

NC ms 2933

# Universal Approximation Capability of Cascade Correlation for Structures\*

Barbara Hammer<sup>†</sup>, Alessio Micheli<sup>‡</sup> and Alessandro Sperduti<sup>§</sup>

July 21, 2004

## Abstract

Cascade correlation (CC) constitutes a training method for neural networks which determines the weights as well as the neural architecture during training. Various extensions of CC to structured data have been proposed: recurrent cascade correlation (RCC) for sequences, recursive cascade correlation (RecCC) for tree structures with limited

---

\*This work has been partially supported by MIUR grant 2002093941\_004. We would like to thank two anonymous referees for profound and valuable comments on an earlier version of the manuscript.

<sup>†</sup>Department of Mathematics/Computer Science, University of Osnabrück, D-49069 Osnabrück, Germany, e-mail: hammer@informatik.uni-osnabrueck.de

<sup>‡</sup>Dipartimento di Informatica, Università di Pisa, Pisa, Italia

<sup>§</sup>Dipartimento di Matematica Pura ed Applicata, Università di Padova, Padova, Italia

fan-out, and contextual recursive cascade correlation (CRecCC) for rooted directed positional acyclic graphs (DPAGs) with limited fan-in and fan-out. We show that these models possess the universal approximation property in the following sense: given a probability measure  $P$  on the input set, every measurable function from sequences into a real vector space can be approximated by a sigmoidal RCC up to any desired degree of accuracy up to inputs of arbitrary small probability. Every measurable function from tree structures with limited fan-out into a real vector space can be approximated by a sigmoidal RecCC with multiplicative neurons up to any desired degree of accuracy up to inputs of arbitrary small probability. For sigmoidal CRecCC networks with multiplicative neurons, we show the universal approximation capability for functions on an important subset of all DPAGs with limited fan-in and fan-out for which a specific linear representation yields unique codes. We give one sufficient structural condition for the latter property which can easily be tested: the enumeration of ingoing and outgoing edges should be compatible. This property can be fulfilled for every DPAG with fan-in and fan-out two via reenumeration of children and parents, and for larger fan-in and fan-out via an expansion of the fan-in and fan-out and reenumeration of children and parents. In addition, the result can be generalized to the case of IO-isomorphic transductions of structures. Thus, CRecCC networks constitute the first neural models for which the universal approximation capability of functions involving fairly general acyclic graph structures is proved.

# 1 Introduction

Pattern recognition methods such as neural networks or support vector machines constitute powerful machine learning tools in a widespread area of applications. Usually, they process data in the form of real vectors of fixed and finite dimensionality. Canonical extensions of these basic models to sequential data which occur in language processing, bioinformatics, or time-series prediction, for example, are provided by recurrent networks or statistical counterparts (Bengio and Frasconi, 1996; Kremer, 2001; Sun, 2001). Recently, an increasing interest in the possibility to deal also with more complex non-standard data has been observed and several models have been proposed within this topic: specifically designed kernels such as string and tree kernels or kernels derived from statistical models constitute a canonical interface for kernel methods for structures (Haussler, 1999; Jaakkola, Diekhans, and Haussler, 2000; Leslie, Eskin, and Noble, 2002; Lodhi et al., 2000; Watkins, 1999). Alternatively, recurrent neural models can be extended to complex dependencies which occur in tree structures, graphs, or spatial data (Baldi et al., 1999; Frasconi, Gori, and Sperduti, 1998; Goller and Küchler, 1996; Hammer, 2000; Micheli, Sona, and Sperduti, 2000; Pollastri et al., 2002; Sperduti, Majidi, and Starita, 1996; Sperduti and Starita, 1997; Wakuya and Zurada, 2001). These structure processing models have the advantage that complex data such as spatial data, tree structures, or graphs can be directly tackled and thus a possibly time consuming and usually not information preserving encoding of the structures by a finite number of real-valued features can be avoided. Successful applications of these models have been reported in different areas such as image and document processing, logic, natural language processing, chemistry, DNA processing, homology detection, or protein structure prediction (Baldi et al., 1999; Diligenti, Frasconi, and Gori, 2003; Goller, 1997;

Jaakkola, Diekhans, and Haussler, 2000; Leslie, Eskin, and Noble, 2002; Lodhi et al., 2000; deMauro et al., 2003; Pollastri et al., 2002; Sturt et al., 2003; Vullo and Frasconi, 2003). Here, we are interested in a variant of so-called recursive networks which constitute a straightforward generalization of well known recurrent networks to tree structures and for which excellent results also for large learning tasks have been reported (Frasconi, 2002).

Training recurrent neural networks faces severe problems such as the problem of long-term dependencies, and the design of efficient training algorithms is still a challenging problem of ongoing research (Bengio, Simard, and Frasconi, 1994; Hammer and Steil, 2002). Since the number of potentially different structures increases exponentially with the size of the data, the space is often only sparsely covered by a given training set. In addition, the generalization ability might be fundamentally worse compared to simple pattern recognition tools because the capacity of the models (measured e.g. in terms of the VC dimension) depends on the structures (Hammer, 2001). For recursive models for structures, the long-term dependencies problem is often reduced because structural representations via trees or graphs yield more compact representation of data. However, the other problems remain. Thus the design of efficient training algorithms which yield sparse models with good generalization ability is a key issue for recursive models for structures.

Cascade correlation (CC) has been proposed by (Fahlmann and Lebiere, 1990) as a particularly efficient training algorithm for feedforward networks which simultaneously determines the architecture of the network and the parameters. Hidden neurons are created and trained consecutively, so to maximize the correlation of the hidden neuron with the current residual error. Thus, hidden neurons can be used for error correction in consecutive steps, which often leads to very sparse solutions with good generalization ability. CC proved to be particularly appropriate

for training problems where standard feedforward network training is difficult such as the two spirals problem (Fahlmann and Lebiere, 1990). The main difference of CC networks with respect to feedforward networks lies in the particularly efficient constructive training method. Since any feedforward network structure can be embedded into the structure of a CC network and vice versa, the principled representational capabilities of CC networks and standard feedforward networks coincide (Hornik, Stinchcombe, and White, 1989).

Like standard feedforward networks, CC only deals with vectors of fixed dimensionality. Recurrent cascade correlation (RCC) has been proposed as a generalization of CC to sequences of unlimited length (Fahlmann, 1991). Recently, more powerful models for tree structures and acyclic directed graphs have also been proposed and successfully applied as prediction models for chemical structures: recursive cascade correlation and contextual recursive cascade correlation, respectively (Bianucci et al., 2000; Micheli, 2003; Micheli, Sona, and Sperduti, 2003b; Micheli, Sona, and Sperduti, 2002; Micheli, Sona, and Sperduti, 2000; Sperduti, Majidi, and Starita, 1996). The efficient training algorithm of CC can be transferred to these latter models and very good results have been reported. Here, we are interested in the in-principle representational capabilities of these models for different type of structures.

RCC extends simple CC by recurrent connections such that input sequences can be processed step by step. Thereby, the iterative training scheme of hidden neurons is preserved and thus training is very efficient. However, because of this iterative training scheme, hidden neurons have to be independent of all hidden neurons which are introduced later. This causes the fact that a RCC network, unlike RNNs, is not fully recurrent. Recurrent connections of a hidden neuron are pointing towards the neuron itself and towards hidden neurons which are introduced later, but

not towards previous hidden neurons. Because of this fact, RNNs can in general not be embedded into RCC networks: the two models differ with respect to the network architectures. RNNs constitute universal approximators (Funahashi and Nakamura, 1993) and the question now occurs whether the restricted topology of RCC networks, which makes the particularly efficient iterative training scheme possible, preserves this property. This is not clear because local recurrence instead of full connectivity of neurons might severely limit the representational capabilities: it has been shown by (Frasconi and Gori, 1996) that the number of functions which can be implemented by a specific locally recurrent network architecture with the Heaviside activation function can be limited regardless of the number of neurons and thus restrictions apply. The work presented in (Giles et al., 1995) explicitly investigates RCC architectures and shows that these networks equipped with the Heaviside function or a monotonically increasing activation function cannot implement all finite automata. Thus, in the long term limit, RCC networks are strictly weaker than fully connected recurrent networks.

However, the universal approximation capability of recurrent networks usually refers to the possibility of approximating every given function on a finite time horizon. This question has not yet been answered in the literature and it will be investigated in this article. We will show that RCC networks are universal approximators with respect to a finite time horizon, i.e. they can approximate every function up to inputs of arbitrary small probability arbitrarily well. Hence RCC networks and recurrent networks do not differ with respect to this notion of approximation capability. Note that RCC networks, unlike RNN networks, are built and trained iteratively, adding one hidden neuron at a time. Thus, the architecture is determined automatically and only small parts of the network are adapted during a training cycle.

As already mentioned, RCC networks have been generalized to recursive cascade correlation (RecCC) networks for tree structured inputs (Bianucci et al., 2000; Sperduti, Majidi, and Starita, 1996; Micheli, 2003). Recursive networks constitute the analogous extension of recurrent networks to tree structures (Frasconi, Gori, and Sperduti, 1998; Goller and Kuchler, 1996; Sperduti and Starita, 1997). For the latter, the universal approximation capability has been established in (Hammer, 2000). However, RecCC networks share the particularly efficient iterative training scheme of RCC and CC networks and thus recurrence is restricted to recursive connections of hidden neurons to neurons introduced later, but not in the opposite direction also in RecCC networks. Thus it is not surprising that the same restrictions as for RCC networks apply and some functions, which can be represented by fully recursive neural networks, cannot be represented by RecCC networks in the long term limit because of these restrictions (Sperduti, 1997). However, the practically relevant representational capabilities if restricted to a finite horizon, i.e. input trees with restricted maximum depth, is not yet answered in the literature. We will show in this article that RecCC networks with multiplicative neurons possess the universal approximation property for finite time horizon and thus constitute valuable and fast alternatives to fully connected recursive networks for practical applications.

Recurrent and recursive models put a causality assumption on data processing: sequences are processed from one end of the sequences to the other; and trees are processed from the leaves to the root. Thus, the state variables associated to a sequence entry depend on the left context, but not on the right one, and state variables associated to internal nodes in a tree depend on their children, but not the other way around. This causality assumption might not be justified by the data. For time series, the causality assumption implies that events in the future depend on events in the past, but not vice versa, which is reasonable. For spatial data such as sen-

tences or DNA strings, the same causality assumption implies that the value at an interior position of the sequence does only depend on the left part of the sequence, but not on the right one. This is obviously not true in general. A similar argument can be stated for tree structures and more general graphs such as they occur e.g. in logic or chemistry. Therefore extensions of recursive models for sequences and tree structures have been proposed which also take additional contextual information into account. The model proposed in (Baldi et al., 1999) trains two RNNs simultaneously to predict the secondary structure of proteins. The networks are thereby transforming the given sequence in reverse directions such that the full information is available at each position of the sequence. A similar approach is proposed in (Pollastri et al., 2002) for grids, whereby four recursive networks are trained simultaneously in order to integrate all available information for each vertex of a regular two-dimensional lattice. Similar problems for sequences have been tackled by a cascade correlation approach including context in (Micheli, Sona, and Sperduti, 2000). Another model with similar ideas has been proposed in (Wakuya and Zurada, 2001). Note that several of these models simply construct disjoint recursive networks according to different causality assumptions, and afterwards integrate the full information in a feedforward network. However, all reported experiments show that the accuracy of the models can significantly be improved by this integration of context if spatial data or graph structures are dealt with.

The restricted recurrence of cascade correlation offers a particularly simple and elegant and, as we will see, fundamentally different way to include contextual information: the hidden neurons are consecutively trained and, once they are trained, frozen. Thus we can introduce recursive as well as contextual connections of these neurons to neurons introduced later without getting cyclic (i.e. ill-defined) dependencies. I.e., given a vertex in a tree or graph, the hidden neuron activation for

this vertex can have direct access to the state of its children and its parents for each previous hidden neuron. Thereby, the effective iterative training scheme of CC is preserved. Obviously, this possibility of context integration essentially depends on the recurrence being restricted and it cannot be transferred to fully recursive networks. This generalization of RecCC networks towards context integration has recently been proposed in (Micheli, Sona, and Sperduti, 2003b; Micheli, Sona, and Sperduti, 2002). These models are applied to prediction tasks for chemical structures, and they compare favorably to models without context integration. Plots of the principal components of the hidden neurons' states of the contextual models indicate that the additional contextual information is effectively used by the network. Unlike the above mentioned approaches, contextual information is here directly integrated into the recursive part of the network such that the information can be used in an earlier stage than e.g. in (Baldi et al., 1999; Pollastri et al., 2002). Thanks to these characteristics, in the cascade approach, the processing of contextual information is efficiently extended, with respect to the previous approaches, from sequence or grids to more complex structured data such as trees and acyclic graphs.

The question now occurs whether the "in principle" extension to contextual processing and the improved accuracy in experiments can be accompanied by more general theoretical results on the universal approximation capability of the model. A first study of the enlarged capacity of the model can be found in (Micheli, Sona, and Sperduti, 2003b; Micheli, 2003), where it is shown that, in a directed acyclic graph, contextual RecCC (CRecCC) takes into account the context of each vertex, including both successors and predecessors of the vertex (as formally proven in (Micheli, Sona, and Sperduti, 2003a)). Moreover, the integration of context extends the class of functions which can be approximated in the sense that more general structures can be distinguished: contextual models can differentiate between ver-

tices with different parents but the same content and children. Finally, contextual RecCC can implement classes of contextual functions where a desired output for a given vertex may depend on the predecessors of the vertex, up to the whole structure (contextual transductions) (Micheli, Sona, and Sperduti, 2003b; Micheli, 2003). Of course, contextual RecCC networks can still implement all the functions which are computable by RCC models (Micheli, Sona, and Sperduti, 2003b; Micheli, 2003).

We will show in this paper that the proposed notion of context yields universal approximators for rooted directed acyclic graphs with appropriate positioning of children and parents. The argumentation can be transferred to the more general case of IO-isomorphic transduction of such structures. Hence context integration enlarges the class of functions which can be approximated to acyclic graph structures.

We will now first formally introduce the various network models and clarify the notation. We then consider the universal approximation capability of cascade correlation networks in detail. We start with the simplest case, the universal approximation property of standard RCC networks. Thereby, only two parts of the proof are specific for the case of sequences, the other parts can be transferred to more general cases. We then prove the universal approximation capability of RecCCs, substituting the specific parts of the previous proof, and finally investigate CRecCCs. The latter case is particularly interesting, since CRecCCs do not constitute universal approximators for all structures, but only a subset. We provide one example for graph structures which cannot be differentiated by these models. We then define a linear encoding of acyclic graph structures and discuss one sufficient property which can easily be tested and which ensures that this encoding is unique. This property requires that positioning of children and parents can be done in a compatible way. We show that this fact can always be achieved via reenumeration and expansion by

empty vertices for directed positional acyclic graphs with limited fan-in and fan-out. Structures which fulfill this property are referred to as directed acyclic bipositional graphs (DBAGs). Afterwards, we show the universal approximation for contextual RecCC networks on DBAGs. This result can immediately be extended to general graph structures for which the proposed encoding is unique. An informal discussion of this property is given. We conclude with a discussion.

