Brief Papers

Processing Directed Acyclic Graphs With Recursive Neural Networks

Monica Bianchini, Marco Gori, and Franco Scarselli

***Abstract*—*Recursive neural networks* are conceived for processing graphs and extend the well-known recurrent model for processing sequences. In Frasconi *et al.*, recursive neural networks can deal only with directed ordered acyclic graphs (DOAGs), in which the children of any given node are ordered. While this assumption is reasonable in some applications, it introduces unnecessary constraints in others. In this paper, it is shown that the constraint on the ordering can be relaxed by using an appropriate weight sharing, that guarantees the independence of the network output with respect to the permutations of the arcs leaving from each node. The method can be used with graphs having low connectivity and, in particular, few outcoming arcs. Some theoretical properties of the proposed architecture are given. They guarantee that the approximation capabilities are maintained, despite the weight sharing.**

***Index Terms*—Function approximation, graph processing by neural networks, permutation invariant algebras, recursive neural networks.**

I. Introduction

In several applications, the information which is relevant for solving problems is organized in entities and relationships among entities. The simplest dynamic data type is the *sequence*, which is a natural way of representing time dependences. Recurrent networks were proved to be a powerful tool in modeling time-dependent phenomena, since they are able to take into account both the symbolic and the subsymbolic information collected in the sequence. Nevertheless, there are domains in which the data involved are encoded in more complex structures, like trees or graphs. *Recursive neural networks* [1]–[3] are a new connectionist model particularly suited for processing graphs and can be viewed as an extension of recurrent neural networks. The possibility of modeling data by graphical structures allows us to effectively face problems arising from automated deduction systems [4], computational chemistry [5] and Internet surfing with recommender systems [6].

However, according to the model initially proposed in [1], recursive neural networks can only deal with limited families of graphs, like directed positional acyclic graphs (DPAGs) and directed ordered acyclic graphs (DOAGs). In the case of DPAGs, each arc coming from a node has an assigned and specific position. In other words, any rearrangement of the children of any node produces a different graph. While such an assumption is useful in some applications, in others it introduces an unnecessary constraint on the representation. For example, this hypothesis is not suitable for the representation of a chemical compound and might not be adequate for several pattern recognition problems.

The concept of invariance or symmetry has emerged as one of the key ideas in many different areas such as physics, mathematics, psychology, and engineering [7] and theoretical and applied research is flourished in this field since the beginning of the 1960s. From a theoretical point of a view, seminal contributions in permutation invariant algebras can be found in [8], [9]. Significant applications are in computer vision [7], [10], in signal theory [11] and in pattern recognition [12], [13]. Even in the neural-network research field there have been some attempts to construct application-specific architectures capable of permutation invariance with regard to inputs. Neural-network approaches aimed at preserving invariance are, in many cases, based on some preprocessing [14], [15] or on the extraction of features [16] which are invariant under input transformations (image or signal translations, image rotations, etc.,). In other cases, invariance is realized by adopting *ad hoc* network models that preserve the output under particular input transformations [17].

In this paper, we show how the constraint on the ordering can be relaxed by introducing an appropriate weight sharing in the recursive neural network, in order to guarantee that the output of the network is independent of the permutations of the arcs of each node. From a practical point of view, the approach can be used with low connectivity graphs, because the number of neurons grows as the factorial of the maximal outdegree. Moreover, we discuss the theoretical properties of the proposed architecture with particular attention to its computational capabilities. It will be proved that the architecture has the universal approximation property and can approximate up to any degree of precision any functions on DAGs.

This paper is organized as follows. In the next section, the notation is introduced. In Section III, we show how general recursive architectures work on DPAGs, while in Section IV, the model is specialized in order to process directed acyclic graphs (DAGs). In the same section, some theoretical results are obtained to assess the computational power of the proposed architecture. Some conclusions are drawn in Section V.

II. Notation

In the following, a graph G is defined as a triple $(V, A, \mathcal{L})$, where V is the set of nodes, $A \subseteq V^2$ is the set of arcs, $\mathcal{L} : V \rightarrow L$ is a labeling function and $L \subset \mathbb{R}^m$ is a finite set of labels.

Manuscript received October 19, 2000; revised May 30, 2001.

The authors are with the Department of Ingegneria dell'Informazione, Università di Siena, Siena 53100, Italy (e-mail: monica@dii.unisi.it; marco@dii.unisi.it; franco@dii.unisi.it).

Publisher Item Identifier S 1045-9227(01)09514-5.

Given any node $v \in V$, ch$[v]$ represents the set of *children* of v. The *outdegree* of v, od$[v]$, is the cardinality of ch$[v]$ and $o = \max_{v \in V}\{\text{od}[v]\}$ is the maximal outdegree. The presence of an arc (v, w) in a labeled graph shows the existence of some sort of causal link between the variables contained in v and w.

In this paper, we consider the class of DAGs, where there is not any path connecting a node to itself and the arcs are directed, i.e., $(v, w) \neq (w, v)$. Therefore, on the set of nodes of a DAG a partial ordering is defined such that $v \prec w$ if a directed path exists starting from v to w. DPAGs differ from DAGs since an ordering is defined on the children of each node by an injective function $o_v : \text{ch}[v] \rightarrow \{1, \ldots, o\}$, which assigns a position $o_v(c)$ to each child c of v. The position of the children is a distinctive feature of a DPAG, such that two DPAGs are equal only when a bijective correspondence exists, not only between nodes and arcs, but also between the position of all the children of each node. From a notational point of view, a DPAG is represented by a quadruplet $(V, A, \mathcal{L}, O)$, where $O = \{o_1, \ldots, o_{|V|}\}$ is the set of functions defining the positions of the children. Moreover, in DAGs the children of a node v are correctly denoted by the set ch$[v]$, whereas for DPAGs they can be more properly represented by a fixed dimension vector $[v_1, \ldots, v_o]$, where some components may be null when no child is associated to the corresponding positions.

