task at hand and the class of data is restricted to hierarchical data with supersource, the use of full-power recursive neural networks still constitutes a principled approach to the problem [44]. The preference among the models clearly depends on the class of graphs and tasks at hand.

However, the aim in this paper was to extend the set of neurocomputing tools beside RNN in order specifically:

- to tackle contextual processing (when the causal assumption hampers the solution or its soundness is unknown);
- to deal with a large class of graphs (including cyclic and undirected graphs).

Analogous discussions concern the realization of the output mapping function. The NN4G model allows the realization of IO-isomorphic transductions (with a structured output where the use of $\boldsymbol{X}(g)$ is not necessary). While this opens new application possibilities, the theoretical issues for general graph transductions deserve further research. However, the focus here was on graph pattern tasks, which are particularly important for a wide set of SDL applications, for instance, in the field of cheminformatics. In the latter case, various computational schemes are possible to realize the $\boldsymbol{X}(g)$ operator. It is worth noting that, although we assumed in Section II-C a simple global average, the arbitrariness of this operator is automatically compensated adapting its results to the task by the learning process that determines both the encoding and output mapping.

Experiments provide empirical evidences of the advantages due to the relaxation of the causality constraints. In effect, although the main characteristics of NN4G are the simplicity of the architecture and of the learning algorithm (with respect to recursive approaches) and the generality of the class of structures that can be treated, the results on QSPR of alkanes showed also an improvement of the state-of-the-art performance for the task. In that case, NN4G allows us to exploit directly the class of undirected free trees. Such advantages are even more clear for the second experiment where the class of data was extended to undirected cyclic graphs, which cannot be directly treated by causal approaches for hierarchical data. Moreover, the cyclic/acyclic experiments allowed us to test the capability of the new model to recognize the occurrence of cycles, which is a relevant topological feature for graphs. The NN4G efficiently achieved an optimal performance, showing the potentiality of the model to treat expressive representation of the data.

The experiments allowed us to observe that the efficiency of the model implemented through a full stationarity assumption was not to be paid in terms of efficacy. However, in general, the *stationarity* is another computational aspect that can be tailored to model various classes of data, using the expressivity needed for each case (see Section II-C). For instance, the treatment of positional or labeled edges can be included in such models. These alternatives can be viewed as a means to remove the constraints on the ordering (or position) of the vertex neighbors or, on the contrary, as a means to enrich the representation expressivity whenever positioning or other labeling are needed effectively to model the data. This flexibility can therefore be exploited in several ways to model specific tasks and applications according to the types, categories, and abstraction levels defined for the graph components.

An emerging discussion topic concerns a critical comparison with respect to other subsymbolic approaches, such as the current developments of kernel-based methods for SDs (see, e.g., [4]–[7]). The RNN approach offers an adaptive transduction system developing the (state variables) encoding space (or feature space) by learning, while, typically, the kernel imposes prior to learning similarity measures for the data exploiting an (implicit) high-dimensional feature space. On the other hand, the kernel is highly flexible in its definition allowing to include a wide set of possible input data. The comparison between RNN and kernel for structured data is still an open field of research and some interesting initial results and discussions have been recently reported in [8], [40], [45], and [46]. In this view, an interesting perspective is offered by models as NN4G and [21], which open the possibility of an adaptive transduction for a more general class of graphs with respect to causal RNN.

Concerning the applications, the cheminformatics field can clearly take advantage of an extension of the classes of data to undirected acyclic/cyclic graphs. Moreover, other interesting fields can be considered, especially for contextual tasks based on local information of each vertex. Relevant examples of such characteristic can be found for the treatment of text categorization for taxonomy construction [47], or for web documents and NLP.