## 2 Cascade correlation models for structures

Standard *feedforward neural networks* (FNN) consist of *neurons* connected in an acyclic directed graph. Neuron  $i$  computes the function

$$x_i = \sigma_i \left( \sum_{j \rightarrow i} w_{ij} x_j + \theta_i \right)$$

of the outputs  $x_j$  of its predecessors, where  $w_{ij} \in \mathbb{R}$  are the *weights*,  $\theta_i \in \mathbb{R}$  is the *bias*, and  $\sigma_i : \mathbb{R} \rightarrow \mathbb{R}$  constitutes the *activation function* of neuron  $i$ . The *input neurons*, i.e. neurons without predecessors, are directly set to external values, the output of the network can be found at specified *output neurons*. All other neurons are called *hidden neurons*. Often, neurons are arranged in layers with full connectivity of the neurons of one layer to the consecutive one. Activation functions  $\sigma_i$  which are of interest in this article include the logistic function

$$\text{sgd}(x) = \frac{1}{1 + \exp(-x)}$$

the Heaviside function

$$\text{H}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases},$$

and the identity  $\text{id}(x) = x$ . More generally, a *squashing* activation function  $f : \mathbb{R} \rightarrow \mathbb{R}$  is a monotonic function such that values  $l_f < h_f$  exist with  $\lim_{x \rightarrow \infty} f(x) =$

$h_f$ ,  $\lim_{x \rightarrow -\infty} f(x) = l_f$ . A network which uses activation functions from a set  $\mathcal{F}$  of functions is referred to as an  $\mathcal{F}$ -FNN. It is well known that FNNs with only one hidden layer with a squashing activation function and linear outputs fulfill a universal approximation property (Hornik, Stinchcombe, and White, 1989). The *universal approximation property* holds for FNNs, if for a given probability measure  $P$  on  $\mathbb{R}^n$ , any Borel-measurable  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , and any  $\epsilon, \delta > 0$  we can find a FNN which computes a function  $g$  such that

$$P(x \in \mathbb{R}^n \mid |f(x) - g(x)| > \delta) < \epsilon.$$

There exist alternative characterizations of the hidden neurons activation function for which the universal approximation property holds, e.g. being analytic (Scarselli and Tsoi, 1998).

Often, more powerful neurons computing products of inputs are considered (Alquezar and Sanfeliu, 1995; Forcada and Carrasco, 1995; Hornik, Stinchcombe, and White, 1989). Let us denote the indices of predecessors of neuron  $i$  by  $i_1, \dots, i_j$ . A *multiplicative* neuron of degree  $d \geq 1$  computes a function of the form

$$x_i = \sigma_i \left( \sum_{k=1}^d \prod_{j_1, \dots, j_k \in \{i_1, \dots, i_j\}} w_{ij_1 \dots j_k} x_{j_1} \cdot \dots \cdot x_{j_k} + \theta_i \right)$$

where  $w_{ij_1 \dots j_k} \in \mathbb{R}$  refers to the weights and  $\theta_i$  is the bias as above. We will explicitly indicate if multiplicative neurons are used. Otherwise, the term ‘neuron’ refers to standard neurons.

Standard neural network training first chooses the network architecture and then fits the parameters on the given training set. In contrast, cascade correlation (CC) determines the architecture and the parameters simultaneously. CC starts with a minimum architecture without hidden neurons, and iteratively adds hidden neurons unless the approximation accuracy is satisfiable. Each hidden neuron is connected to

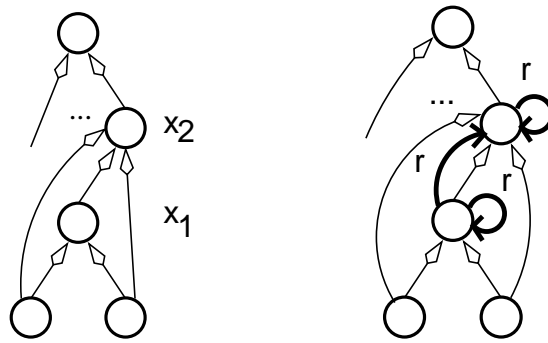


Figure 1: Left: An example of a cascade correlation network with cascaded hidden neurons  $x_i$  added iteratively to the network. Right: A recursive cascade correlation network where recursive connections, indicated by thick lines marked with  $r$ , are used to store contextual information from the first part of a processed sequence. Thereby, the activation of the neuron of the *previous* time step is propagated through recurrent connections. Note that recurrent connections point from  $x_1$  to  $x_2$  but not vice versa.

all inputs and all previous hidden neurons. The weights associated to these connections are trained in such a way to maximize the correlation between their output and the residual error of the network constructed so far. Afterwards, the weights entering the hidden neuron are frozen. All weights to the outputs are re-trained according to the given task. If needed, new hidden neurons are iteratively added in the same way. One CC network with two hidden neurons is depicted in Fig. 1. Since only one neuron is trained at each stage, this algorithm is very fast. Moreover, it usually provides excellent classification results with a small number of hidden neurons which, because of the specific training method (Fahlmann and Lebiere, 1990), serve as error correcting units. This training scheme is also used for all further cascade correlation models which are introduced in this article. Since we are here not interested in the specific training method but in the in-principle representation and

approximation capabilities of the architectures, we will not explain the training procedure in more details. The only point of interest in this paper is that the iterative training method assumes that hidden neurons are functionally dependent on previous hidden neurons, while they are functionally independent of hidden neurons introduced later, because of the iterative training scheme.

## 2.1 Recurrent models

Obviously, FNNs can only deal with input vectors of finite and fixed size. *Recurrent networks* (RNNs) constitute one alternative approach which allows the processing of sequences of unlimited length. Denote by  $\mathcal{L}$  a set where the labels are taken from, e.g.  $\mathcal{L} = \mathbb{R}^n$  for continuous labels or  $\mathcal{L} = \{1, \dots, r\}$  for discrete labels. Assume a sequence  $[l_1, \dots, l_T]$  of length  $T$  over  $\mathcal{L}$  is given. Denote the set of finite sequences over  $\mathcal{L}$  by  $\mathcal{L}^*$ . A recurrent network recursively processes a given sequence from the left to the right. Assume that the output of neuron  $i$  has to be computed for a given sequence entry  $l_t$  at position  $t$  of the sequence. Assume  $N$  neurons are present. We assume for simplicity that the immediate predecessors of neuron  $i$  are the neurons from 1 to  $i - 1$ .<sup>1</sup> The functional dependence of neuron  $i$  can then be written as

$$x_i(l_t) = \tau_i(l_t, x_1(l_t), \dots, x_{i-1}(l_t), x_1(l_{t-1}), \dots, x_N(l_{t-1})) \quad (1)$$

---

<sup>1</sup>It can always be achieved via reenumeration that the predecessors are a subset of this set of neurons. Thus, setting some weights to 0, we can represent every FNN in this form.

where  $\tau_i$  is a function computed by a single neuron,<sup>2</sup> and for the initial case  $t = 1$  we set  $x_i(l_0)$ , interpreted as the output for the empty sequence, to some specified value, e.g. 0. The computed values depend on the values of all neurons in the previous time step and thus, indirectly, on the left part of the sequence. The final output of the sequence is given by the values  $x_i(l_T)$  obtained after processing the last entry.

Recurrent cascade correlation (RCC) also refers to the activations of neurons in the previous time step. We assume, that the neurons are enumerated in the order in which they are created during training, i.e. neuron  $i$  is added to the RCC network in the  $i$ th construction step. Because of the iterative training scheme, hidden neuron  $i$  is functionally dependent only on neurons introduced in previous steps. I.e., the functional dependence now reads as

$$x_i(l_t) = \tau_i(l_t, x_1(l_t), \dots, x_{i-1}(l_t), x_1(l_{t-1}), \dots, x_i(l_{t-1}))$$

where  $\tau_i$  is the function computed by a single neuron. Again,  $x_i(l_0)$  is set to some specified initial value. Note that, unlike RNNs, RCC networks are not fully connected. Recurrent connections which start at neurons  $j > i$  to neuron  $i$  are dropped, because all weights to neuron  $i$  are, once trained, frozen. One example network with two hidden neurons is depicted in Fig. 1.

It is well known that fully recurrent neural networks constitute universal approximators in the sense that, given appropriate activation functions of the neurons, any function can be approximated. <sup>2</sup>Here we assume only dependences from the previous time step. Dependences from time steps  $t - i, i > 1$  could be considered and added to eq. (1), since the definition would still be well defined. Since more information is then available at one time step, benefits for concrete learning problems could result. Obviously, the universal approximation capability of eq. (1) immediately transfers to the more general definition. Thus, for simplicity, we restrict ourselves to eq. (1) in the following, and we restrict in a similar way for the case of input trees or structures.

measurable function from sequences into a real vector space can be approximated arbitrarily well, i.e. the distance of the desired output from the network output is at most  $\delta$  for inputs of probability at least  $1 - \epsilon$  (Hammer, 2000). We will later show that an analogous result holds also for RCC networks with restricted recurrence.

## 2.2 Recursive models

Both, RNNs and RCC networks have been generalized to tree structures in the literature. A tree over  $\mathcal{L}$  consists either of an empty structure  $\xi_t$ , or of a vertex  $v$  with label  $l(v) \in \mathcal{L}$  and a finite number of subtrees some of which may be empty. We refer to the  $i$ th children of  $v$  by  $ch_i(v)$ . A tree has limited fan-out  $k$  if each vertex has at most  $k$  children or subtrees. In this case we can always assume that exactly  $k$  children are present for any nonempty vertex, introducing empty vertices if necessary. The set of finite trees with fan-out  $k$  over  $\mathcal{L}$  is denoted by  $\mathcal{L}_k^*$ . For simplicity, we only deal with fan-out 2 in the following. The generalization to larger fan-out will be immediate. Fan-out 1 gives sequences. Since trees form a recursive structure, recursive processing of each vertex in the tree starting from the leaves up to the root is possible. This leads to the notion of *recursive networks* (RecNNs) as introduced e.g. in (Frasconi, Gori, and Sperduti, 1998; Goller and Küchler, 1996; Sperduti and Starita, 1997) and *recursive cascade correlation* (RecCC) as proposed in (Bianucci et al., 2000; Sperduti, Majidi, and Starita, 1996). As beforehand, for RecNN we assume direct dependence of neuron  $i$  from neurons  $< i$  in the network, and we assume the network has  $N$  neurons. Given a vertex  $v$  in the tree, neuron  $i$  then computes a function of the form

$$x_i(v) = \tau_i(l(v), x_1(v), \dots, x_{i-1}(v), \\ x_1(ch_1(v)), \dots, x_N(ch_1(v)), x_1(ch_2(v)), \dots, x_N(ch_2(v)))$$

where  $\tau_i$  is given by a (possibly multiplicative) neuron. Thus, the activation of  $v$  depends on the recursively computed activations of all neurons of the network for the children of  $v$ . The output of a tree is taken as the output for the root vertex of the tree.

As for RCC networks, the functional dependence of hidden neuron  $i$  of a RecCC network for an input vertex  $v$  has to be restricted to take the iterative training scheme into account. It becomes

$$x_i(v) = \tau_i(l(v), x_1(v), \dots, x_{i-1}(v), \\ x_1(ch_1(v)), \dots, x_i(ch_1(v)), x_1(ch_2(v)), \dots, x_i(ch_2(v)))$$

where  $\tau_i$  can be computed by some (possibly multiplicative) neuron. This formula can be alternatively written as

$$x_i(v) = \tau_i(l(v), x_1(v), \dots, x_{i-1}(v), q^{-1}(x_1(v)), \dots, q^{-1}(x_i(v)))$$

whereby we define the operator  $q^{-1}(x_i(v)) = (x_i(ch_1(v)), x_i(ch_2(v)))$ . This compact notation shows how the dependencies can be transferred to larger fan-out  $k$ : the operator is then substituted by  $q^{-1}(x_i(v)) = (x_i(ch_1(v)), \dots, x_i(ch_k(v)))$ .

It has been shown in (Hammer, 2000) that fully recursive networks constitute universal approximators on tree structures. We will show in this article an analogous result for RecCC networks with multiplicative neurons.

### 2.3 Contextual models

The restricted recurrence of RecCC networks allows us to introduce further functional dependencies without getting cycles which might be particularly valuable for more general structures. We are interested in so-called directed positional acyclic graphs, which constitute a generalization of tree structures in the sense that vertices might have more than one parent. Formally, we define DPAGs as specific graph

structures: a labeled directed acyclic graph (DAG) over  $\mathcal{L}$  consists of a finite set  $vert(\mathcal{D})$  of vertices  $v$  with labels  $l(v) \in \mathcal{L}$  assigned to each  $v$  and a finite set of edges  $edge(\mathcal{D}) \subset vert(\mathcal{D}) \times vert(\mathcal{D})$ , such that no cycles can be observed in the graph. For each  $v \in vert(\mathcal{D})$  its parents are given by  $parent(v) = \{u \in vert(\mathcal{D}) \mid (u, v) \in edge(\mathcal{D})\}$  and the children are  $children(v) = \{u \in vert(\mathcal{D}) \mid (v, u) \in edge(\mathcal{D})\}$ . As usual, the fan-in and fan-out of vertices are given by  $fan\_in(v) = |parent(v)|$  and  $fan\_out(v) = |children(v)|$ . A rooted directed positional acyclic graph over  $\mathcal{L}$  (DPAG) consists of a labeled directed acyclic graph  $\mathcal{D}$  such that all vertices in the graph can be reached from one specific vertex, the root  $root(\mathcal{D})$ . In addition, an ordering of the parents and children is specified, i.e. two injective functions  $P_v : parent(v) \rightarrow \{1, \dots, fan\_in(v)\}$  and  $C_v : children(v) \rightarrow \{1, \dots, fan\_out(v)\}$ , respectively, are fixed for each  $v$  which specify the order in which the children and parents are enumerated when considering  $v$ .

For each  $v \in vert(\mathcal{D})$  we define its height  $height(v)$  as the length of a longest path from the root to  $v$  plus one, and the height of the DPAG,  $height(\mathcal{D})$  as the maximum height of the contained vertices. I.e. the root has height 1. Note that a tree is just a specific form of DPAGs where each vertex (except for the root) possesses exactly one parent.