Finally, we assume that all the graphs have a supersource. A supersource s is a node from which all the other nodes can be reached (formally, $s \prec v$, for all $v \in V, v \neq s$).

III. Recursive Neural Networks for Processing DPAGs

Recursive neural networks process a DPAG G following the scheme described in Fig. 1. The recursive neural network is unfolded through the graph structure, producing the *encoding* network. At each node v, the *state* $\mathbf{X}_v$ is computed by a *transition* function of the input label $\mathbf{U}_v$ and the state of its children $\mathbf{X}_{v_1}, \ldots, \mathbf{X}_{v_o}$

$$\mathbf{X}_v = f(\mathbf{X}_{v_1}, \ldots, \mathbf{X}_{v_o}, \mathbf{U}_v, \theta_f).$$

θ_f is a vector of parameters, being θ_f independent of node v.[1] In other words, the computation of the state at each node is carried out according to a bottom-up strategy, i.e., $\mathbf{X}_v$ is calculated only when all the states of the children of v are available. When the ith child does not exist (v_i = null), $\mathbf{X}_{v_i}$ is assigned with a predefined state $\mathbf{X}_{null}$.

At the supersource also an *output* function is evaluated, by a feedforward network called the *output* network

$$\mathbf{Y}_s = g(\mathbf{X}_s, \theta_g).$$

The parametric representations of f and g can be implemented by a variety of neural-network models. In the case of a three-layered perceptron $\mathcal{N}$, with sigmoidal activation functions in the hidden units and linear activation functions in the output units, f is the network input–output function $f_{\mathcal{N}}$.

[1]The weight vector is the same at each node; in this case, we say that the recursive network is *stationary* [1].

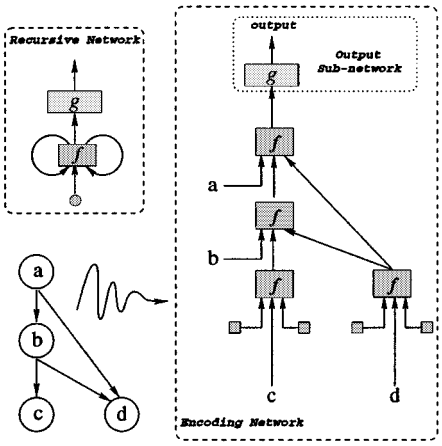

Fig. 1. The *encoding* and the *output* networks associated to a DPAG. The *recursive neural network* is unfolded through the structure of the DPAG.

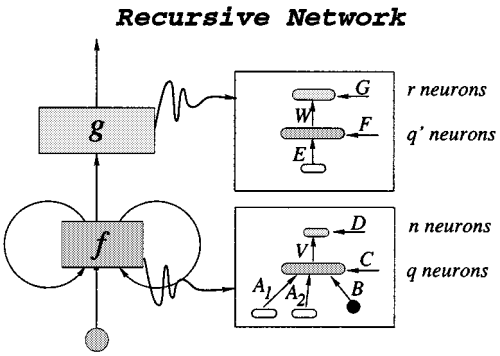

Fig. 2. A three-layered perceptron implementation of the recursive architecture shown in Fig. 1. Hidden layers are sigmoidal; instead, output layers are linear in both the feedforward networks.

Thus, the dependence of node v on its children and on its label is expressed by the following relationship:

$$\begin{aligned}\mathbf{X}_v =& f_{\mathcal{N}}(\mathbf{X}_{v_1}, \ldots, \mathbf{X}_{v_o}, \mathbf{U}_v, \theta_f) \\ =& \mathbf{V} \cdot \vec{\sigma}\left(\sum_{k=1}^{o} \mathbf{A}_k \cdot \mathbf{X}_{v_k} + \mathbf{B} \cdot \mathbf{U}_v + \mathbf{C}\right) + \mathbf{D} \end{aligned} \quad (1)$$

where $\vec{\sigma}$ is a vectorial sigmoidal function and the network weights θ_f collect $\mathbf{A}_k \in \mathbb{R}^{q,n}$, $k = 1, \ldots, o$, $\mathbf{B} \in \mathbb{R}^{q,m}$, $\mathbf{C} \in \mathbb{R}^q$, $\mathbf{D} \in \mathbb{R}^n$ and $\mathbf{V} \in \mathbb{R}^{n,q}$. Here, m is the dimension of the label space, n the dimension of the state space and q represents the number of hidden neurons (see Fig. 2). A similar equation holds for g

$$\mathbf{Y}_s = g_{\mathcal{N}}(\mathbf{X}_s, \theta_g) = \mathbf{W} \cdot \vec{\sigma}(\mathbf{E} \cdot \mathbf{X}_s + \mathbf{F}) + \mathbf{G}$$

where θ_g collects $\mathbf{E} \in \mathbb{R}^{q',n}$, $\mathbf{F} \in \mathbb{R}^{q'}$, $\mathbf{G} \in \mathbb{R}^r$ and $\mathbf{W} \in \mathbb{R}^{r,q'}$ (see Fig. 2).