VI. Conclusion

We introduced a new ML approach for graphs processing, called NN4G. The proposed method provides a further advancement in the set of adaptive transduction approaches, which enriches the set of neural networks models for learning in SDs. Summarizing, the main features that characterize the NN4G approach are the simplicity of the neural architecture and learning algorithm, the generality of the classes of graphs that can be dealt with, and the adaptive nature of the transduction on SD. NN4G does not entail a recursive or dynamical system. It belongs to feedforward architecture class of models. Hence, we avoid both the assumption of causality over hierarchical data and the possible introduction of cyclic dependencies in the definition of the system state variables for cyclic and/or undirected graphs (which in case should be treated as a dynamical system problem; see Section I). As a result of the former aspect, we relaxed the need of the RNN constraints related to the topological order on structures, allowing a natural extension of the domain to acyclic/cyclic, directed/undirected labeled graphs. To achieve these ends, the NN4G exploits, as main ingredients, a constructive neural network applied by a traversal process to the input graphs and the contextual information of state variables for each vertex.

Contextual processing of SDs opens new ways to deal with graph data. In this paper, the discussion has been based on the new NN4G approach that iteratively integrates contextual information. Through the composition of the local context of each unit, the NN4G model can deal with contextual information of increasing extension, which is built according to the graph topology. Hence, although the single units use local information on the graph, the global model can deal with learning tasks involving graph patterns and its whole topological information.

In this paper, we characterized the information used by the model as a model that implements a contextual transduction, proposing a compact expression that abstracts and generalizes the process of the context development. Moreover, the analysis of the properties in terms of the context opens a way to characterize the different approaches to SDL (in particular, with respect to previous RNN approaches). In effect, the formal analyses used here and in [17], which are independent from the specific neural realization of the NN4G, can be extended to formally characterize the context of other graphical models defined on SDs.

The quite large classes of SDs encompassed by the NN4G computational framework, and by the set of previous or emerging recursive approaches, in classification and regression tasks for both flat and structured output, provide a source of relevant flexibility in the field of SDL. Indeed, besides the specific computational characteristics of the various approaches, the ability to treat the proper inherent nature of the input data is a key feature for the successful application of the ML methodologies. The author hopes that the introduction of simple and general approaches can contribute to the further exploitation of neural network approaches for widespread applications of the SD and relational learning methods.

References

[1] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. Cambridge, MA: MIT Press, 2007.

[2] L. D. Raedt, "Statistical relational learning: An inductive logic programming perspective," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2005, vol. 3721, pp. 3–5.

[3] S. Džeroski and N. Lavrač, *Relational Data Mining*. Berlin, Germany: Springer-Verlag, Sep. 2001.

[4] T. Gärtner, "A survey of kernels for structured data," *SIGKDD Explor. Newslett.*, vol. 5, no. 1, pp. 49–58, 2003.

[5] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge, U.K.: Cambridge Univ. Press, Jun. 2004.

[6] B. Hammer and B. J. Jain, "Neural methods for non-standard data," in *Proc. Eur. Symp. Artif. Neural Netw.*, 2004, pp. 281–292.

[7] B. Schölkopf, K. Tsuda, and J. Vert, *Kernel Methods in Computational Biology*. Cambridge, MA: MIT Press, 2004.

[8] A. Micheli, A. Sperduti, and A. Starita, "An introduction to recursive neural networks and kernel methods for cheminformatics," *Curr. Pharm. Des.*, vol. 13, Special Issue on "Ground-Breaking Mathematical Models for Basic and Applied Research", no. 14, pp. 1469–1495, 2007.

[9] J. Ramon and T. Gartner, "Expressivity versus efficiency of graph kernels," in *Proc. 1st Int. Workshop Mining Graphs Trees Sequences*, 2003 [Online]. Available: http://www.ar.sanken.osaka-u.ac.jp/~washio/list/7.pdf

[10] T. Gartner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Proc. 16th Annu. Conf. Comput. Learn. Theory/7th Kernel Workshop*, Aug. 2003, pp. 129–143.

[11] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, pp. 768–786, Sep. 1998.

[12] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 714–735, May 1997.

[13] P. Frasconi, M. Gori, A. Küchler, and A. Sperduti, "From sequences to data structures: Theory and applications," in *A Field Guide to Dynamical Recurrent Networks*, K. J. F. and K. S. C. , Eds. Piscataway, NJ: IEEE Press, 2001, ch. 19.