*Contextual recursive cascade correlation* (CRecCC) (Micheli, Sona, and Sperduti, 2003b; Micheli, Sona, and Sperduti, 2002; Micheli, 2003) can also take into account information about parents. Assume DPAGs with limited fan-in at most  $k_1$  and fan-out at most  $k_2$  are given. In the following, we restrict ourselves to DPAGs with fan-in and fan-out two. The generalization to larger fan-in and fan-out is immediate unless stated otherwise. We denote the set of DPAGs over  $\mathcal{L}$  with limited fan-in and fan-out 2 by  $DPAG_{\mathcal{L}}^2$ . As beforehand, for a vertex  $v$  in a DPAG,  $ch_1(v)$  and  $ch_2(v)$  refer to the first and second child of vertex  $v$ , respectively.  $pa_1(v)$  and

$pa_2(v)$  refer to the two parents of the vertex. An empty child is denoted by  $\xi_c$ .  $\xi_p$  refers to an empty parent. The empty DPAG is also denoted by  $\xi_c$ . In a CRecCC network, each hidden neuron has also access to the activation of previous hidden neurons for all parents of the vertex. Formally, given a vertex  $v$  of a DPAG, the functional dependencies of hidden neuron number  $i$ , can be written as

$$\begin{aligned} x_i(v) = & \tau_i(l(v), x_1(v), \dots, x_{i-1}(v), \\ & x_1(ch_1(v)), \dots, x_i(ch_1(v)), x_1(ch_2(v)), \dots, x_i(ch_2(v)), \\ & x_1(pa_1(v)), \dots, x_{i-1}(pa_1(v)), x_1(pa_2(v)), \dots, x_{i-1}(pa_2(v))) \end{aligned}$$

where  $x_i(\xi_p) := 0_p^i$ ,  $x_i(\xi_c) = 0_c^i$  for some fixed values  $0_p^i$  and  $0_c^i$  in  $\mathbb{R}$ , and  $\tau_i$  constitutes the function computed by the respective (possibly multiplicative) hidden neuron. For  $i = 1$ , this reduces to

$$x_1(v) = \tau_1(l(v), x_1(ch_1(v)), x_1(ch_2(v))).$$

As beforehand, the output of the root vertex is considered output of a given DPAG. Defining the operator  $q^{+1}(x_i(v)) := (x_i(pa_1(v), x_i(pa_2(v)))$  allows us to recast the functional dependence as

$$\begin{aligned} x_i(v) = & \tau_i(l(v), x_1(v), \dots, x_{i-1}(v), \\ & q^{-1}(x_1(v)), \dots, q^{-1}(x_i(v)), q^{+1}(x_1(v)), \dots, q^{+1}(x_{i-1}(v))) \end{aligned}$$

The general functional dependence for larger fan-in and fan-out is thus obtained by extending  $q^{-1}$  and  $q^{+1}$  to more children and parents, respectively. Thus neuron  $i$  also depends on the activation of parents and, indirectly, its parents children, via the previous hidden neuron. This is possible without getting cyclic definitions because of the restricted recurrence of RecCC networks. In particular, the strict causality assumption of vertices depending only on successors is here no longer valid and increasing context is taken into account for large  $i$ , eventually involving the whole structure in the context.

The above recurrent dynamic is well defined due to two features: on one hand, the DPAGs are acyclic and rooted, such that, as in any tree structure, induction over the structure is possible. Starting from the vertices without further successors we can continue to process data up to the root. We refer to this principle as *structural induction*. On the other hand, the network architecture is inductively constructed whereby hidden neurons constructed in earlier steps are functionally independent of neurons introduced later. Hence *induction over the hidden neurons* is possible, whereby we can assume that the output of all previously constructed hidden neurons for all vertices in a DPAG are available. Since RecNNs might be fully connected, structural induction cannot be applied for general RecNNs: each hidden neuron is dependent on the output of every other hidden neuron.

Note that we can substitute a CRecCC network for each given fixed DPAG structure to be processed by an equivalent feedforward network: we take copies of all neurons for all vertices in the DPAG and unfold the recurrent connections according to the given structure over this set. This general principle, *unfolding* a recursive network according to a given fixed input structure, can obviously be done for all network models defined so far. It is based on the principle that no cycles are given in the dynamics.

We would like to add a last remark to the definition of cascade architectures: in practice, a set of output neurons is usually introduced and all hidden neurons are connected to the output neurons. The output connections are trained directly on the given training set. For simplicity, we will drop this output layer in the following, since we are not interested in training. We assume that the last hidden neuron of the cascade correlation network serves as output neuron, to simplify notation. We will, for simplicity, only deal with the case of functions to  $\mathbb{R}$  instead of outputs with higher dimensionality. The generalization of our results to outputs in  $\mathbb{R}^m$  will be

immediate.

### 3 Universal approximation ability

Having introduced the cascade correlation models, we will now consider the universal approximation capability of these models with respect to sequences, tree structures, and DPAGs, respectively. We investigate the case of sequences first and point out the structure of this first proof of approximation completeness. It will turn out that we only have to substitute two steps which are specific for sequences. A generalization to tree structures will be quite simple. The case of graphs, however, will be much more involved, because fundamentally new results are implied in this part: the investigation will include an analysis of the possibility to represent acyclic graphs by linear representations. It will, as a by-product, show that contextual networks can also approximate so-called IO-isomorphic transductions (as defined below), i.e. they can deal with the case that not only the inputs, but also the outputs are structured.

We first want to clarify which type of functions we are interested in. Input sets are sequences, trees, or DPAGs of arbitrary but finite size. We assume that the respective labels are contained in a subset  $\mathcal{L}$  of a real-vector space. For discrete  $\mathcal{L}$ , the discrete  $\sigma$ -algebra over  $\mathcal{L}$  is considered, otherwise we deal with the Borel  $\sigma$ -algebra over  $\mathcal{L}$ . This algebra is expanded to structures by interpreting structures as the direct sum of objects of one fixed structure but possibly varying labels, e.g. sequences decompose into the sum of sequences of a fixed length  $T$ . Thus the set of structures is equipped with the sum of the Borel  $\sigma$ -algebras of objects with a fixed structure. Note that the choice of the sum-algebra has consequences on the semantic: for every fixed probability measure  $P$  we can find a size  $n$  such that

most information lies on the structures of size at most  $n$ .  $n$  depends on the chosen probability measure and might vary with  $P$ . A formal argument for this fact will be given in the proof of Theorem 3.2. We do not consider infinite length sequences or infinite size structures in this article.

A given (measurable) function with structural inputs shall be approximated by a cascaded network. We assume, that the weights of the network are fixed when processing data, i.e. stationarity of the function  $\tau_i$  computed by neuron  $i$  with respect to the structure holds.

There are different possibilities to use recursive models: so-called supersource transduction refers to the task to map an entire structure to a given value, e.g. map a chemical molecule to its activity. Thus only the value obtained after having processed the whole structure is of interest. We will mainly focus on functions of this type, whereby we assume that the outputs are element of the real numbers. I.e. functions to be approximated are measurable functions of the form  $f : S \rightarrow \mathbb{R}$ ,  $S$  denoting the considered structures, i.e. sequences, graphs, or DPAGs. Recursive models can alternatively be used for IO-isomorphic transduction, which are functions  $f : S \rightarrow S$  which map a given structure to a structure of the same form but possibly different labels. The output  $f(s)$  is obtained substituting each vertex label  $l(v)$  of the input structure  $s$  by the output value  $x_i(v)$  obtained for this vertex in the recursive model. This setting describes problems such as mapping the sequence of proteins to its secondary structure, where the form of the secondary structure of the protein at each aminoacid position has to be predicted. We will shortly discuss this situation and it will turn out that our results on the approximation capability of CRecCC networks for supersource transductions will imply results for IO-isomorphic transductions.

### 3.1 Recurrent CC

We start with the simplest case, recurrent networks for sequence processing. The key ideas of the proof borrow steps from the universal approximation proof of recursive neural networks as given in (Hammer, 2000). For convenience, we first outline the general steps of the proof some of which can be reused for recursive cascade correlation and contextual recursive cascade correlation. The key points are:

- (I) Find a linear code for the given structures over discrete labels up to a finite height; a linear code can be represented as a real number whereby the number of digits is not priorly limited. This code serves as a candidate to act as an internal distributed representation of the structure within a neural network.
- (II) Show that this code can be computed with the considered network structure; as a consequence of such a construction, it is clear that the network can turn a structured input to an internal distributed code suitable for processing with a standard feedforward network.
- (III) Integrate a universal FNN; a universal FNN can map the encoded structures to the desired outputs. As a result of these three steps, it is proved that the considered network is capable of mapping every finite number of discrete labeled structures to arbitrary output values.
- (IV) Integrate discretization of real-valued labels; this allows us to turn structures with real-valued labels to similar discrete structures which we can process. As a consequence of this step, we can approximate functions on structures with real-valued labels by first turning them internally to discrete labels.
- (V) Approximate non-standard activation functions; the above construction steps use specific activation functions (exact identity, Heaviside function, etc.) to

make sure that the desired outputs are precisely met. These have to be substituted by the considered activation functions such as the logistic transfer function.

Only step **(I)** and **(II)** will be specific for the considered network and input structure. The other steps do not specifically refer to the model but can be similarly done in the other cases. We will use this fact later to generalize universal approximation for RCC networks to RecCC and CRecCC networks.

Now we present the construction steps for RCC networks for sequences. The first step is trivial, since sequences are naturally represented in a linear way. Assume  $\mathcal{L}$  is a finite alphabet. A sequence over  $\mathcal{L}$  with elements  $l_i \in \mathcal{L}$  is given by  $[l_1, \dots, l_t]$ , a linear code. This is essentially step **(I)**. This linear representation can be embedded in the real numbers by concatenating the entries of the sequence. More precisely, assume  $\mathcal{L} = \{1, \dots, r\} \subset \mathbb{N}$  and assume the numbers can be represented by at most length  $L$ . Then the representation  $0.l_t \dots l_1$  where the  $l_i$  are expanded by leading entries 0 to exactly  $L$  places is a unique representation of the sequence  $[l_1, \dots, l_t]$ . This observation is obvious because each entry  $l_i$  occupies exactly the same number of digits within this representation.

The next observation of the construction is that this code can be “computed” by a recursive cascade correlation network. More precisely, we can construct a RCC network with only one neuron and output  $0.l_t \dots l_1$  ( $l_i$  expanded by leading entries 0 to  $L$  digits) for an input sequence  $[l_1, \dots, l_t]$ . We define the output of the empty sequence by 0. The functional dependence of neuron  $x_1$  of a RCC network is

$$x_1(l_t) = \tau_1(l_t, x_1(l_{t-1})).$$

Assume as a structural assumption that  $x_1(l_{t-1})$  already represents  $0.l_{t-1} \dots l_1$ . Then

$$\tau_1(l_t, 0.l_{t-1} \dots l_1) := 0.1^L \cdot l_t + 0.1^L \cdot 0.l_{t-1} \dots l_1 = 0.l_t l_{t-1} \dots l_1.$$

That means the function

$$\tau_1(a_1, a_2) = 0.1^L \cdot a_1 + 0.1^L \cdot a_2$$

gives the desired functionality. Note that  $\tau_1$  can be computed by a standard RCC neuron with linear activation function. Hence step **(II)** is very easy for the sequence setting as well; we have found a linear RCC network with only one neuron which encodes all sequences with a finite input alphabet and finite height  $h$  (in this case also infinite  $h$ ) uniquely within a real number. Steps **(III)** to **(V)** now mainly consist of general observations which are not specific for RCC networks.

Step **(III)** is the integration of feedforward parts into a RCC network without violating the structure. For any given function  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  we denote by  $f^D : (\mathbb{R}^n)^* \rightarrow (\mathbb{R}^m)^*$  the extension to sequences which maps each entry of a given sequence over  $\mathbb{R}^n$  to the image of the entry under  $f$  in  $\mathbb{R}^m$  thus giving a sequence over  $\mathbb{R}^m$ . Later, we will use the same notation also for more general structures, trees and DPAGs, to refer to the map which substitutes each entry of a structure by its image under  $D$ . So far, we have defined cascade correlation networks to compute functions with output domain  $\mathbb{R}$ . We can extend to  $\mathbb{R}^m$  for  $m > 1$  by interpreting the output of the last  $m$  neurons  $x_{\max-m}, \dots, x_{\max}$  as the output of the network.

**Lemma 3.1** *Assume  $f : \mathbb{R}^i \rightarrow \mathbb{R}^n$  and  $g : \mathbb{R}^m \rightarrow \mathbb{R}$  are computed by FNNs. Assume  $h : (\mathbb{R}^n)^* \rightarrow \mathbb{R}^m$  is computed by a RCC network. Then the composition  $g \circ h \circ f^D : (\mathbb{R}^i)^* \rightarrow \mathbb{R}$  can also be computed by a RCC network.*

*Proof.* This statement is easy because we can interpret all neurons of  $f$  and  $g$  as additional hidden neurons of the RCC network, setting connections which are not used in these subnetworks but which are introduced due to the definition of RCC architectures to 0. More precisely, denote the neurons in  $f$  by  $n_1, \dots, n_{i_f}$ , whereby connections  $n_i \rightarrow n_j$  only exist if  $i < j$ . Denote the neurons in  $g$  by

$m_1, \dots, m_{i_g}$ , again with connections existing only to neurons with larger indices. Denote the neurons of the RCC network by  $o_1, \dots, o_{i_h}$  arranged in the order of their appearance. Then we can choose the weights of a RCC network such that hidden neuron number  $i$  takes the role of

$$x_i = \begin{cases} n_i & \text{if } i \leq i_f \\ o_{i-i_f} & \text{if } i_f < i \leq i_f + i_h \\ m_{i-i_f-i_h} & \text{otherwise} \end{cases}$$

This is possible because the functional dependence of the RCC hidden neurons ensure direct access to the respective predecessors in the single networks: each hidden neuron has access to the activation of all previous neurons. All recursive connections in the RCC network for  $n_i$  and  $m_i$  are set to 0. The same holds for all additional feedforward as well as recursive connections not present in the original part  $n_i$ ,  $o_i$ , and  $m_i$ . Note that recurrence in the first part corresponding to  $f$  has the effect that  $f^D$  is computed on input sequences. Recurrence in  $g$  has no effect, because the output of the last sequence entry is considered output of the entire computation.  $\square$

Note that an analogous result holds for RecCC networks and CRecCC networks. It is not necessary to change the proof if  $h$  and  $g \circ h \circ f^D$  are computed by RecCC networks or CRecCC networks.

We obtain as an immediate result for the sequence case:

**Theorem 3.2** *Assume  $\mathcal{L} = \{1, \dots, r\}$  is a finite set. Assume  $P$  is some probability measure on sequences over  $\mathcal{L}$ . Assume  $f : \mathcal{L}^* \rightarrow \mathbb{R}$  is a function. Assume  $\epsilon > 0$ . Then there exists a RCC network with activation functions in  $\{\text{id}, \sigma\}$  which computes a function  $g$  such that*

$$P(x \in \mathcal{L}^* \mid f(x) \neq g(x)) < \epsilon.$$

$\sigma$  is thereby some activation function for which the universal approximation property of FNNs is fulfilled.

*Proof.*  $\mathcal{L}^*$  decomposes into sets of sequences of different length. We denote these sets by  $(\mathcal{L}^*)_i$  for  $i = 1, \dots, \infty$  and an arbitrary enumeration.<sup>3</sup> Since  $(\mathcal{L}^*)_i$  is measurable,  $1 = P(\mathcal{L}^*) = \sum_{i=1}^{\infty} P((\mathcal{L}^*)_i)$  thus we find  $\sum_{i=n_0}^{\infty} P((\mathcal{L}^*)_i) < \epsilon/4$  for some  $n_0$ . Therefore we can find some length  $h$  (the maximum length in the sets  $(\mathcal{L}^*)_i$  for  $i < n_0$ ) such that larger sequences have probability smaller than  $\epsilon$ . For sequences of length at most  $h$  over  $\mathcal{L}$ , we can find a RCC network with the identity as activation function unit which maps these inputs to unique values, as shown above. Because FNNs with activation function  $\sigma$  and linear outputs possess the universal approximation property, we can find a FNN which maps these encodings exactly to the outputs given by  $f$  (Sontag, 1992). The composition of these two networks can be interpreted as a RCC network because of Lemma 3.1 with the desired properties.  $\square$

Note that we have restricted ourselves to outputs in  $\mathbb{R}$ , for convenience. It is obvious from the above construction that the same result can be obtained for functions  $f : \mathcal{L}^* \rightarrow \mathbb{R}^m$  if we integrate several output neurons for RCC networks. The same remark holds for all universal approximation results which we present in this paper thus giving us approximation results for output set  $\mathbb{R}^m$  instead of  $\mathbb{R}$ .