Thus, a recursive neural network implements a function h : DPAGs $\rightarrow \mathbb{R}^r$, where $h(G) = \mathbf{Y}_s$. Formally, $h = g \circ \tilde{f}$, where $\tilde{f}(G) = \mathbf{X}_s$ denotes the process that takes a graph and returns the state at the supersource. In [18]–[20], recursive neural networks are proved to be able to approximate, in probability, any function on trees. More precisely, given a set $T \subset$ DPAGs of positional trees,[2] a function $t : T \rightarrow \mathbb{R}^r$, a probability measure

[2]A positional tree is a DPAG where each node has only one parent.

P on T and any real ε, there is a function h, realized by a recursive neural network, such that $P(|h(G) - t(G)| \geq \varepsilon) \leq \varepsilon$. The above result completely characterizes also the approximation capabilities of recursive neural networks with regard to the functions on DPAGs [21].

IV. Recursive Neural Networks for Processing DAGs

Recursive neural networks have been designed to deal directly with DPAGs. Function f naturally considers the position of each child of a node, because the child state has a particular position in the input of f. A simple approach allows us to process also DAGs: each input DAG $(V, A, \mathcal{L})$ is mapped into a DPAG $(V, A, \mathcal{L}, O)$, which is processed instead of the original DAG. The transformation only requires us to assign a position to the children of each node. However, such an assignment, which is arbitrary, affects the result of the computation.

In order to avoid this situation, f must produce the same result despite of the position assigned to the children. Formally, f must satisfy

$$f(\mathbf{X}_1, \ldots, \mathbf{X}_o, \mathbf{U}, \theta_f) = f(\mathbf{X}_{\pi(1)}, \ldots, \mathbf{X}_{\pi(o)}, \mathbf{U}, \theta_f) \quad (2)$$

for any permutation π and any $\mathbf{X}_1, \ldots, \mathbf{X}_o \in \mathbb{R}^n$, $\mathbf{U} \in \mathbb{R}^m$, θ_f. Notice that, transforming each DAG $(V, A, \mathcal{L})$ into all the DPAGs $(V, A, \mathcal{L}, O_1)$, $(V, A, \mathcal{L}, O_2)$, $\ldots$ $(V, A, \mathcal{L}, O_d)$ that differ only for the ordering of the children, the training algorithm may produce a function f that satisfies (2). However, in practice, this requires a huge learning set and a long training time. Alternatively, one can implement f by a model that naturally satisfies (2), i.e., a model that can realize exclusively all the functions defined in (2). In the following, we show how a three-layered neural network, appropriately constrained, can be used for this purpose.

A. A Network Architecture for Processing DAGs

The idea we adopt is similar to the one proposed by LeCun [22] for convolutional neural networks. Convolutional neural networks are used in image processing for their ability to be insensitive to image translations.

Example 1: Let us consider a three-layered network with two input units, $2q$ hidden units and one output unit (Fig. 3). The ith hidden neuron contributes to the output of the network by a value $v_i\sigma(a_{i,1}x_1 + a_{i,2}x_2 + c_i)$, where x_1, x_2 are the inputs and $a_{i,1}, a_{i,2}, c_i, v_i$ are the network parameters. Suppose that, for the hidden unit j, the input-to-hidden layer weights are exchanged $(a_{j,1} = a_{i,2}, a_{j,2} = a_{i,1})$, whereas other weights are the same $(c_j = c_i, v_j = v_i)$. Therefore, the contribution to the network output due to i and j is

$$h_i(x_1, x_2) = v_i\sigma(a_{i,1}x_1 + a_{i,2}x_2 + c_i) + v_i\sigma(a_{i,2}x_1 + a_{i,1}x_2 + c_i)$$

where $h_i(x_1, x_2) = h_i(x_2, x_1)$, which is a special case of constraint (2). If such a behavior is shared by q pairs of particularly selected hidden units, the whole output of the network can be calculated as

$$f(x_1, x_2) = \frac{1}{2}\sum_{i=1}^{2q} h_i(x_1, x_2) \quad (3)$$

where f fulfills the constraint (2).

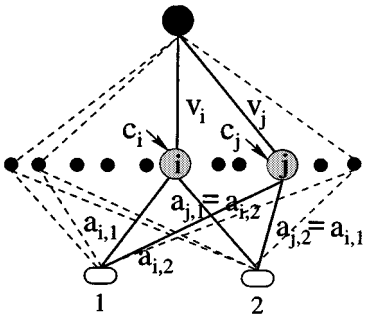

Fig. 3. A neural network with shared weights.

The arguments of Example 1 can be easily applied to more general architectures, i.e., to networks with many inputs and many outputs. In fact, in order to implement a function $f(x_1, \ldots, x_n)$ invariant to the input permutations, the hidden layer of the network should consist of sets of units sharing the input-to-hidden weights: each set should contain a unit for each permutation of the weights, that is a neuron i with weights $(a_{i_1}, a_{i_2}, a_{i_3}, \ldots)$, a neuron j with weights $(a_{j_1} = a_{i_2}, a_{j_2} = a_{i_1}, a_{j_3} = a_{i_3}, \ldots)$, a neuron k with $(a_{k_1} = a_{i_1}, a_{k_2} = a_{i_3}, a_{k_3} = a_{i_2}, \ldots)$ and so on. Furthermore, (2) provides a different constraint with regard to the insensibility to the permutations of all the inputs, since f must remain unchanged only when the states $\mathbf{X}_{v_k}$ are permuted. However, the rationale carried out in the following, in order to build up a network that fulfills (2), is essentially the same as that outlined so far.