[14] S. Fahlman and C. Lebiere, "The recurrent cascade-correlation learning architecture," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-91-100, May 1991.

[15] A. Bianucci, A. Micheli, A. Sperduti, and A. Starita, "Application of cascade correlation networks for structures to chemistry," *Appl. Intell.*, vol. 12, pp. 117–146, 2000.

[16] A. Micheli, "Recursive processing of structured domains in machine learning," Ph.D. dissertation, Dept. Comput. Sci., Univ. Pisa, Pisa, Italy, 2003, TD-13/03.

[17] A. Micheli, D. Sona, and A. Sperduti, "Contextual processing of structured data by recursive cascade correlation," *IEEE Trans. Neural Netw.*, vol. 15, no. 6, pp. 1396–1410, Nov. 2004.

[18] M. Bianchini, M. Gori, and F. Scarselli, "Processing directed acyclic graphs with recursive neural networks," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1464–1470, Nov. 2001.

[19] B. Hammer, A. Micheli, and A. Sperduti, "Universal approximation capability of cascade correlation for structures," *Neural Comput.*, vol. 17, no. 5, pp. 1109–1159, 2005.

[20] M. Bianchini, M. Gori, L. Sarti, and F. Scarselli, "Recursive processing of cyclic graphs," *IEEE Trans. Neural Netw.*, vol. 17, no. 1, pp. 10–18, Jan. 2006.

[21] F. Scarselli, A. Tsoi, M. Gori, and M. Hagenbuchner, "A new neural network model for graph processing," Univ. Siena, Siena, Italy, Tech. Rep. DII 01/05, Jul. 2005.

[22] V. Di Massa, G. Monfardini, L. Sarti, F. Scarselli, M. Maggini, and M. Gori, "A comparison between recursive neural networks and graph neural networks," in *Proc. World Congr. Comput. Intell./Int. Joint Conf. Neural Netw.*, 2006, pp. 778–785.

[23] B. Hammer and J. J. Steil, "Tutorial: Perspectives on learning with RNNs," in *Proc. Eur. Symp. Artif. Neural Netw.*, 2002, pp. 357–368.

[24] O. De Jesus and M. T. Hagan, "Backpropagation algorithms for a broad class of dynamic networks," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 14–27, Jan. 2007.

[25] W. Yilei, S. Qing, and L. Sheng, "A normalized adaptive training of recurrent neural networks with augmented error gradient," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, pp. 351–356, Feb. 2008.

[26] R. Ilin, R. Kozma, and P. J. Werbos, "Beyond feedforward models trained by backpropagation: A practical training tool for a more efficient universal approximator," *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 351–356, Jun. 2008.

[27] A. Micheli and A. Sestito, "A new neural network model for contextual processing of graphs," in *Neural Nets: 16th Italian Workshop on Neural Nets, WIRN2005, June 8-11, 2005*, ser. Lecture Notes in Computer Science, B. Apolloni, M. Marinaro, G. Nicosia, and R. Tagliaferri, Eds. Berlin, Germany: Springer-Verlag, 2006, vol. 3931, pp. 10–17.

[28] A. Micheli and A. Sestito, "Dealing with graphs by neural networks," in *Proc. Workshop Sub-Symbolic Paradigms for Learning in Structured Domains/16th Eur. Conf. Mach. Learn./9th Eur. Conf. Principles Practice Knowl. Disc. Databases*, M. Gori and P. Avesani, Eds., Oct. 2005, pp. 66–75.

[29] P. Baldi, S. Brunak, P. Frasconi, G. Pollastri, and G. Soda, "Exploiting the past and the future in protein secondary structure prediction," *Bioinformatics*, vol. 15, no. 11, pp. 937–946, 1999.

[30] P. Baldi and G. Pollastri, "The principled design of large-scale recursive neural network architectures-DAG-RNNs and the protein structure prediction problem," *J. Mach. Learn. Res.*, vol. 4, pp. 575–602, 2003.