We can extend the above result to sequences with real labels, introducing a first part which maps real-valued labels to a discrete set.

---

<sup>3</sup>We could simply choose  $(\mathcal{L}^*)_i$  as the set of sequences of length  $i$ . However, we would like to achieve compatibility with the more general case of tree structures or DPAGs, such that we use an arbitrary enumeration of the subsets with fixed structure.

**Theorem 3.3** Assume  $\mathcal{L} = \mathbb{R}^n$ . Assume  $P$  is some probability measure on sequences over  $\mathcal{L}$ . Assume  $f : \mathcal{L}^* \rightarrow \mathbb{R}$  is a Borel-measurable function. Assume  $\epsilon > 0$ ,  $\delta > 0$ . Then there exists a RCC network with activation functions in  $\{\text{id}, \sigma, \text{H}\}$  which computes a function  $g$  such that

$$P(x \in \mathcal{L}^* \mid |f(x) - g(x)| > \delta) < \epsilon.$$

$\sigma$  is thereby some activation function for which the universal approximation property of FNNs is fulfilled.

*Proof.* We can fix some length  $h$  such that longer sequences have probability at most  $\epsilon/4$ . We can find a constant  $C > 0$  such that sequences with one entry outside  $[-C, C]^n$  have probability smaller than  $\epsilon/4$ .<sup>4</sup> For sequences of length at most  $h$  and labels in  $[-C, C]^n$ , we can approximate  $f$  by a continuous function such that the probability of values with distance larger than  $\delta/2$  is at most  $\epsilon/4$ .<sup>5</sup> Hence substituting  $\epsilon$  by  $\epsilon/4$  and  $\delta$  by  $\delta/2$ , it is sufficient to prove the stated theorem for continuous functions  $f$  on sequences of limited length at most  $h$  with limited labels in  $[-C, C]^n$ .

Note that such function  $f$  is uniformly continuous, being a function on a compact interval. Hence we can find  $\epsilon_0 > 0$  with the following property<sup>6</sup>: given any two sequences  $\mathcal{D}$  and  $\mathcal{D}'$  of the same structure, i.e. the same length. Assume  $l_v$  is

---

<sup>4</sup>This is the same argument as above because the set of sequences decomposes into subsets of sequences with entries in a fixed interval with boundaries given by natural numbers.

<sup>5</sup>This is a well-known fact about functions over the real numbers, and it follows e.g. from the approximation results in (Hornik, Stinchcombe, and White, 1989).

<sup>6</sup>We use a notation which could refer to sequences as well as trees or DPAGs for convenience. An identical prove shows an analogous result for these more general cases.

entry of  $\mathcal{D}$  and  $l_{v'}$  is an entry of  $\mathcal{D}'$  at the same position. Assume that for every such pair of entries the inequality  $|(l_v)_i - (l_{v'})_i| < \epsilon_0$  holds for all positions  $i = 1, \dots, n$ ,  $(l_v)_i$  and  $(l_{v'})_i$ , respectively, denoting component number  $i$  of the respective entry. Then  $|f(\mathcal{D}) - f(\mathcal{D}')| < \delta$ . Next we choose points  $I_0 < I_1 < \dots, I_p$  in  $[-C, C]$  such that  $I_0 = -C$ ,  $I_p = C$ , and  $|I_j - I_{j-1}| < \epsilon_0$  for all  $j = 1, \dots, p$ . Note that varying the components of the entries of a sequence within some interval  $[I_{j-1}, I_j)$  does not change the output with respect to  $f$  by more than  $\delta$ . Therefore we can find a function  $f^1$  which differs from  $f$  by at most  $\delta$  for all inputs and which maps all sequences of the same structure where the components of the sequence elements are contained in the same interval  $[I_{j-1}, I_j)$  to a constant value.

We will now show that this function  $f^1$  can be computed by a RCC network, which hence approximates  $f$  as stated in the theorem. The characteristic function of  $[I_{j-1}, I_j)$  can be computed as  $1_{[I_{j-1}, I_j)}(x) = H(x - I_{j-1}) - H(x - I_j)$ . The function

$$d : [-C, C]^n \rightarrow \{1, \dots, (p+1)^n - 1\}, (x_1, \dots, x_n) \mapsto \sum_{i=1}^n \sum_{j=1}^p 1_{[I_{j-1}, I_j)}(x_i) (p+1)^{i-1}$$

uniquely encodes the intervals in which the components of the input lie.  $d$  thereby uses an encoding to base  $p+1$  of the  $p$  intervals in  $n$  dimensions. Since  $f^1$  is piecewise constant, we find a function  $f^2$  such that  $f^1 = f^2 \circ d^D$  where, as beforehand,  $d^D$  denotes the application of  $d$  to each element in a sequence. This function  $f^2$  just maps a code to the output of some fixed sequence with entries in the encoded intervals as indicated by the input. Because of Theorem 3.2, we can find a RCC network  $g^1$  which computes  $f^2$ . (Since we are dealing with only a finite length at this point, the approximation can be done for all input values.) Hence  $g^1 \circ d^D$  equals with  $f^1$ . Note that  $d$  can be computed by a feedforward network with activation function in  $\{\mathbf{H}, \text{id}\}$ . Because of Lemma 3.1,  $g^1 \circ d^D$  can be computed by a RCC network with activation function in  $\{\mathbf{H}, \text{id}, \sigma\}$ .  $\square$

Usually, a sigmoidal activation function is used as activation function of the RCC network for all neurons but the last one, which is linear to account for the fact of a priori unbounded output set. So we are interested in the possibility to substitute the above specific activation functions by a sigmoidal function  $\sigma$ . Note that for any function  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  which is continuously differentiable in the vicinity of some point  $x_0$  with nonvanishing derivative  $\sigma'(x_0)$  the following holds:

$$\frac{\sigma(x_0 + \epsilon \cdot x) - \sigma(x_0)}{\epsilon \cdot \sigma'(x_0)} \rightarrow x \quad (\epsilon \rightarrow 0)$$

This convergence is uniform on compacta. If the above property holds for  $\sigma$  we refer to  $\sigma$  as being *locally  $C^1$* . Moreover, for a squashing function  $\sigma$  with limits  $l_\sigma$  and  $h_\sigma$  we find

$$(\sigma(x/\epsilon) - l_\sigma)/(h_\sigma - l_\sigma) \rightarrow H(x) \quad (\epsilon \rightarrow 0)$$

for all  $x \neq 0$ . The convergence is uniform on  $(-\infty, -\delta] \cup [\delta, \infty)$  for any  $\delta > 0$ . These two properties are in particular fulfilled for the standard logistic activation function. The following theorem allows us to substitute the above specific activation functions by an activation function as used in practice.

**Theorem 3.4** *Assume  $\mathcal{L} = \mathbb{R}^n$ . Assume  $P$  is some probability measure on sequences over  $\mathcal{L}$ . Assume  $f : \mathcal{L}^* \rightarrow \mathbb{R}$  is a Borel-measurable function. Assume  $\epsilon > 0$ ,  $\delta > 0$ . Assume  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  is a continuous and locally  $C^1$  squashing function. Then there exists a RCC network with activation functions  $\sigma$  and linear activation function for the last neuron of the network which computes a function  $g$  such that*

$$P(x \in \mathcal{L}^* \mid |f(x) - g(x)| > \delta) < \epsilon.$$

*Proof.* Note that a squashing activation function leads to the universal approximation property of FNNs (Hornik, Stinchcombe, and White, 1989), such that we can

construct as in Theorem 3.3 a RCC network which contains the activation functions  $\text{id}$  and  $\text{H}$  in addition to  $\sigma$ . We now show that the activation functions  $\text{id}$  and  $\text{H}$  can be substituted in the network by the above approximations for appropriate  $\epsilon$  such that the difference of the outputs is small for inputs of high probability. Note that we argued in the proof of Theorem 3.3 that we can restrict to finite length  $h$  and labels in  $[-C, C]^n$ . Moreover, we can choose the interval borders  $I_j$  which have been used to discretize the inputs in such a way that the probability of sequences which contain a label with a component in  $(I_j - \delta_0, I_j + \delta_0)$  is smaller than  $\epsilon/2$  for some  $\delta_0$ . Denote by  $\mathcal{L}' = ([-C, C] \setminus \cup_{j=0}^p (I_j - \delta_0, I_j + \delta_0))^n$ . Since the involved functions are continuous, the set of activations  $x_i(l_v)$  for all elements  $l_v$  of sequences of length at most  $h$  with labels in  $\mathcal{L}'$  is compact, say it is contained in  $[-C_0, C_0]$ .

We formally substitute each activation function  $\text{id}$  (except for the output neuron, i.e. the last neuron in the network) by the term (which we simply regard as a slightly more complicated activation function, for the moment)  $(\sigma(x_0 + \epsilon_0 \cdot x) - \sigma(x_0)) / (\epsilon_0 \cdot \sigma'(x_0))$  with parameter  $\epsilon_0$  which has to be fixed later. Each activation function  $\text{H}$  in the network is substituted by the term  $(\sigma(x/\epsilon_0) - l_\sigma) / (h_\sigma - l_\sigma)$  where  $\epsilon_0$  has to be chosen later. Denote by  $x_i$  the original neurons, by  $x_i^{\epsilon_0}$  the neurons which involve the above more complicated activation functions; thereby,  $x_i = x_i^{\epsilon_0}$  if the activation function is  $\sigma$ . We will now show that some  $\epsilon_0$  can be found such that the resulting network differs from the original one by no more than  $\delta/2$  for all inputs in  $(\mathcal{L}')^*$  with length at most  $h$ . The following observations are easy:

1. Given some neuron  $x_i$  with activation function  $\text{H}$ . Note that such a neuron is functionally dependent only on the label of a given entry  $l_v$  in the above network by construction, hence we can write  $x_i(l_v)$  as a shorthand notation. Given some  $\delta_1 > 0$ . Then  $\epsilon_0$  can be found such that  $|x_i(l_v) - x_i^{\epsilon_0}(l_v)| \leq \delta_1$  for all inputs  $l_v \in \mathcal{L}'$ . This follows immediately because of the uniform

convergence of  $(\sigma(x/\epsilon_0) - l_\sigma)/(h_\sigma - l_\sigma)$  to  $H$  if a neighborhood of 0 is not contained in the inputs.

2. Given some neuron  $x_i$  with linear activation function. Assume  $\vec{l}_v$  are the inputs of  $x_i$  directly referring to input entries, assume  $\vec{a} = (a_1, \dots, a_{i_0})$  are the additional inputs referring to outputs of other neurons. We write  $x_i(\vec{l}_v, \vec{a})$ , for convenience. Given some  $\delta_1 > 0$ . Then we can find  $\epsilon_0$  and  $\delta_2$  such that  $|x_i(\vec{l}_v, \vec{a}) - x_i^{\epsilon_0}(\vec{l}_v, \vec{a}')| \leq \delta_1$  for all  $\vec{a}, \vec{a}'$  with  $a_i, a'_i \in [-C_0 - \delta_0, C + \delta_0]$  with  $|a_i - a'_i| \leq \delta_2$  and  $\vec{l}_v$  labels in  $\mathcal{L}'$ . This follows from the fact that  $x_i$  is continuous, hence uniformly continuous on compacta, and  $(\sigma(x_0 + \epsilon \cdot x) - \sigma(x_0))/(\epsilon \cdot \sigma'(x_0))$  converges to id uniformly on compacta, hence  $x_i^{\epsilon_0}$  converges to  $x_i$  uniformly on compacta. We therefore find  $\epsilon_0$  and  $\delta_2$  such that

$$\begin{aligned} & |x_i(\vec{l}_v, \vec{a}) - x_i^{\epsilon_0}(\vec{l}_v, \vec{a}')| \\ & \leq |x_i(\vec{l}_v, \vec{a}) - x_i(\vec{l}_v, \vec{a}')| + |x_i(\vec{l}_v, \vec{a}') - x_i^{\epsilon_0}(\vec{l}_v, \vec{a}')| \\ & \leq \delta_1/2 + \delta_1/2 \end{aligned}$$

for  $|a_i - a'_i| \leq \delta_2$ .

3. Given some neuron  $x_i = x_i^{\epsilon_0}$  with activation function  $\sigma$ . Note that, by construction,  $x_i$  is functionally dependent only on outputs of other neurons. Denote, for convenience, the direct functional dependence by  $x_i(\vec{a})$ . Given some  $\delta_1 > 0$ . Then we can find  $\delta_2$  such that  $|x_i(\vec{a}) - x_i(\vec{a}')| \leq \delta_1$  for all  $\vec{a}, \vec{a}'$  with  $a_i, a'_i \in [-C_0 - \delta_0, C + \delta_0]$  and  $|a_i - a'_i| \leq \delta_2$ . This follows immediately from the fact that  $\sigma$  is continuous, hence uniformly continuous on compacta.

These properties allow us to substitute each single activation function and to control the effect of this substitution. We now formally unfold the RCC for all possible sequences of length at most  $h$ , obtaining several feedforward computations composed of functions computed by single neurons. For each such feedforward computation,

we start at the rightmost neuron and consider the neurons in the order of their appearance in this feedforward computation from right to left (thereby visiting the original neurons more than once since recurrence accounts for duplicates). The final output should deviate from the original one by at most  $\delta/2$  on sequences with entries in  $\mathcal{L}'$ . Starting with  $\delta_1 = \delta/2$ , we determine for each neuron the value  $\epsilon_0$  and a number  $\delta_2 > 0$  such that the following holds: if the inputs which come from other neurons deviate by no more than  $\delta_2$  and the activation function is substituted by the above term including  $\epsilon_0$ , the output deviates by at most  $\delta_1$ . For the three types of neurons, this can be achieved due to properties 1 – 3 as explained above. We can then continue with neurons to the left setting  $\delta$  to the minimum of the found  $\delta_2$  and  $\delta_0$ . (The latter bound  $\delta_0$  thereby ensures that values lie in the compactum  $[-C_0 - \delta_0, C_0 + \delta_0]$  on which uniform approximation is guaranteed.) So we collect a number  $\epsilon_0$  for each copy of a neuron such that this choice guarantees appropriate approximation of the whole function. We can now choose for the entire network the minimum of all values  $\epsilon_0$  as a uniform value which simultaneously fulfills all constraints.