Let $\mathcal{P} = \{\pi_1, \ldots, \pi_p\}$, $p = o!$, be the set of all the permutation functions on $\{1, \ldots, o\}$. The network we consider, $\mathcal{SN}$, has the same parameters $(\theta_f = (\mathbf{A}_1, \ldots, \mathbf{A}_o, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{V}))$ as the network $\mathcal{N}$ in (1), but those parameters are shared among a number of connections. In fact, the network has qp hidden units. The hidden-to-output connection weights are $\overline{\mathbf{V}} \in \mathbb{R}^{n,qp}$, the input-to-hidden connection weights are $\overline{\mathbf{A}} \in \mathbb{R}^{qp,no+m}$, the hidden thresholds are $\overline{\mathbf{C}} \in \mathbb{R}^{qp}$ and the output threshold are $\overline{\mathbf{D}} \in \mathbb{R}^n$, where

$$\begin{aligned}
\overline{\mathbf{V}} &= (\mathbf{V}, \ldots, \mathbf{V}) \\
\overline{\mathbf{A}} &= \begin{pmatrix} \mathbf{A}_{\pi_1(1)} & \cdots & \mathbf{A}_{\pi_1(o)} & \mathbf{B} \\ \vdots & & \vdots & \vdots \\ \mathbf{A}_{\pi_p(1)} & \cdots & \mathbf{A}_{\pi_p(o)} & \mathbf{B} \end{pmatrix}. \\
\overline{\mathbf{C}} &= (\mathbf{C}, \ldots, \mathbf{C})' \\
\overline{\mathbf{D}} &= p\mathbf{D}.
\end{aligned} \quad (4)$$

Moreover, let $\mathbf{I}_v = (\mathbf{X}'_{v_1}, \ldots, \mathbf{X}'_{v_o}, \mathbf{U}'_v)'$, where " $'$ " is the transpose operator. Intuitively, for each hidden unit in $\mathcal{N}$, there are p units in $\mathcal{SN}$ sharing the same parameters. It is easy to verify that the output of $\mathcal{SN}$ is

$$\begin{aligned}
&f_{\mathcal{SN}}(\mathbf{X}_{v_1}, \ldots, \mathbf{X}_{v_o}, \mathbf{U}_v, \theta_f) \\
&= \overline{\mathbf{V}} \cdot \overline{\sigma}\left(\overline{\mathbf{A}} \cdot \mathbf{I}_v + \overline{\mathbf{C}}\right) + \overline{\mathbf{D}} \\
&= \sum_{i=1}^{p}\left[\mathbf{V} \cdot \vec{\sigma}\left(\sum_{k=1}^{o} \mathbf{A}_{\pi_i(k)} \cdot \mathbf{X}_{v_k} + \mathbf{B} \cdot \mathbf{U}_v + \mathbf{C}\right) + \mathbf{D}\right] \\
&= \sum_{i=1}^{p} f_{\mathcal{N}}(\mathbf{X}_{\pi_i(v_1)}, \ldots, \mathbf{X}_{\pi_i(v_o)}, \mathbf{U}_v, \theta_f).
\end{aligned} \quad (5)$$

Remark 1: Learning in recursive neural networks with shared weights can be carried out via standard backpropagation through structure [3], [23] so as in standard (not with shared weights) architectures. The equality constraint is guaranteed during training simply due to the initial equality condition. In fact, the hidden-to-output weights are updated by the same quantities, also implying the equality of the backpropagated error contribution on the input-to-hidden weights.

B. Theoretical Results

It is a crucial matter to study the computational capabilities of the model we have proposed. In order to give to this problem a formal setup, let $\mathcal{CF}(K)$ be the set of the continuous functions $f : \mathbb{R}^{no+m} \rightarrow \mathbb{R}^n$ that fulfill (2) and are defined on a compact set $K \subset \mathbb{R}^{no+m}$. Moreover, suppose that $\mathcal{CF}(K)$ is equipped with the supremum norm $L_\infty(K)$. The following theorem proves that the network $\mathcal{SN}$ always fulfills our initial project. Proofs of the theorems are collected in the Appendix.

Theorem 1: Any network $\mathcal{SN}$ satisfies (2), i.e., $f_{\mathcal{SN}} \in \mathcal{CF}(K)$ for any K.

On the other hand, the particular weight sharing schema that constrains $\mathcal{SN}$ does not limit the computational capabilities of the network. In fact, the following theorem shows that our model can approximate any function in $\mathcal{CF}(K)$.

Theorem 2: For any compact set K, any function $l \in \mathcal{CF}(K)$ and any precision $\varepsilon > 0$, there is a network $\mathcal{SN}$, whose input–output function $f_{\mathcal{SN}}$ is such that

$$\|l - f_{\mathcal{SN}}\|_{L_\infty(K)} \leq \varepsilon.$$

Theorem 2 can be extended to other sets of functions and networks with different activation functions, for which the universal approximation property has been proved [24]. The proof of Theorem 2 still works, for instance, if $\mathcal{CF}(K)$ is replaced by the set $\mathcal{F}^p(\mu)$ of the functions that fulfill (2) and have a finite measure with regard to the norm

$$\|f\|_{L^p(\mu)} = \left(\int_{\mathbb{R}^{no+m}} |f(x)|^p d\mu(x) \right)^{1/p}.$$

Theorem 2 allows us to extend the results on approximation capabilities of recursive neural networks from DPAGs to DAGs. Furthermore, since it can be easily proved that for each DPAG/DAG an "*output-equivalent*" (eventually not ordered) tree exists [21], for which a recursive architecture produces exactly the same output as for the corresponding graph,[3] from then on we restrict ourselves to trees. Obviously, in this particular case, the output-equivalent tree to a certain DAG is nonordered. In fact, the following theorem proves that the proposed model has the same computational capabilities of recursive neural networks, with the only difference that, of course, the domain consists of nonordered, instead of ordered, trees.