[31] A. Ceroni, P. Frasconi, and G. Pollastri, "Learning protein secondary structure from sequential and relational data," *Neural Netw.*, vol. 18, no. 8, pp. 1029–1039, 2005.

[32] S. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-90-100, Aug. 1990.

[33] E. Littmann and H. Ritter, "Learning and generalization in cascade network architectures," *Neural Comput.*, vol. 8, no. 7, pp. 1521–1539, 1996.

[34] J. L. Elman, "Finding structure in time," *Cogn. Sci.*, vol. 14, pp. 179–211, 1990.

[35] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1992.

[36] S. Haykin, *Neural Networks, A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[37] K. Doya, "Recurrent networks: Supervised learning," in *The Handbook of Brain Theory and Neural Networks*, M. Arbib, Ed. Cambridge, MA: MIT Press, 1995, pp. 796–800.

[38] A. Cayley, "On the theory of analytic forms called trees," *Philos. Mag.*, vol. 13, pp. 19–30, 1857.

[39] O. Ivanciuc, T. Ivanciuc, and A. Balaban, "Quantitative structure-property relationships for the normal boiling temperatures of acyclic carbonyl compounds," *Internet Electron. J. Molecular Design*, vol. 1, no. 5, pp. 252–268, May 2002.

[40] A. Micheli, F. Portera, and A. Sperduti, "A preliminary empirical comparison of recursive neural networks and tree kernel methods on regression tasks for tree structured domains," *Neurocomputing*, vol. 64, pp. 73–92, Mar. 2005.

[41] A. Bianucci, A. Micheli, A. Sperduti, and A. Starita, *A Novel Approach to QSPR/QSAR Based on Neural Networks for Structures*, H. Cartwright and L. M. Sztandera, Eds. New York: Springer-Verlag, Mar. 2003, pp. 265–296.

[42] D. Cherqaoui and D. Villemin, "Use of neural network to determine the boiling point of alkanes," *J. Chem. Soc. Faraday Trans.*, vol. 90, no. 1, pp. 97–102, 1994.

[43] A. Passerini and P. Frasconi, "Kernels on prolog ground terms," in *Proc. 19th Int. Joint Conf. Artif. Intell.*, Edinburgh, Scotland, U.K., 2005, pp. 1626–1627.

[44] B. Hammer, "Learning with recurrent neural networks," in *Lecture Notes in Control and Information Sciences*. Berlin, Germany: Springer-Verlag, 2000, vol. 254.

[45] S. Menchetti, F. Costa, P. Frasconi, and M. Pontil, "Comparing convolution kernels and recursive neural networks for learning preferences on structured data," in *Proc. IAPR—TC3 Int. Workshop Artif. Neural Netw. Pattern Recognit.*, 2003 [Online]. Available: http://www.dsi.unifi.it/ANNPR/CR/27.pdf

[46] S. Menchetti, F. Costa, P. Frasconi, and M. Pontil, "Wide coverage natural language processing using kernel methods and neural networks for structured data," *Pattern Recognit. Lett.*, vol. 26, no. 12, pp. 1896–1906, 2005.

[47] G. Adami, P. Avesani, and D. Sona, "Clustering documents into a web directory for bootstrapping a supervised classification," *J. Data Knowl. Eng.*, vol. 54, pp. 301–325, 2005.

Alessio Micheli (S'00–M'04) received the laurea degree and the Ph.D. degree in computer science from the University of Pisa, Pisa, Italy, in 1998 and 2003, respectively.

Currently, he is Assistant Professor at the Computer Science Department, University of Pisa. He has been Visiting Fellow at Wollongong University, Australia, and at the Neural Computing Research Group (NCRG), Aston University, U.K. His research interests include machine learning, neural networks, structured domains learning, relational learning, recursive neural networks, kernel-based learning for nonvectorial data, applications to natural science and biochemistry, cheminformatics and quantitative structure property/activity relationships (QSPR/QSAR) analysis. He has been involved in several National and European projects in the field of computational intelligence and cheminformatics.

Dr. Micheli is a member of the IEEE Computational Intelligence Society and of the Cheminformatics and QSAR Society.