Finally, it remains to show that the network which involves the above complicated activation functions can be interpreted as a RCC network with linear output and hidden neurons with activation function  $\sigma$ . Note that the above terms are both of the following form:

$$a \cdot \sigma(b \cdot x + c) + d$$

for real values  $a, b, c, d$ . We can simply integrate the term  $c$  into the bias of the respective neuron,  $b$  into the weights,  $a$  into the weights of all neurons which have direct connections from the neuron, and  $d$  into the bias of these neurons. Note that the last neuron of the RCC network (which does not possess a successor) is linear, hence no substitution has to be done at this part. Note, furthermore, that outgoing

connections include, of course, recurrent connections. (We do not have to deal with cycles because the integration of  $a$  and  $b$  is done only once for each connection of the network.) Hence the initial values for the empty sequence  $x_i(l_0)$  is affected by this change, too. We account for this fact resetting the term to  $x_i(l_0)/a - b$ .  $\square$

Hence RCC networks constitute universal approximators for sequences.

### 3.2 Recursive CC

Referring to the above proof, the essential steps to show approximation completeness for RecCC networks for input tree structures are **(I)** the construction of a unique linear code for tree structures over a finite set  $\mathcal{L}$ , and **(II)** the construction of a network with possibly multiplicative neurons and appropriate activation functions which can compute these codes. Then construction steps **(III)** to **(V)** can be transferred from the previous case.

The linear representation of trees is slightly more difficult than the representation of sequences since they are usually represented as abstract nonlinear data types. The prefix notation of trees offers one possibility which we use in the following. Assume  $\mathcal{L}$  is a finite set which does not contain the symbols ‘(’, ‘)’, and ‘,’. Assume  $\epsilon_\xi$  and  $f$  are additional symbols not contained in  $\mathcal{L}$ . Define for each tree  $t$  over  $\mathcal{L}$  with root vertex  $v$  and subtrees  $t_1$  and  $t_2$  the term-representation  $rep(t)$  recursively over the structure as follows:

$$\begin{aligned} rep(\xi_t) &= \epsilon_\xi, \\ rep(t) &= f(l(v), rep(t_1), rep_1(t_2)). \end{aligned}$$

This representation is obviously unique and hence step **(I)** is achieved.

Step **(II)** can be done as follows: assume  $\mathcal{L} = \{1, \dots, r\}$ , assume  $n_f, n_\xi$  are numbers not contained in  $\mathcal{L}$ , assume the maximum length of these numbers is  $L$

and assume the numbers are extended by leading entries 0 to occupy exactly  $L$  positions. Denote by  $nrep(t)$  the number which we obtain if we substitute in  $rep(t)$  the symbols  $f$  by  $n_f$  and  $\epsilon_\xi$  by  $n_\xi$  and extend the numbers from  $\mathcal{L}$  by leading entries 0. Then  $nrep(t)$  is unique, too. Note that it is not necessary to introduce symbols for ‘(’, ‘)’, and ‘,’ because we deal with only one function symbol. We now show that two multiplicative hidden neurons  $x_1$  and  $x_2$  of a RecCC network with the identity as activation function can be constructed which compute

$$x_1(t) = \exp_{0,1} \text{length}(nrep(t))$$

and

$$x_2(t) = 0.nrep(t)$$

where  $\text{length}$  denotes the length of the number including leading entries 0. We set

$$x_1(\xi_t) = 0.1^L, \quad x_2(\xi_t) = 0.n_\xi.$$

Assume the root vertex of a tree  $t$  is  $v$  and the two children are  $t_1$  and  $t_2$ . The functional dependence of  $x_1$  is

$$x_1(t) = \tau_1(l(v), x_1(t_1), x_1(t_2))$$

where by structural induction  $x_1(t_1)$  and  $x_1(t_2)$  represent the terms  $\exp_{0,1}(nrep(t_1))$  and  $\exp_{0,1}(nrep(t_2))$ . Hence the choice

$$\tau_1(a_1, a_2, a_3) = 0.1^{2L} \cdot a_2 \cdot a_3$$

gives the appropriate length. The functional dependence of  $x_2$  is

$$x_2(t) = \tau_2(l(v), x_1(t), x_1(t_1), x_2(t_1), x_1(t_2), x_2(t_2))$$

where by structural assumption  $x_2(t_1)$  and  $x_2(t_2)$  constitute  $0.nrep(t_1)$  and  $0.nrep(t_2)$ .

Hence the function

$$\tau_2(a_1, \dots, a_6) = 0.1^L \cdot n_f + 0.1^{2L} \cdot a_1 + 0.1^{2L} \cdot a_3 + 0.1^{2L} \cdot a_2 \cdot a_5$$

gives  $0.nrep(t)$ . Note that these functions can be computed by multiplicative neurons of order 2 with the linear activation function. Thus, in contrast to the case of sequences, we deal with multiplicative neurons in these approximation results.

Now we can proceed with steps **(III)**, **(IV)**, and **(V)** just as in the case of RCC networks and we obtain as a results, that Theorems 3.2, Theorem 3.3, and 3.4 also hold for tree structures instead of sequences, i.e. for functions  $f : \mathcal{L}_2^* \rightarrow \mathbb{R}$ , whereby the RecCC network has multiplicative neurons.

Due to the construction we use multiplicative neurons of order 2 in this case. In practical applications, usually standard neurons instead of multiplicative neurons are used. One can directly substitute the multiplication if, instead of single neurons, a feedforward network is attached in each recursive construction step, i.e.  $\tau_i$  may be computed by a feedforward network with a fixed number of neurons. Note that an activation function for which some point  $x_1$  exists such that  $\sigma$  is twice continuously differentiable in a neighborhood of  $x_1$  with nonvanishing second derivative  $\sigma''(x_1) \neq 0$  can be used to approximate the function  $x^2$  in the following way:

$$\frac{\sigma(x_1 + \epsilon \cdot x) + \sigma(x_1 - \epsilon \cdot x) - 2 \cdot \sigma(x_1)}{\epsilon^2 \cdot \sigma''(x_1)} \rightarrow x^2 \quad (\epsilon \rightarrow 0)$$

with uniform convergence on compacta. We refer to this property as  $\sigma$  being *locally*  $C^2$ . In addition, multiplication can be substituted by the square using the identity  $x \cdot y = ((x + y)^2 - (x - y)^2)/4$ . Note that the property locally  $C^2$  implies  $\sigma$  being locally  $C^1$ . Analogous to Theorem 3.4, we can thus substitute each function computed by a multiplicative neuron  $\tau_i$  by a feedforward network which in this case contains only one hidden layer with 8 hidden neurons in the hidden layer and activation function  $\sigma$ .<sup>7</sup>

---

<sup>7</sup>We do not specify this remark for trees but formalize the result later for the case of graph structures.

RecCC networks are usually not complete if they deal with DPAGs as inputs which are first transformed to a tree structure. However, some specific settings can be observed where uniqueness of representation is assured and hence approximation is possible such as the case of DPAGs such that no two vertices in a DPAG contain the same label. In this case, the label can also serve as a identifier of the vertex and the original graph can easily be reconstructed from the tree.

As beforehand, we have only considered supersource transductions so far. For tree structures IO-isomorphic transductions as a generalization of the more simple supersource transductions are interesting. This means that an input tree is mapped to an output tree of the same structure whereby the labels are substituted by values computed by the transduction. Thereby, the output label  $l'(v)$  of a vertex  $v$  might depend on the vertex  $v$  and the whole tree structure. We can use RecCC networks for IO-isomorphic transduction by substituting the label at each vertex by the output computed with the network for the subtree rooted at this vertex. It is an immediate consequence of the principled dynamics of RecCC networks and our previous approximation results that exactly those (measurable) supersource transductions can be approximated, for which a strict causality assumption applies: the output label of a vertex can depend on the subtree rooted at the vertex, but not on any other vertex of the tree structure. Since the approximation of functions which fulfill this causality assumption can be reduced to the problem to approximate a supersource transduction for any given subtree of the considered trees, the approximation completeness is obvious in this case. Conversely, the RecCC dynamics is based on this causality assumption with respect to functional dependencies of vertices. Thus, no functions violating the causality assumption can be approximated by a RecCC network.

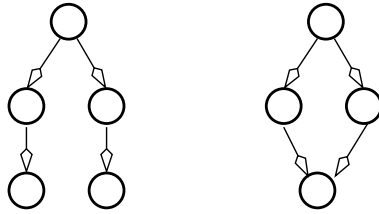


Figure 2: Example of two graph structures which cannot be differentiated by RecCC networks if the labels of the vertices at each layer are identical.

### 3.3 Contextual RecCC

CRecCC networks integrate contextual information since hidden neurons have access to the parents' activation of previous neurons. This possibility expands the functions which can be addressed with these networks to functions on graphs and IO-isomorphic transductions, as we will see. However, the situation is more complicated for DPAGs than for tree structures because it is not clear how to represent DPAGs in a linear code based on the information available within a contextual recursive neuron. We now discuss this issue in detail and we develop a code for a subset of DPAGs.

The context integrated in CRecCC neurons is of crucial importance. The available context of vertex  $v$  at neuron  $i$  can, for example, be measured in terms of the vertices of the DPAG which contribute directly or indirectly to  $x_i(v)$ . This context is precisely computed in (Micheli, Sona, and Sperduti, 2003a). As demonstrated in (Micheli, Sona, and Sperduti, 2003b; Micheli, 2003), this contextual information can in principle be used to compute certain transductions on graphs which require global knowledge of the structure and hence cannot be computed with simple recursive cascade correlation networks. Hence the functions which can be computed by RecCC networks on DPAGs form a proper subset of the functions which can be computed by CRecCC networks, e.g. the two structures depicted in Fig. 2 cannot

be differentiated by a RecCC network, but they can be differentiated by a CRecCC network. RecCC networks constitute universal approximators for tree structures. RecCC networks and CRecCC networks do not cover different function classes if restricted to tree structures of limited height. For DPAGs, CRecCC networks are strictly more powerful. Here we go a step further and identify a subset of DPAGs which fulfill a canonical structural constraint and we show that CRecCCs possess the universal approximation property for these structures. In addition, we construct a simple counterexample which shows that there exist structures outside this class which cannot be distinguished even by CRecCC networks regardless of the chosen functions of the neurons. As beforehand, we restrict ourselves to DPAGs where the fan-in and fan-out is restricted by two.

### 3.3.1 A counterexample

The question arises whether CRecCC network can differentiate all possible DPAG structures. The following example shows that this is not the case. However, this example requires a highly symmetric structure where different parts of the DPAG can neither be differentiated by the structure nor the labels of the vertices. Hence the example can be regarded as an ‘artificial’ example.

**Theorem 3.5** *There exist graphs in DPAG which cannot be distinguished by any CRecCC network.*

The counterexample is shown in Fig. 3. The formal proof that these two structures cannot be differentiated by a CRecCC network can be found in the appendix.

Note that this construction would not have been possible, if a different enumeration of children and parents had been chosen: consider edges  $(0, 4)$  and  $(0', 4')$ . 4 and  $4'$  constitute the second child of 0 and  $0'$ , respectively. Assume we had enumerated parents in such a way, that 0 and  $0'$  also constituted the *second parent* of 4

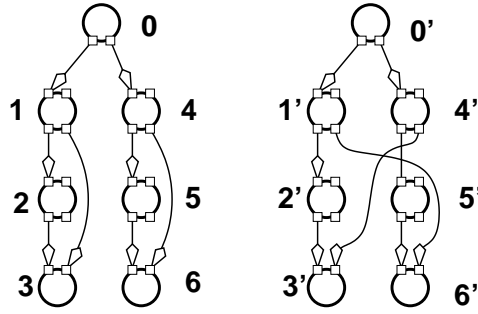


Figure 3: Example of two graphs which are mapped to identical values with CRecCC networks for a non-compatible enumeration of parents. Thereby, the enumeration of parents and children is indicated by boxes at the vertices, the left box being number one, the right box being number two.

and  $4'$ , respectively. Then the proof for the counterexample in Theorem 3.5 would no longer hold: the vertices 1 and 4 could then be differentiated by an appropriate choice of the weights and activation functions of the neurons, and the same would hold for  $1'$  and  $4'$ . This can be seen e.g. due to the activation of the second hidden neuron whose functional dependence is:

$$x_2(1) = \tau_2(l, x_1(1), x_1(2), x_2(2), x_1(3), x_2(3), x_1(0), 0_p^1) \neq \tau_2(l, x_1(1), x_1(2), x_2(2), x_1(3), x_2(3), 0_p^1, x_1(0)) = x_2(4).$$

This inequality holds for appropriate  $\tau_2$  because the inputs  $0_p^1$  and  $x_1(0)$  are given at different positions. The possibility to differentiate these two vertices leads to the possibility to differentiate all vertices  $i$  in one of the structures, and finally to the possibility to differentiate the two structures since the vertices are connected in a different way in the two structures. This example leads to a simple structural condition which ensures that different structures can be differentiated by CRecCC networks.

### 3.3.2 Identification of a subclass of DPAGs

Motivated by the previous example, we now identify one subclass of DPAGs for which such an example cannot be found and which allow a linear encoding of the structures. The following property constitutes *one sufficient condition* to ensure this property, it is not a necessary condition. However, it has the advantage that it can easily be tested and it does often not put severe constraints on the respective application.

**Definition 3.6** *Assume  $\mathcal{D}$  is a DPAG. We say that the ordering of the parents and children is compatible if for each edge  $(u, v)$  of  $\mathcal{D}$  the equality*

$$P_v(u) = C_u(v)$$

*holds. This property states that if vertex  $u$  is enumerated as parent number  $i$  of  $v$  then  $v$  is enumerated as child number  $i$  of  $u$  and vice versa. We call the subset of DPAGs with compatible positional ordering rooted directed bipositional acyclic graphs (DBAGs).*

Note that the above counterexample used elements in  $\text{DPAG} \setminus \text{DBAG}$ . A different enumeration as DBAG would have been possible in this case, such that the structures could be distinguished. We now investigate in which sense the requirement on the presence of a compatible enumeration puts restrictions on the considered structures. Due to the definition, DPAGs with compatible positioning necessarily have the same fan-in and fan-out. However, we can always extend a DPAG with fan-in  $k_1 \neq$  fan-out  $k_2$  by empty children and parents, respectively, such that the fan-in and fan-out coincide. Moreover, the following theorem shows that any given DPAG with limited fan-in and fan-out two can always be transformed into a DBAG via reenumeration of children and parents.

**Theorem 3.7** *Given a DPAG  $\mathcal{D}$  with limited fan-in and fan-out two, then we can find an enumeration  $P'_v$  and  $C'_v$  of parents and children for each vertex  $v$  such that the resulting DPAG constitutes a DBAG with respect to the enumerations  $P'_v$  and  $C'_v$  for vertices  $v$  of the structure.*

The proof can be found in the appendix. This proof holds only for fan-in and fan-out two. For larger fan-in and fan-out, an analogous result can be stated if, in addition, the vertices might be expanded by empty children and parents.

**Theorem 3.8** *Assume  $\mathcal{D}$  is a DPAG with fan-in and fan-out limited by at most  $k$ . Then we can find a DBAG which arises from  $\mathcal{D}$  by an expansion of the vertices of  $\mathcal{D}$  with empty children and parents to fan-in and fan-out  $2k - 1$  and reenumeration of children and parents.*

The proof is provided in the appendix. We conjecture that  $\mathcal{D}$  can also be transformed to a DBAG via reenumeration of children and parents with fan-in and fan-out  $k$  and without expanding. However, we could not find a proof, neither a counterexample for this conjecture.