[3]Starting from a graph, the output-equivalent tree is constructed inserting multiple copies of each node having many parents (one for each parent). On the other hand, the original graph can be directly reconstructed from the output-equivalent tree by collapsing together the duplicates of multiparent nodes. The way in which the recursive processing is carried out implies that the network will calculate exactly the same state at the supersource for output-equivalent structures.

Theorem 3: Given a set of trees $T \subset$ DAGs, a function $t : T \rightarrow \mathbb{R}^r$, a probability measure P on T and any real ε, there is a function $h_{\mathcal{SN}}$, realized by a recursive neural network $\mathcal{SN}$, with shared weights, such that $P(|h_{\mathcal{SN}}(G) - t(G)| \geq \varepsilon) \leq \varepsilon$.

Notice that, according to our notation, graphs can have only a finite set of labels. However, Theorem 2 can be easily extended to the case of rational and even real labels. Moreover, the approximation with regard to a probability measure can be replaced by the approximation with regard to the superior norm, but in this case the height of trees must be bounded and the label set must be finite.

Finally, in [19] and [20] a bound is provided on the number of neurons needed to realize the transition network f. In fact, it is proved that a network, which implements f, can be built with $\mathcal{O}(\log_2 o)$ layers having $\mathcal{O}(o)$ neurons per layer. In order to extend those results to our case, let $\mathcal{MSN}$ be any composition of a three-layered network with shared weights $\mathcal{SN}$ and a multilayer network $\mathcal{N}$ with s layers. The network $\mathcal{MSN}$ extends our architecture to the case of many hidden layers, since $\mathcal{MSN}$ has $s+3$ layers and implements a function $f_{\mathcal{MSN}} = f_{\mathcal{N}} \circ f_{\mathcal{SN}}$. It is straightforward that $f_{\mathcal{MSN}} \in \mathcal{CF}(K)$ holds. Using the new architecture in the proof of Theorem 3 and the results established in [19] and [20], it follows that the encoding network can be realized using $\mathcal{O}((o!)\, o \log_2 o)$ hidden neurons.

C. Some Remarks

The hidden units of $\mathcal{SN}$ are $qp = qo!$. Thus, this method cannot be applied to DAGs with a large outdegree o, due to the factorial growth of the number of the hidden units. This may be an important limit for some applications, but in many other cases it is possible to build a domain that contains graphs with a small outdegree. On the other hand, the number of hidden units of $\mathcal{SN}$ depends linearly on q, which means that the dimension of the states and the dimension of the labels affect $\mathcal{SN}$ as they affect $\mathcal{N}$.

Moreover, the difficulty in processing DAGs with a large number of children is a characteristic of the recursive model and cannot be overcame easily. When $\mathcal{N}$ is used in place of $\mathcal{SN}$, $\mathcal{N}$ has to learn constraint (2) from examples. This necessarily requires a number of examples that grows factorially with o, such that the computational burden due to the dimension of $\mathcal{SN}$ is transformed into a computational burden due to the number of examples.

On the other hand, the transition network $\mathcal{SN}$ cannot be replaced by a network $\mathcal{N}$ which exactly realizes the same function with a smaller number of hidden neurons. In fact, the weights of a feedforward neural network are uniquely identifiable by the network input–output function [25], thus $f_{\mathcal{SN}} = f_{\mathcal{N}}$ implies $\mathcal{SN} = \mathcal{N}$. Therefore, the above result does not formally prove that the approximation of a generic function on DAGs needs a factorial number of units, because it regards only the transition function $f_{\mathcal{SN}}$, instead of the whole network function $h_{\mathcal{SN}}$; moreover, it deals with the exact realization, instead of the approximation, of functions in $\mathcal{CF}(K)$. However, the proposed result gives a suggestion on the difficulty of realizing permutation invariance in the sense of Theorems 1 and 2.

Finally, the role of input preprocessing must be highlighted. Often, real-world information can be represented in more than

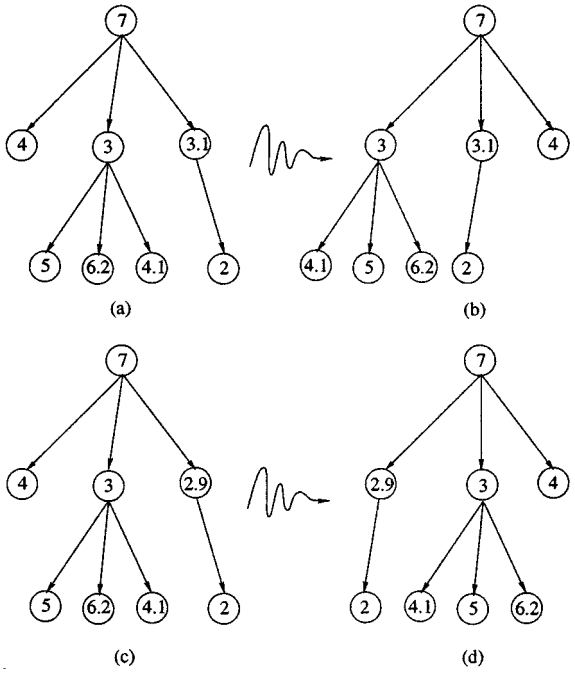

Fig. 4. Graphs (b) and (d) are the normalization of (a) and (c), respectively. Graph (c) is a noisy version of (a).

one graphical form. Thus, choosing the correct way of representing the input data will deeply affect the learning problem. In fact, if all the graphs belonging to the domain are distinct up to any rearrangement of the children of each node, network $\mathcal{N}$ is theoretically sufficient to learn the data (as each DAG corresponds to a single DPAG in the domain). Therefore, when a preprocessing algorithm exists that represents the input information through distinct graphical structures, the learning problem on network $\mathcal{N}$ is well posed.