In applications such as chemistry, positioning of children and parents is often done only by convention without referring to semantic implications such as in the case of directed acyclic graphs (without prior positioning). Hence restricting to DBAGs does not limit the applicability of CRecCC in these cases. In alternative scenarios such as logic, positioning might be crucial and it might not be possible to change the positioning without affecting the semantic. A simple example for this fact is the term `implies(loves(Bill,Mary),smiles_at(Bill,Mary))`, or, as a whole sentence ‘If Bill loves Mary then Bill smiles at Mary’. This cannot be transferred into a DBAG unless ‘Bill’ becomes the first child of the first symbol ‘loves’ and the second child of the second symbol ‘smiles ’ or vice versa. I.e. we would obtain ‘If

Bill loves Mary then Mary smiles at Bill’ or ‘If Mary loves Bill then Bill smiles at Mary’, both nice sentences, however, with a different meaning than the original one. We will later see that alternative guarantees for the universal approximation ability can be found and a large number of structures in DPAG\DBAG can be appropriately processed as well. This alternative characterization is not as straightforward as the restriction to DBAGs which constitutes a purely structural property and can easily be tested.

### 3.3.3 Encoding of DBAGs and certain DPAGs

Next we introduce a recursive mapping of vertices in a DPAG to terms which describes information contained in the DPAG. The purpose of this transformation is to obtain linear representations of DPAGs. It will turn out that these linear representations characterize DBAGs up to a certain height uniquely. Naturally, the representation characterizes also a large number of DPAGs uniquely and approximation completeness will hold for these DPAGs, too. However, the fact whether this holds for any given set of DPAGs needs to be tested for each application. For DBAGs it is always valid because of the structural restrictions.

Assume  $\mathcal{L}$  is a finite set which does not contain the symbols ‘(’, ‘)’, and ‘,’. Assume  $\epsilon_c$ ,  $\epsilon_p$ , and  $f$  are additional symbols not contained in  $\mathcal{L}$ . Define for each  $i \geq 0$  and vertex  $v$  in a given DPAG the term-representation  $rep_i(v)$  recursively over  $i$  and the structure as follows:

$$rep_i(\xi_c) = \epsilon_c, rep_i(\xi_p) = \epsilon_p,$$

$$rep_1(v) = f(l(v), rep_1(ch_1(v)), rep_1(ch_2(v))),$$

$$rep_i(v) = f(l(v), rep_i(ch_1(v)), rep_i(ch_2(v)), rep_{i-1}(pa_1(v)), rep_{i-1}(pa_2(v))),$$

for  $i > 1$ .

This representation contains *all* information available for a CRecCC network, since the following result holds:

**Theorem 3.9** *Assume  $\mathcal{D}$  and  $\mathcal{D}'$  are DPAGs which can be differentiated by hidden neuron  $h$  of some CRecCC network. Then  $rep_h(\text{root}(\mathcal{D}))$  and  $rep_h(\text{root}(\mathcal{D}'))$  are different.*

*Proof.* The functional dependence of CRecCC neurons has the form

$$\begin{aligned} x_i(v) = & \tau_i(l(v), x_1(v), \dots, x_{i-1}(v), \\ & x_1(ch_1(v)), \dots, x_i(ch_1(v)), x_1(ch_2(v)), \dots, x_i(ch_2(v)), \\ & x_1(pa_1(v)), \dots, x_{i-1}(pa_1(v)), x_1(pa_2(v)), \dots, x_{i-1}(pa_2(v))) \end{aligned}$$

Hence all available information for neuron  $i$  is contained in the term which we obtain if we evaluate  $l(v)$  to the respective label and all other terms to symbols in the corresponding Herbrand-algebra. We denote the evaluation  $l(v)$  by the same symbol, for simplicity.  $x_i(\xi_p)$  is evaluated to the constant  $0_p^i$  and  $x_i(\xi_c)$  is evaluated to the constant  $0_c^i$ . We denote the resulting string for  $x_i(v)$  by  $rep'_i(v)$ . Two DPAGs which can be differentiated by hidden neuron  $h$  lead to different representations  $rep'_h(\text{root}(\mathcal{D}))$  and  $rep'_h(\text{root}(\mathcal{D}'))$ . We now show that if for vertices  $v$  of  $\mathcal{D}$  and  $v'$  of  $\mathcal{D}'$  and some  $i$  the terms  $rep'_i(v)$  and  $rep'_i(v')$  differ then so do  $rep_i(v)$  and  $rep_i(v')$ .

This is done by simultaneous induction over  $i$  and the structure of the terms: if one of the vertices  $v$  or  $v'$  is an empty child or parent, we get by definition  $0_p^i$  or  $0_c^i$  for the representation  $rep'_i$ .  $rep'_i$ , however, also gives a unique identifier  $\epsilon_p$  or  $\epsilon_c$  for the empty child or parent, respectively. So we can assume that  $v$  and  $v'$  are not empty.

$i = 1$ : We obtain

$$rep'_1(v) = \tau_1(l(v), rep'_1(ch_1(v)), rep'_1(ch_2(v)))$$

and

$$rep'_1(v') = \tau_1(l(v'), rep'_1(ch_1(v')), rep'_1(ch_2(v'))),$$

hence one of the three subterms is different: it holds  $l(v) \neq l(v')$ , or  $rep'_1(ch_1(v)) \neq rep'_1(ch_1(v'))$ , or  $rep'_1(ch_2(v)) \neq rep'_1(ch_2(v'))$ . In the latter two cases we can assume by structural assumption that  $rep_1(ch_1(v)) \neq rep_1(ch_1(v'))$  or  $rep_1(ch_2(v)) \neq rep_1(ch_2(v'))$  holds.  $rep_1$  evaluates to

$$rep_1(v) = f(l(v), rep_1(ch_1(v)), rep_1(ch_2(v)))$$

and

$$rep_1(v') = f(l(v'), rep_1(ch_1(v')), rep_1(ch_2(v'))),$$

and gives therefore also two different terms.

Inductive step for  $i$ : We find

$$\begin{aligned} rep'_i(v) = & \tau_i(l(v), rep'_1(v), \dots, rep'_{i-1}(v), \\ & rep'_1(ch_1(v)), \dots, rep'_i(ch_1(v)), rep'_1(ch_2(v)), \dots, rep'_i(ch_2(v)), \\ & rep'_1(pa_1(v)), \dots, rep'_{i-1}(pa_1(v)), rep'_1(pa_2(v)), \dots, rep'_{i-1}(pa_2(v))) \end{aligned}$$

and a similar term for  $rep'_i(v')$ . If these terms are different, one of the subterms differs. Hence one of the following holds:

1.  $l(v) \neq l(v')$ ,
2.  $rep'_j(ch_1(v)) \neq rep'_j(ch_1(v'))$  for some  $j \leq i$ ,
3.  $rep'_j(ch_2(v)) \neq rep'_j(ch_2(v'))$  for some  $j \leq i$ ,
4.  $rep'_j(pa_1(v)) \neq rep'_j(pa_1(v'))$  for some  $j < i$ ,
5.  $rep'_j(pa_2(v)) \neq rep'_j(pa_2(v'))$  for some  $j < i$ .

In cases 2 and 3 we can assume by structural assumption that also  $rep_j$  gives different values for the respective terms, in cases 4 and 5 we can assume this fact because of the inductive hypothesis. If  $rep_j$  yields different values for two terms, then so does  $rep_{j'}$  for all  $j' \geq j$  as we will show afterwards (Lemma 3.10). Hence we find that the terms

$$rep_i(v) = f(l(v), rep_i(ch_1(v)), rep_i(ch_2(v)), rep_{i-1}(pa_1(v)), rep_{i-1}(pa_2(v)))$$

and

$$rep_i(v') = f(l(v'), rep_i(ch_1(v')), rep_i(ch_2(v')), rep_{i-1}(pa_1(v')), rep_{i-1}(pa_2(v')))$$

are also different. □

We postponed in Theorem 3.9 the proof of the following lemma:

**Lemma 3.10** *Assume  $\mathcal{D}$  and  $\mathcal{D}'$  are DPAGs with vertex  $v$  in  $\mathcal{D}$  and  $v'$  in  $\mathcal{D}'$ , respectively. Assume  $rep_i(v) \neq rep_i(v')$  for some  $i$ . Then  $rep_j(v) \neq rep_j(v')$  for all  $j \geq i$ .*

*Proof.* The proof is by simultaneous induction over  $j$  and the structure. For  $j = 1$ ,  $i = 1$  also holds and the result is obvious.

Inductive step for  $j$ : If one of the vertices, say  $v$  is empty, the result is obvious because  $rep_j(v)$  yields  $\epsilon_p$  or  $\epsilon_c$ , respectively.  $rep_j(v')$  yields a different value by definition for all  $j$  unless  $v'$  also constitutes an empty child or parent.

Assume the vertices are both not empty. Assume w.l.o.g. that  $j > i$ . Assume  $rep_i(v) \neq rep_i(v')$ . The representations are of the form

$$rep_i(v) = f(l(v), rep_i(ch_1(v)), rep_i(ch_2(v)), rep_{i-1}(pa_1(v)), rep_{i-1}(pa_2(v)))$$

and

$$rep_i(v') = f(l(v'), rep_i(ch_1(v')), rep_i(ch_2(v')), rep_{i-1}(pa_1(v')), rep_{i-1}(pa_2(v')))$$

(The last two subterms are missing for  $i = 1$ .) Hence one of the subterms differs, i.e.  $l(v) \neq l(v')$ ,  $rep_i(ch_1(v)) \neq rep_i(ch_1(v'))$ ,  $rep_i(ch_2(v)) \neq rep_i(ch_2(v'))$ ,  $rep_{i-1}(pa_1(v)) \neq rep_{i-1}(pa_1(v'))$ , or  $rep_{i-1}(pa_2(v)) \neq rep_{i-1}(pa_2(v'))$ . In cases 2 and 3 we can assume by structural assumption that the same holds also for the index  $j$ , in the last two cases we can assume by the inductive hypothesis that the same holds also for the index  $j - 1$ . In all cases we obtain at least one subterm of  $rep_j(v)$  which is different from the respective subterm of  $rep_j(v')$  hence  $rep_j(v) \neq rep_j(v')$  holds.  $\square$

Therefore, CRecCC networks can constitute universal approximators at most on DPAGs for which  $rep_i$  yields a unique representation for some  $i$  by definition of the functional dependencies. We will now show that the mapping which maps DBAGs of height at most  $h$  to  $rep_{h+1}(v)$ ,  $v$  denoting the root of the respective DBAG, is injective. As a consequence of this fact, all information available in a given DBAG is provided to a CRecCC network via the defined dynamics. For this purpose, we describe an algorithm which allows to recover the original DBAG from the respective linear representation. Note that each term  $rep_i(v)$  can uniquely be decomposed into its subterms. The reconstruction is in several steps.

**Definition 3.11** *Assume a path of the DBAG  $\mathcal{D}$  which starts at the root is given which consecutively visits the vertices  $v_{i_0}, \dots, v_{i_h}$ . A linear path representation consists in a string  $s = s_1 \dots s_{i_h}$ , where  $s_j \in \{1, 2\}^*$  equals*

$$s_j = C_{v_{i_{j-1}}}(v_{i_j})$$

for all  $j = 1, \dots, h$ .

Hence a linear path representation collects which child of the previously visited vertex is visited at each position in the path.  $s_i = 1$  indicates that the left child of

the actual vertex in the path is visited,  $s_i = 2$  indicates that the next vertex is given by the right child. We assume the convention that the empty string represents the path of length 0 which stays at the root. Obviously, the mapping of rooted paths to linear path representations is injective.

**Lemma 3.12** *For any vertex  $v$  of height at most  $i$  in a DBAG  $\mathcal{D}$ , it is possible to recover the linear path representation of all paths from the root of  $\mathcal{D}$  to  $v$  from  $rep_{i+1}(v)$ .*

*Proof.* The proof is via induction. If  $i = 1$ ,  $v$  can only be the root of the DBAG with linear path representation  $\epsilon$  of the path of length 0. If  $i > 1$ , then we find

$$rep_{i+1}(v) = f(\dots, rep_i(pa_1(v)), rep_i(pa_2(v))).$$

Note that we can decide whether the left or right parent is empty since  $\epsilon_p$  is a symbol not contained in  $\mathcal{L}$ . If a parent is not empty, we can recover the linear path representation of all paths from the root to  $pa_1(v)$  and  $pa_2(v)$  from  $rep_i(pa_1(v))$  or  $rep_i(pa_2(v))$ , respectively, by the inductive hypothesis, because the height of  $pa_1(v)$  and  $pa_2(v)$  is smaller than  $i$ . Note that the DBAG has a compatible positioning of children and parents. Hence we obtain the linear path representation of all paths from the root to  $v$  in the following way: we extend the linear path representation of the paths to  $pa_1(v)$  (if this parent is existing) by the letter 1. To this set we add the linear path representation of the paths to  $pa_2(v)$  (if existing) extended by the letter 2. □

The above lemma tells us that we can recover all paths from the root to a vertex from the given linear representation. Note that the linear path representations of a rooted path to a vertex differs for each vertex. The next lemma extends this result to the extraction of all vertices and their respective paths from the linear representation.

**Lemma 3.13** *For a DBAG  $\mathcal{D}$  of height at most  $h$ , it is possible to extract from  $rep_{h+1}(root(\mathcal{D}))$  the set  $S = \{(l(v), B) \mid v \in vert(\mathcal{D}), B \subset \{1, 2\}^*, B \text{ is ordered lexicographically, } B \text{ contains the linear path representation of all paths from the root to } v \text{ in } \mathcal{D}\}$ .*

*Proof.* For an empty DBAG, the term  $rep_{h+1}(root(\mathcal{D}))$  equals  $\epsilon_c$  and we simply extract the empty set. Otherwise, we can start with the empty set  $S$  and iteratively add tuples  $(l(v), B)$  to the set  $S$  which collect all vertices  $v$  of the DBAG and the set of linear path representations of the vertex. This starts from the root and the given term  $rep_{h+1}(root(\mathcal{D}))$ , and recursively extracts vertices and subterms which describe all paths for these vertices until we arrive at the leaves.