Such a condition is easily achieved by a normalization of the graphs in the domain. For example, the normalization process can be carried out by ordering the children of each node according to their labels[4] (see Fig. 4). However, the domain normalization is a solution that is not equivalent to the use of our model. In fact, a small perturbation of inputs may induce the normalization process to produce different graphs for very similar data [in Fig. 4, the noisy version (c) of graph (a) produces (d) instead of (b)], which causes network $\mathcal{N}$ to return completely different outputs. In many applications, this is not the desired behavior because similar outputs are expected in correspondence of similar inputs. This particularly happens in domains where there are noisy data.

From this point of view, our strategy is similar to the one adopted by LeCun [17] to process handwritten digits, where a neural network is fed directly on nonnormalized input data. In fact, the normalization of the digits, which is commonly adopted in other methods, is replaced in [17] with the use of the convolutional neural networks which are insensitive to translations of the input image. In [22], using experimental arguments, it is proved that convolutional neural networks outperform other approaches based on linear classifiers, multilayer perceptrons, support vector machines, principal component analysis and k-nearest neighbor classifiers.

[4]The lexicographic order can be used for labels. Given two labels $u_1, u_2 \in \mathbb{R}^l$, $u_1 < u_2$ if all the components (of the labels) up to the kth one are equal, while $u_{1,k+1} < u_{2,k+1}$.

V. Conclusion

In this paper, we present a new recursive network model that allows us to process DAGs. The feedforward neural network that realizes the transition function has a special architecture with shared weights, which yields the independence with respect to the ordering of the children. Moreover, it was proved that the model can approximate any function on DAGs in probability up to any degree of precision.

Future interesting research topics include the study of a possible extension of our theoretical results to convolutional neural networks [22], due to the similarity in the approach adopted to make the network insensitive to the input rearrangements. Moreover, being the encoding network an architecture with shared weights, it would be very interesting to define a common theory on approximation properties for such architectures, that covers also recursive neural networks. Finally, whereas our architecture requires a factorial number of neurons, it is an open question whether this bound is not only sufficient but also necessary to approximate a generic permutation invariant function. In general, the answer to this problem depends on the activation function adopted in the network and on the norm used to estimate the the approximation accuracy.

Appendix

A. Proof of Theorem 1

Without loss of generality, suppose that the permutation π in (2) corresponds to $\pi_j \in \mathcal{P}$. By the substitution $k = \pi_j^{-1}(z)$, we immediately get

$$\begin{aligned} &f_{\mathcal{SN}}\left(\mathbf{X}_{v_{\pi_j(1)}}, \ldots, \mathbf{X}_{v_{\pi_j(o)}}, \mathbf{U}_v, \theta_f\right) \\ &= \sum_{i=1}^{p}\left[\mathbf{V}\cdot\vec{\sigma}\left(\sum_{k=1}^{o}\mathbf{A}_{\pi_i(k)}\cdot\mathbf{X}_{v_{\pi_j(k)}} + \mathbf{B}\cdot\mathbf{U}_v + \mathbf{C}\right) + \mathbf{D}\right] \\ &= \sum_{i=1}^{p}\left[\mathbf{V}\cdot\vec{\sigma}\left(\sum_{z=1}^{o}\mathbf{A}_{\pi_i(\pi_j^{-1}(z))}\cdot\mathbf{X}_{v_z} + \mathbf{B}\cdot\mathbf{U}_v + \mathbf{C}\right) + \mathbf{D}\right]. \end{aligned}$$

Observing that $\mathcal{P} = \{\pi_i \circ \pi_j^{-1} |\ i = 1, \ldots, o\}$ holds, the sum can be rewritten as

$$\begin{aligned} &\sum_{i=1}^{p}\left[\mathbf{V}\cdot\vec{\sigma}\left(\sum_{z=1}^{o}\mathbf{A}_{\pi_i(z)}\cdot\mathbf{X}_{v_z} + \mathbf{B}\cdot\mathbf{U}_v + \mathbf{C}\right) + \mathbf{D}\right] \\ &= f_{\mathcal{SN}}(\mathbf{X}_{v_1}, \ldots, \mathbf{X}_{v_o}, \mathbf{U}_v, \theta_f) \end{aligned}$$

which yields the thesis. □

B. Proof of Theorem 2

According to the well known results on the approximation properties of feedforward neural networks (see, for example, [24]), there is a common (i.e., without shared weights) three-layered network $\mathcal{N}$ such that

$$\left|\frac{l(x)}{p} - f_{\mathcal{N}}(x)\right| \leq \frac{\varepsilon}{p} \tag{6}$$

for each $x \in K$. Let us define a network $\mathcal{SN}$, with shared weights, starting from $\mathcal{N}$ and using the same procedure as in (4). Because of (5) and (6), for each $x \in K$ and any permutation $\pi_i \in \mathcal{P}$

$$\begin{aligned}
&|l(x) - f_{\mathcal{SN}}(x)| \\
&= \Big| l(\mathbf{X}_{v_1}, \ldots, \mathbf{X}_{v_o}, \mathbf{U}_v, \theta_f) \\
&\quad - \sum_{i=1}^{p} f_{\mathcal{N}}(\mathbf{X}_{\pi_i(v_1)}, \ldots, \mathbf{X}_{\pi_i(v_o)}, \mathbf{U}_v, \theta_f) \Big| \\
&\leq \sum_{i=1}^{p} \Big| \frac{l(\mathbf{X}_{v_1}, \ldots, \mathbf{X}_{v_o}, \mathbf{U}_v, \theta_f)}{p} \\
&\quad - f_{\mathcal{N}}(\mathbf{X}_{\pi_i(v_1)}, \ldots, \mathbf{X}_{\pi_i(v_o)}, \mathbf{U}_v, \theta_f) \Big| \\
&= \sum_{i=1}^{p} \Big| \frac{l(\mathbf{X}_{\pi_i(v_1)}, \ldots, \mathbf{X}_{\pi_i(v_o)}, \mathbf{U}_v, \theta_f)}{p} \\
&\quad - f_{\mathcal{N}}(\mathbf{X}_{\pi_i(v_1)}, \ldots, \mathbf{X}_{\pi_i(v_o)}, \mathbf{U}_v, \theta_f) \Big| \leq \varepsilon
\end{aligned}$$

being $l \in \mathcal{CF}(K)$. □

C. Proof of Theorem 3

In [18]–[20] the theorem was proved for $T \subset$ DPAGs. The proof proposed in [18]–[20] follows the rationale which can be summarized in the following five steps.[5]

1) *There exists a continuous function f[6] such that $\tilde{f}$ is injective on T.*
2) *Let T have a finite number of elements. If $\tilde{f}$ is injective on T, then there exists a continuous function g such that $t = g \circ \tilde{f}$ on T.*

   (This assertion has a simple intuitive explanation, since when $\tilde{f}$ is injective, the recursive processing produces a different state at the supersource for each tree. Moreover, since T is finite, function g has to solve a simple interpolation problem $g(\tilde{f}(G)) = t(G),\ G \in T$.)
3) *Let T have an infinite number of elements. If $\tilde{f}$ is injective on T then, for any probability measure P and any positive real number ε, there exists a continuous function g such that $P(|t(G) - (g \circ \tilde{f})(G)| > \varepsilon) < \varepsilon$ on T.*

   (By point 2, the assertion holds for a finite subset $T' \subseteq T$ such that $P(T') > 1 - \varepsilon$; then the function $g \circ \tilde{f}$ defined in point 2 obviously fulfills the statement.)
4) *Let f_α and g_α be functions that converge uniformly to f and g, respectively, for $\alpha \to 0$; then, there exists α such that $P(|t(G) - (g_\alpha \circ \tilde{f}_\alpha)(G)| > \varepsilon) < \varepsilon$ on T.*

   (The proof is consequential to the selection of an α so small that $|t(G) - (g_\alpha \circ \tilde{f}_\alpha)(G)| \leq \varepsilon$ holds for all the $G \in T'$, where T' is the finite set of point 3.)
5) *Two feedforward networks are built that realize f_α and g_α.*

[5]In [18], steps 1) and 2) are proved in the discussion of the first lemma of p. 3, step 5) is in the second lemma of p. 3, step 3) is in the theorem of p. 2. Notice that whereas the five steps are more evident in [18], the proofs in [19], [20] are more detailed.

[6]Notice that here f is a generic function not constrained to be realizable by a neural network.

In order to prove the theorem, we have to extend the above reasoning to the case of DAGs. However, steps 2–4 are not affected by the replacement of DPAGs with DAGs and the proof in [18] and [19] still works without any change. Thus, it remains to prove that assertions 1 and 5 still hold. In the following, a constructive procedure to realize a function f' such that the corresponding $\tilde{f}'$ is injective on DAGs is shown (step 1).

Let f and $\tilde{f}$ be defined as in point 1 for the case of DPAGs. We will adopt a total order $\prec_{f,v}$ on the states computed by f at each node v: $\mathbf{X}_{v_h} \prec_{f,v} \mathbf{X}_{v_k}$ if $a_i < b_i$, where a_i and b_i are the ith components of $\mathbf{X}_{v_h}$ and $\mathbf{X}_{v_k}$, respectively, and i is the first index for which $a_i \neq b_i$. Let $\pi_{f,v}$ be a permutation on $\{1, 2, \ldots, o\}$ that rearranges the states of the children of v in an increasing order, i.e., $\mathbf{X}_{v_{\pi_{f,v}(1)}} \prec_{f,v} \mathbf{X}_{v_{\pi_{f,v}(2)}} \cdots \prec_{f,v} \mathbf{X}_{v_{\pi_{f,v}(o)}}$. Finally, let us denote with $=_{\text{DPAGs}}$ and $=_{\text{DAGs}}$ the equality relationships on DPAGs and DAGs, respectively.