The procedure is as follows: Note that all vertices in the DBAG have height at most  $h$  by assumption. Given the term

$$rep_{h+1}(v) = f(l(v), rep_{h+1}(ch_1(v)), rep_{h+1}(ch_2(v)), \dots)$$

we recover the vertex  $v$ : its label  $l(v)$  can be obtained from the term  $rep_{h+1}(v)$ . All linear path representation of all paths from the root to  $v$  can be recovered because of Lemma 3.12. We sort the linear path representations lexicographically getting  $B$ , and add the tuple  $(l(v), B)$  to the set  $S$ . Note that this tuple might already be contained in  $S$  if vertex  $v$  can be reached by more than one path from the root. In this case,  $S$  is not changed since we take the union and since the linear path representation of a path to a vertex characterizes the vertex uniquely. We can decide, given  $rep_{h+1}(v)$ , whether a left and/or right child of  $v$  exists, because empty children are represented by the unique symbol  $\epsilon_c$  as second or third subterm of  $rep_{h+1}(v)$ . If nonempty children exist, we continue with each of the two children and the representation  $rep_{h+1}(ch_1(v))$  and/or  $rep_{h+1}(ch_2(v))$ , respectively. Both representations can be obtained from  $rep_{h+1}(v)$ . This iterative step yields all vertices which are

direct or indirect successors of the previous vertex  $v$ , thus the entire structure will be visited during this iterative process, starting from the root of  $\mathcal{D}$ .  $\square$

Hence we finally get the uniqueness of the linear representation for DBAGs:

**Theorem 3.14** *The mapping which maps DBAGs  $\mathcal{D}$  over  $\mathcal{L}$  of height at most  $h$  to  $rep_{h+1}(root(\mathcal{D}))$  is injective.*

*Proof.* Because of Lemma 3.13 we can recover from the given linear representation for any DBAG  $\mathcal{D}$  of height at most  $h$  and each vertex of  $\mathcal{D}$  a tuple consisting of the label of the vertex and the linear path representations of all paths from the root to the respective vertex. This information uniquely characterizes  $\mathcal{D}$  because the DBAG can easily be reconstructed using this information: we introduce a vertex for each tuple and assign the given label to it. The problem is, of course, that linear path representations of paths do not include the identifier for the vertices which we have just chosen. However, the root is uniquely characterized by the fact that the only path to the root is empty. We can then iteratively add all edges to the DBAG and add the positioning of children and parents which are implicitly contained in the linear path representations as follows: we start with the disconnected vertices, and iteratively add connections starting from the root up to the leaves. In an iterative step, we consider all paths starting from the root in the already constructed part, and we choose a shortest path in the linear path representations of vertices of which not yet all edges are included (note that it is possible to test this property since we can identify the root and afterwards follow the successors given by the identifiers of the linear path representation). Because we choose the shortest path, only the last edge of the path, say  $(u, v)$ , is not yet included in the DBAG. We can uniquely identify  $u$  via the first part of the linear path representation. The identifier of  $v$  is known (since the path is contained in the tuple which describes  $v$ ), and hence we can connect  $u$

and  $v$  in the DBAG, i.e. introduce the edge and assign the positioning as indicated in the linear path representation to  $u$  and  $v$ . Since DBAGs are considered, the positioning of children, which is given in the linear path representation, also gives the positioning of parents.  $\square$

The representation  $rep_i$  preserves all information available for a CRecCC network as we have shown in Theorem 3.9. Hence CRecCC networks with appropriate choice of the weights and activation function (such that different terms  $rep_i$  are mapped to different values also by a CRecCC network) can exactly map all DPAGs appropriately for which  $rep_i$  yields a unique representation for large enough  $i$ . We will later show that multiplicative CRecCC neurons with a standard activation function can perform this task. Since DBAGs can uniquely be reconstructed from  $rep_i$ , CRecCC networks are complete with respect to DBAGs. Note that the reconstruction of a DPAG from the encoding  $rep_i$  is in general not possible because forward paths in a DPAG together with the respective positioning of the vertices (i.e. linear path representations) cannot be obtained from backward connections from the children towards its parents, as has been done in Lemma 3.12. Thereby, backward connections are connections from a vertex to a parent (together with the respective positioning) and forward connections are connections from a vertex to a child (together with the respective positioning). Nevertheless we can extract all paths which start at the root, follow a certain number of forward and/or backward connections, and end up at some vertex even for a DPAG. This is possible because a forward connection starting from  $v$  can be obtained from  $rep_i(ch_1(v))$  and  $rep_i(ch_2(v))$ , while a backward connection can be obtained from  $rep_{i-1}(pa_1(v))$  and  $rep_{i-1}(pa_2(v))$ , whereby full information is available for large enough  $i$ . Hence  $rep_i$  can differentiate for large enough  $i$  between any two DPAGs such that at least one path starting from the root exists (following forward and/or backward connections) which leads

to a vertex with different label or different number of children or parents in the two DPAGs. Theorem 3.5 shows that this is not always possible. However, we claim that the capacity of CRecCC networks to differentiate between these DPAGs (and to approximate functions on these DPAGs as we will see later) is sufficient for practical applications because these highly symmetrical structures usually do not occur in practice.

### 3.3.4 Approximation with a CRecCC network

We will now show that specific CRecCC networks can ‘compute’ the representation  $rep_i$  in some form. Hence they can uniquely encode DBAGs (and DPAGs for which the encoding is unique for some  $i$ ) into a real vector. This gives us step **(II)** of the general proof of approximation completeness. We restrict the presentation in the next section to DBAGs, for convenience, keeping in mind that similar constructions can be done for certain DPAGs, too.

**Lemma 3.15** *Assume  $\mathcal{L} = \{1, \dots, r\}$ . Assume  $h \in \mathbb{N}$ . Then a CRecCC network with multiplicative neurons of order at most 4 and linear activation function with  $2(h + 1)$  hidden neurons can be found such that the output  $x_{2(h+1)}(\text{root}(\mathcal{D}))$  yields unique representations of DBAGs  $\mathcal{D}$  over  $\mathcal{L}$  of height at most  $h$ .*

*Proof.* Fix different numbers  $n_f, n_., n_{(, n_p},$  and  $n_c$  not contained in  $\{1, \dots, r\}$ . Assume that 0 is not contained in this set of numbers. Assume  $L$  is the maximum length of the digital representation of the above symbols (including elements of  $\mathcal{L}$ ). We assume in the following, that all numbers are extended by leading entries 0 such that they occupy precisely  $L$  places. We can then uniquely represent the term  $rep_i(v)$  by a number: we set  $n_c$  for  $\epsilon_c,$   $n_p$  for  $\epsilon_p.$  A general term  $rep_i(v)$  is represented by the number which we obtain substituting in  $rep_i(v)$  the symbols ‘ $f$ ’ by  $n_f,$  ‘(’ by

$n_(< ; >$  by  $n_(< >$  by  $n_>$ ),  $l(v)$  by the respective element of  $\mathcal{L}$ , and the contained references to  $rep$  by the recursively constructed numbers (always including leading entries 0). We refer to the number which represents  $rep_i(v)$  by the symbol  $nrep_i(v)$ .

We will now iteratively construct hidden neurons  $x_{2i-1}$  and  $x_{2i}$  for  $i \geq 1$  such that

$$x_{2i-1}(v) = \exp_{0,1}(\text{length}(nrep_{i-1}(v))) \text{ and } x_{2i}(v) = 0.nrep_{i-1}(v).$$

The latter term denotes the decimal number smaller than 1 with digits  $nrep_{i-1}(v)$ ,  $\text{length}(nrep_{i-1}(v))$  denotes the length of the term including leading entries 0. Since the representation  $rep_{h+1}(\text{root}(\mathcal{D}))$  is unique for DBAGs of height at most  $h$ , the output  $x_{2(h+1)}(\text{root}(\mathcal{D}))$  of hidden neuron  $2(h+1)$  gives then unique values for these DBAGs.

Per definition,  $x_i(\xi_p) = 0_p^i$  and  $x_i(\xi_c) = 0_c^i$  are fixed symbols for all  $i$ . We choose  $0_p^i = 0_c^i = (0.1)^L$  for odd  $i$  and  $0_p^i = 0.n_p$  and  $0_c^i = 0.n_c$  for even  $i$ . Assume in the following that  $v$  is nonempty. For the first hidden neuron, we find  $x_1(v) = \tau_1(l(v), x_1(ch_1(v)), x_1(ch_2(v)))$ . Assume  $l_1$  and  $l_2$ , respectively, denote the length of  $nrep_1$  of the first and second child, respectively, of  $v$ . Since the term  $rep_1(v) = f(l(v), rep_1(ch_1(v)), rep_1(ch_2(v)))$  consists of 6 symbols and two subterms which represent  $ch_1(v)$  and  $ch_2(v)$ , the length of  $rep_1(v)$  is given by  $6L + l_1 + l_2$ . Hence the choice

$$\tau_1(a_1, a_2, a_3) = (0.1)^{6L} \cdot a_2 \cdot a_3$$

gives us the desired output, since via structural induction the latter two entries can be identified with  $a_2 = (0.1)^{l_1}$  and  $a_3 = (0.1)^{l_2}$ . Note that this function can be computed by a multiplicative neuron of order 2 with linear activation function. The functional dependence of the second hidden neuron is  $x_2(v) = \tau_2(l(v), x_1(v), x_1(ch_1(v)), x_2(ch_1(v)), x_1(ch_2(v)), x_2(ch_2(v)), x_1(pa_1(v)), x_1(pa_2(v)))$ . By struc-

tural induction we can assume that  $x_2(ch_1(v))$  and  $x_2(ch_2(v))$  contain the number representation of  $rep_1$  of the two children. As shown beforehand,  $x_1(ch_1(v))$  and  $x_1(ch_2(v))$  represent the length of  $rep_1(ch_1(v))$  and  $rep_1(ch_2(v))$ , respectively.

Hence the function

$$\begin{aligned} \tau_2(a_1, \dots, a_8) = & \\ & (0.1)^L n_f + (0.1)^{2L} n_c + (0.1)^{3L} a_1 + (0.1)^{4L} n_1 + (0.1)^{4L} a_4 + (0.1)^{5L} a_3 \cdot n, \\ & + (0.1)^{5L} a_3 \cdot a_6 + (0.1)^{6L} a_3 \cdot a_5 \cdot n) \end{aligned}$$

computes the desired number representing  $nrep_1(v)$ . Thereby, concatenation of digits which represent the symbols in  $rep_1(v)$  is done by summing the appropriately shifted numbers. Note that the length of the already computed subnumbers representing children is available. This function  $\tau_2$  can be computed by a multiplicative neuron of order 2 with linear activation function.

Assume we have constructed hidden neurons up to number  $2i$ . It is

$$rep_{i+1}(v) = f(l(v), rep_{i+1}(ch_1(v)), rep_{i+1}(ch_2(v)), rep_i(pa_1(v)), rep_i(pa_2(v)))$$

The functional dependence of hidden neuron number  $2i + 1$  is

$$\begin{aligned} x_{2i+1}(v) = & \tau_{2i+1}(l(v), x_1(v), \dots, x_{2i}(v), \\ & x_1(ch_1(v)), \dots, x_{2i+1}(ch_1(v)), x_1(ch_2(v)), \dots, x_{2i+1}(ch_2(v)), \\ & x_1(pa_1(v)), \dots, x_{2i}(pa_1(v)), x_1(pa_2(v)), \dots, x_{2i}(pa_2(v))) \end{aligned}$$

By induction,  $x_{2i-1}(\ast)$  yields the term  $\exp_{0.1}(\text{length}(nrep_i(\ast)))$ , where  $\ast$  is one of  $ch_1(v)$ ,  $ch_2(v)$ ,  $pa_1(v)$  and  $pa_2(v)$ . By structural induction, the inputs  $x_{2i+1}(ch_1(v))$  and  $x_{2i+1}(ch_2(v))$ , respectively, give the terms  $\exp_{0.1}(\text{length}(nrep_{i+1}(ch_1(v))))$  and  $\exp_{0.1}(\text{length}(nrep_{i+1}(ch_2(v))))$ . Since  $rep_{i+1}$  consists of 8 symbols and it further contains the subterms which describe the length of the representations of children and parents, the function

$$\tau_{2i+1}(a_1, \dots, a_{10i+3}) = (0.1)^{8L} \cdot a_{4i+2} \cdot a_{6i+3} \cdot a_{8i+2} \cdot a_{10i+2}$$

computes  $\exp_{0.1}(\text{length}(\text{nrep}_{i+1}(v)))$ . Note that this function can be computed by a multiplicative neuron of order 4 with linear activation function. The functional dependence of hidden neuron  $2i + 2$  is

$$\begin{aligned} x_{2i+2}(v) = & \tau_{2i+2}(l(v), x_1(v), \dots, x_{2i+1}(v), \\ & x_1(\text{ch}_1(v)), \dots, x_{2i+2}(\text{ch}_1(v)), x_1(\text{ch}_2(v)), \dots, x_{2i+2}(\text{ch}_2(v)), \\ & x_1(\text{pa}_1(v)), \dots, x_{2i+1}(\text{pa}_1(v)), x_1(\text{pa}_2(v)), \dots, x_{2i+1}(\text{pa}_2(v))) \end{aligned}$$

$x_{2j-1}(\ast)$  yields the term  $\exp_{0.1}(\text{length}(\text{nrep}_j(\ast)))$  for  $j \leq i + 1$  and  $\ast \in \{\text{ch}_1(v), \text{ch}_2(v), \text{pa}_1(v), \text{pa}_2(v)\}$ .  $x_{2(i+1)}(\ast)$  yields the term  $0.\text{nrep}_{i+1}(\ast)$  for  $\ast \in \{\text{pa}_1(v), \text{pa}_2(v)\}$ .  $x_{2j}(\ast)$  gives the term  $0.\text{nrep}_j(\ast)$  for  $j \leq i + 1$  and  $\ast \in \{\text{ch}_1(v), \text{ch}_2(v)\}$ .

Hence the function

$$\begin{aligned} & \tau_{2i+2}(a_1, \dots, a_{10i+8}) \\ = & (0.1)^L n_f + (0.1)^{2L} n_c + (0.1)^{3L} a_1 + 0.1^{4L} n, \\ + & (0.1)^{4L} a_{4i+4} + (0.1)^{5L} a_{4i+3} \cdot n, \\ + & (0.1)^{5L} a_{4i+3} \cdot a_{6i+6} + (0.1)^{6L} a_{4i+3} \cdot a_{6i+5} \cdot n, \\ + & (0.1)^{6L} a_{4i+3} \cdot a_{6i+5} \cdot a_{8i+6} + (0.1)^{7L} a_{4i+3} \cdot a_{6i+5} \cdot a_{8i+5} \cdot n, \\ + & (0.1)^{7L} a_{4i+3} \cdot a_{6i+5} \cdot a_{8i+5} \cdot a_{10i+7} + (0.1)^{8L} a_{4i+3} \cdot a_{6i+5} \cdot a_{8i+5} \cdot a_{10i+6} \cdot n \end{aligned}$$

computes the representation  $0.\text{nrep}_{i+1}(v)$ . Note that this function can be computed by a multiplicative neuron of order 4 with linear activation function.  $\square$

A similar construction is possible if fan-in and fan-out  $k \geq 2$  is considered. Then products of at most  $2k$  factors are to be dealt with, hence CRecCC networks with multiplicative neurons of order at most  $2k$  can in this case compute a unique representation.

It follows in the same way as Lemma 3.1 that integration of feedforward networks into a CRecCC network is possible. In analogy to Theorem 3.3 this can be extended to real-valued labels, and, mimicking the proof of Theorem 3.4, we can

substitute specific activation functions by standard ones. Hence CRecCC networks with multiplicative neurons constitute universal approximators for structures analogous to Theorem 3.4. This result has been proved for DBAGs with limited fan-in and fan-out two. It can immediately be generalized to larger fan-in and fan-out  $k$  giving us approximation completeness for these DBAGs for CRecCC networks with multiplicative neurons of order  $2k$ .