We claim that the following function f' fulfills our purpose

$$f'(\mathbf{X}_{v_1}, \ldots, \mathbf{X}_{v_o}, \mathbf{U}_v) = f(\mathbf{X}_{v_{\pi_{f,v}(1)}}, \ldots, \mathbf{X}_{v_{\pi_{f,v}(o)}}, \mathbf{U}_v)$$

In fact, notice that $\tilde{f}'$ produces the same result as $\tilde{f}$ except for an eventual rearrangement of the children in each node. Thus, for each tree G, a tree $\overline{G}$ exists such that $G =_{\text{DAGs}} \overline{G}$ and $\tilde{f}'(G) = \tilde{f}(\overline{G})$. Since $\tilde{f}$ is injective on DPAGs, $G_1 \neq_{\text{DAGs}} G_2$ implies $\tilde{f}'(G_1) = \tilde{f}(\overline{G}_1) \neq \tilde{f}(\overline{G}_2) = \tilde{f}'(G_2)$, i.e., $\tilde{f}'$ is injective on DAGs, which proves step 1.

Finally, with regard to step 5, notice that we can use the same g as for DPAGs, thus g_α can be realized by the same neural network as in [19]. On the other hand, the existence of a network that approximates f' is proved by Theorem 2, since the definition of f' straightforwardly implies $f' \in \mathcal{CF}(K)$. □

ACKNOWLEDGMENT

The authors gratefully thank Prof. A. Sperduti (University of Pisa) for his very useful comments and suggestions.

REFERENCES

[1] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE Trans. Neural Networks*, vol. 9, pp. 768–786, Sept. 1998.

[2] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Trans. Neural Networks*, vol. 8, pp. 429–459, 1997.

[3] A. Küchler and C. Goller, "Inductive learning in symbolic domains using structure-driven recurrent neural networks," in *Advances in Artificial Intelligence*, G. Görz and S. Hölldobler, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 183–197.

[4] C. Goller, "A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems," Ph.D. dissertation, Technische Universität, München, Germany, 1997.

[5] A. Bianucci, A. Micheli, A. Sperduti, and A. Starita, "Analysis of the internal representation developed by neural networks for structures applied to QSAR studies of benzodiazepines," J. Chem. Inform. Comput. Sci..

[6] M. Gori, M. Maggini, E. Martinelli, and F. Scarselli, "Learning user profiles in NAUTILUS," in *Proc. Int. Conf. Adaptive Hypermedia Adaptive Web-Based Syst.—Lecture Notes Comput. Sci., 1892*, Trento, Italy, Aug. 2000.

[7] R. Lenz, "Group theoretical transforms in image processing," *Current Topics Pattern Recognition Res.*, vol. 1, pp. 83–106, 1994.

[8] C. W. Curtis and L. Reiner, *Representation Theory of Finite Groups and Associative Algebras*. New York: Wiley, 1988.

[9] W. Fulton and J. Harris, "Representation theory," in *Graduate Texts in Mathematics*. New York: Springer-Verlag, 1991.

[10] J. L. Mundy, A. Zisserman, and D. Forsyth, *Applications of Invariance in Computer Vision*. New York: Springer-Verlag, 1994, vol. 825, Lecture Notes in Computer Science.
[11] W. Schempp, "Harmonic analysis on the Heisenberg nilpotent Lie group, with applications to signal theory," in *Pitman Research Notes in Mathematics*. Harlow, Essex, U.K.: Longman , 1986.
[12] R. Lenz, "Optimal filters for the detection of linear patterns in 2–D and higher dimensional images," *Pattern Recognition*, vol. 20, no. 2, pp. 163–172, 1987.
[13] ——, "A group theoretical approach to filter design," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1989.
[14] J. Lahtinen, T. Martinsen, and J. Lampinen, "Improved rotational invariance for statistical inverse in electrical impendance tomography," in *Proc. IJCNN'2000*, vol. II, Como, Italy, 2000, pp. 154–158.
[15] H. P. Chiu and D. C. Tseng, "Invariant handwritten Chinese character recognition using fuzzy min-max neural networks," *Pattern Recognition Lett.*, vol. 18, no. 5, pp. 481–491, 1997.
[16] E. Nordström, O. Gällmo, L. Asplund, M. Gustafsson, and B. Eriksson, "Neural networks for admission control in an ATM network," in *Connectionism in a Broad Perspective: Selected Papers From the Swedish Conference on Connectionism–1992*, L. F. Niklasson and M. B. Bodén, Eds: Ellis Horwood, 1994, pp. 239–250.
[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, pp. 541–551, 1989.
[18] B. Hammer and V. Sperschneider, "Neural networks can approximate mappings on structured objects," in *Int. Conf. Comput. Intell. Neural Networks'97*, P. P. Wang, Ed., 1997, pp. 211–214.
[19] B. Hammer, "Learning with Recurrent Neural Networks," Ph.D. dissertation, Fachbereich Mathematik/Informatik—Universität Osnabrück, 1999.
[20] ——, "Approximation capabilities of folding networks," in *ESANN '99*, Bruges, Belgium, April 1999, pp. 33–38.
[21] M. Bianchini, M. Gori, and F. Scarselli, "Theoretical Properties of Recursive Networks With Linear Neurons," *IEEE Trans. Neural Networks*, vol. 12, pp. 953–967, Sept. 2001.
[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[23] A. Küchler, "Adaptive Processing of Structured Data: From Sequences to Trees and Beyond," Ph.D. dissertation, Faculty Comput. Sci., Univ. Ulm, Germany, 1999.
[24] F. Scarselli and A. C. Tsoi, "Universal approximation using feedforward neural networks: A survey of some existing methods and some new results," *Neural Networks*, vol. 11, no. 1, pp. 15–37, 1998.
[25] F. Albertini and E. D. Sontag, "For neural networks, function determines form," *Neural Networks*, vol. 6, pp. 975–990, 1992.