In practical applications, usually standard neurons instead of multiplicative neurons are used. In analogy to the case of trees, we can substitute multiplicative neurons by networks of standard neurons in the following way:

**Theorem 3.16** *Assume  $\mathcal{L} = \mathbb{R}^n$ . Assume  $P$  is some probability measure on DBAGs over  $\mathcal{L}$ . Assume  $f : \text{DBAG}_{\mathcal{L}}^2 \rightarrow \mathbb{R}$  is a Borel-measurable function. Assume  $\epsilon > 0$ ,  $\delta > 0$ . Assume  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  is a continuous and locally  $C^2$  squashing function. Then there exists a CRecCC network with the following properties: it has a linear activation function for the output neuron. The function of the other hidden states  $\tau_i$  are computed by a feedforward network with at most two hidden layers and a universally bounded number of neurons in the network with activation functions  $\sigma$ . This CRecCC network computes a function  $g$  such that*

$$P(x \in \text{DBAG}_{\mathcal{L}}^2 \mid |f(x) - g(x)| > \delta) < \epsilon.$$

*Proof.* The network which we constructed in Theorem 3.4 uses multiplicative neurons because of the computation of the linear codes of DBAGs as constructed in Lemma 3.15. The multiplicative neurons which are used in this construction contain multiplications of order at most 4. We can substitute one multiplication  $x \cdot y$  by the term  $((x + y)^2 - (x - y)^2)/4$ . The function  $x \rightarrow x^2$  can be approximated uniformly by the term  $(\sigma(x_1 + \epsilon_0 \cdot x) + \sigma(x_1 - \epsilon_0 \cdot x) - 2 \cdot \sigma(x_1))/(\epsilon_0^2 \cdot \sigma''(x_1))$ . Hence we can approximate the multiplicative neurons arbitrarily well by networks

of standard neurons with approximation function  $\sigma$ . The same argumentation as in Theorem 3.4 shows that the approximation can be done uniformly up to inputs of arbitrarily small probability. Since the multiplicative neuron with maximum complexity in Lemma 3.15 includes two multiplicative terms of order 2, two terms of order 3, and two terms of order 4, a feedforward network with at most two hidden layers and 49 neurons in the feedforward network can approximate the computation of a multiplicative neuron.  $\square$

### 3.3.5 Some implications

Note that the key point of the construction is given by Theorem 3.14 and Lemma 3.15. Theorem 3.14 defines a linear code which represents DBAGs up to a given height  $h$  over a finite alphabet  $\mathcal{L}$  uniquely and Lemma 3.15 shows that this code can be computed in an appropriate way with a CRecCC network. The other ingredients are steps (III) to (V) of the construction for RCC networks, which can be directly transferred to trees and DBAGs. Note that we can perform the above construction steps formally for DPAGs instead of DBAGs as well. The only problem is that the linear codes provided by  $rep_i$  might not encode all DPAGs uniquely. We have already found one example of DPAGs (Theorem 3.5) which are not uniquely encoded by any  $rep_i$ . If such situations are prevented, universal approximation for DPAGs can be guaranteed as well:

**Theorem 3.17** *Assume  $\mathcal{L} = \mathbb{R}^n$ . Assume  $P$  is some probability measure on DPAGs over  $\mathcal{L}$ . Assume  $f : \text{DPAG}_{\mathcal{L}}^2 \rightarrow \mathbb{R}$  is a Borel-measurable function. Assume  $\epsilon > 0$ ,  $\delta > 0$ . Assume  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  is a continuous and locally  $C^1$  squashing function. Assume the set  $\{\mathcal{D} \in \text{DPAG}_{\mathcal{L}}^2 \mid \exists \mathcal{D}' \in \text{DPAG}_{\mathcal{L}}^2 \quad f(\mathcal{D}) \neq f(\mathcal{D}'), \forall i \quad rep_i(\text{root}(\mathcal{D})) = rep_i(\text{root}(\mathcal{D}'))\}$  has probability at most  $\delta/2$ . Then there exists a CRecCC network with multiplicative neurons with activation functions  $\sigma$  and linear activation func-*

tion for the last neuron of the network which computes a function  $g$  such that

$$P(x \in \text{DPAG}_{\mathcal{L}}^2 \mid |f(x) - g(x)| > \delta) < \epsilon.$$

*Proof.* We can disregard the set of DPAGs which are not uniquely encoded by  $\text{rep}_i$  by assumption. For the remaining part, we can construct an approximative CRecCC network as in the previous case. Note, that encoding has only to be done for DPAGs of limited height with discretized labels, i.e. for a finite number of structures, such that a maximum number  $i$  which differentiates all these structures can be found. The remaining steps are analogous as for DBAGs whereby the encoding network computes  $\text{rep}_i$  for the above  $i$ .  $\square$

In practice, this allows us to approximate reasonable mappings on DPAGs as well:  $\text{rep}_i(\mathcal{D}) = \text{rep}_i(\mathcal{D}')$  for two DPAGs  $\mathcal{D} \neq \mathcal{D}'$  and all  $i$  require that for any vertex in the given two DPAGs and all paths following the connections towards children or parents, only vertices of the two DPAGs with the same label and the same number of children and parents are reached in  $\mathcal{D}$  and  $\mathcal{D}'$ . Hence this requires highly symmetric situations such that CRecCC networks can be expected to perform quite well in practical applications even for general DPAGs.

We would like to add a remark on the form of functions to be approximated. So far we have considered supersource transductions, i.e. functions which map a structure to a single value, i.e. the recursively computed output of the supersource of the structure. IO-isomorphic transductions refer to more general functions, which map a given input DPAG to a DPAG of the same structure where each vertex label of the input DPAG  $l(v)$  is substituted by a value  $l'(v)$ . Thereby, the value  $l'(v)$  of an output vertex might depend on the entire input DPAG. Thus IO-isomorphic transductions are functions  $f : \text{DPAG}_{\mathcal{L}}^2 \rightarrow \text{DPAG}_Y^2$  with label sets  $\mathcal{L}$  and  $Y$  such that  $f(x)$  has the same structure as  $x$  for every  $x \in \text{DPAG}_{\mathcal{L}}^2$ . As beforehand, we assume that  $\mathcal{L}$  and

$Y$  are subsets of a real-vector space. We call a function  $f$  measurable if for every fixed input structure of DPAGs and every fixed vertex in the structure the mapping, which maps the input DPAG to the output value of the vertex is Borel-measurable. The distance of two output DBAGs of the same structure is measured by taking the maximum distance of the labels of vertices at the same position. We can obviously obtain IO-isomorphic transductions from CRecCC networks by setting the output label of a vertex  $v$  of a given input structure to the output of the CRecCC network after processing the vertex  $v$ . The question now occurs whether all measurable IO-isomorphic transductions on DBAGs can be approximated by some CRecCC network.

To assess this question, we can use the same steps as beforehand, whereby the only difference occurs for step **(I)**: we have to compute an appropriate output after each presentation of a vertex, thus we have to find unique linear codes for each vertex in each possible DBAG. If this is possible, we can solve the problem to map each encoded vertex of a DBAG to the desired output as beforehand. Thus, steps **(II)** to **(V)** can be transferred from the previous case. Assume, a DBAG  $\mathcal{D}$  and a vertex  $v$  in  $\mathcal{D}$  is given. Consider the representation  $rep_h(v)$ . If the height of  $v$  is smaller than  $h$ , we can reconstruct the actual vertex  $v$  including its label  $l(v)$  and all linear path representations of the root to  $v$  from this term because of Lemma 3.12. In addition, we can mimic the construction of Lemma 3.13 and collect all direct or indirect successors of  $v$  together with the linear path representations of all paths from the root to these vertices. For predecessors of  $v$ , however, we have to add an argument: note that  $rep_h(v)$  has the form  $\tau_h(l(v), rep_h(ch_1(v)), rep_h(ch_2(v)), rep_{h-1}(pa_1(v)), rep_{h-1}(pa_2(v)))$ . Thus, if the DBAG has height at most  $h - 2$ , we can also reconstruct the two parents of  $v$  and all linear path representations from  $rep_{h-1}(pa_1(v))$  and  $rep_{h-1}(pa_2(v))$ . Iterating this argument yields the result, that the vertices and

all linear path representations of the DBAG can be reconstructed from  $rep_h(v)$  as in Lemma 3.13 if the DBAG has height at most  $h/2 - 1$ . Note that for each reconstruction step the index  $i$  of  $rep_i(v)$  is decreased by one, such that we have to start with twice the height. Stated in other words,  $rep_{2h+1}(v)$  is a unique linear representation of  $v$  with respect to the vertex  $v$  and the entire DBAG if the height of the DBAG is at most  $h$ . As a consequence we can conclude that CRecCC networks are also universal approximators for IO-isomorphic transduction on DBAGs.

## 4 Discussion

We have considered various cascade correlation models enhanced by recurrent or recursive connections for structured input data. These models have been proposed in the literature based on recurrent or recursive neural networks in analogy to simple cascade correlation. Recurrent and recursive networks constitute well established machine learning tools to deal with sequences or tree structures as input, respectively. They have successfully been applied for various type of classification and regression problems on structured data ranging from picture processing up to chemistry (Frasconi, Gori, and Sperduti, 1998). However, it is well known that training recurrent networks faces severe problems (Bengio, Simard, and Frasconi, 1994) and the generalization ability might be considerably worse compared to standard feedforward networks (Hammer, 2001). Moreover, classification tasks for sequences and tree structures are likely more difficult than standard learning problems for feedforward networks due to the increased complexity of data. Hence constructive learning algorithms which divide the problem into several subtasks and automatically find an appropriate (preferably small) architecture are particularly useful within structured domains. Cascade correlation for structures has shown superior

performance for several problems in chemistry (Bianucci et al., 2000) and seems particularly appropriate due to very low training effort (the weights of only one neuron are trained at a time) and an automatically growing structure.

Unlike the modification of standard feedforward networks to feedforward cascade correlation networks, the specific training scheme of CC restricts the possible dynamics of recurrent cascade architectures to only partially recurrent systems. As a consequence, the class of recurrent cascade architectures constitutes a proper subclass of the class of recurrent network architectures. This might restrict the capability of these type of networks compared to the general model, as demonstrated e.g. in (Giles et al., 1995). Hence the question was whether the restriction of the recurrence in cascade models poses severe restrictions on the applicability of the models because possibly only very simple functions can be represented with these restricted architectures. We have shown in this article that in terms of the universal approximation capability of these models, no restriction can be observed: CC-architectures can approximate every measurable function with sequences or tree structures as input, respectively, arbitrarily well up to a set of arbitrary small probability in spite of their restricted recurrence. So if fundamental differences concerning the capacity of these systems in comparison to their fully recurrent counterparts exist (as demonstrated in (Giles et al., 1995)) then these fundamental differences will not manifest for any given finite set of training data or for any given finite time horizon or height of the trees, respectively, provided enough neurons are available.

The restriction of recurrence in cascade models offers a very natural possibility to enlarge the dynamics: not only information provided by successors of vertices of a graph or tree structure can be taken into account, but also contextual information provided by the predecessors of the vertices. These contextual recursive models have been proposed in (Micheli, Sona, and Sperduti, 2003b; Micheli, Sona, and

Sperduti, 2002). It has been demonstrated that this additional information is actually used by the contextual models such that tasks which involve global information of the structure can be solved more easily with these models. Note that also for standard recurrent and recursive networks several models which integrate global information for a restricted form of structured data such as sequences or grids have been proposed in the literature and successfully been applied e.g. in bioinformatics where often spatial data, instead of temporal data, has to be dealt with (Baldi et al., 1999; Micheli, Sona, and Sperduti, 2000; Pollastri et al., 2002; Wakuya and Zurada, 2001). However, unlike contextual cascade models, structure processing is restricted to special subclasses and integration of contextual information is here done on top of the basic recursive networks. This is necessary because of full recurrence of the networks. Context integration in form of structural induction would yield a cyclic, ill-defined dynamics.

The restriction of the recurrence in cascade models allows to integrate the contextual information directly into the model in a very elegant and efficient way. As already demonstrated in (Micheli, Sona, and Sperduti, 2003b), this moreover enlarges the class of structures the models can deal with from tree structures to acyclic rooted positional graphs with limited fan-in and fan-out. We went a step further in this article by showing that contextual cascade models constitute in fact universal approximators for the class of functions defined on acyclic directed graphs as input. More precisely, we have identified a subclass of DPAGs which we called DBAGs where enumeration of children and parents is compatible and we have shown the universal approximation capability for this subclass. Moreover, we have shown that any DPAG can be transformed into a DBAG in a natural way via possible expansion by empty children and parents and reenumeration of children and parents. Hence acyclic graph structures where the positioning is not relevant can easily be dealt with

in this way. In addition, the approximation capability is extended to IO-isomorphic transductions, i.e. CRecCC networks constitute one of the first models for which a proof exists that they can also approximate functions with structured outputs. Thus, the universal approximation capability of cascade architectures with restricted recurrence and context integration is enlarged to general data structures such as rooted directed acyclic graphs with limited fan-in and fan-out.

## References

- Alquezari, R. and Sanfeliu, A. (1995). An algebraic framework to represent finite state machines in single-layer recurrent neural networks. *Neural Computation*, 7(5):931-949.
- Baldi, P., Brunak, S., Frasconi, P., Pollastri, G., and Soda, G. (1999). Exploiting the past and future in protein secondary structure prediction. *Bioinformatics*, 15(11):937-946.
- Bengio Y. and Frasconi, P. (1996). Input/output HMMs for sequence processing. *IEEE Transactions on Neural Networks*, 7(5):1231-1249.
- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157-166.
- Bianucci, A.M., Micheli, A., Sperduti, A., and Starita, A. (2000). Application of cascade correlation networks for structures to chemistry. *Journal of Applied Intelligence*, 12:117-146.
- Diligenti, M., Frasconi, P., and Gori, M. (2003). Hidden tree Markov models for document image classification. *IEEE Transactions on Pattern Analysis*

*and Machine Intelligence*, 25(4):519-523.

Fahlmann, S.E. and Lebiere, C. (1990). The cascade-correlation learning architecture. In: D.S.Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann, pp. 524-532.

Fahlmann, S.E. (1991). The recurrent cascade-correlation architecture. In: R.P.Lippmann, J.E.Moody, and D.S.Touretzky (eds.), *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann, pp. 190-196.

Frasconi, P. (2002). Comparing convolution kernels and RNNs on a wide-coverage computational analysis of natural language. *NIPS 2002*, Workshop on unreal data.

Frasconi, P. and Gori, M. (1996). Computational capabilities of local-feedback recurrent networks acting as finite-state machines. *IEEE Transactions on Neural Networks*, 7(6):1521-1524.

Frasconi, P., Gori, M., and Sperduti, A. (1998). A general framework of adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5): 768-786.

Forcada M.L. and Carrasco, R.C. (1995). Learning the initial state of a second order recurrent neural network during regular language inference. *Neural Computation*, 7(5):923-949.

Funahashi K. and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801-806.

Giles, C.L., Chen, D., Sun, G.Z., Chen, H.H., Lee, Y.C., and Goudreau, M.W. (1995). Constructive learning of recurrent neural networks: limitations of

- recurrent cascade correlation and a simple solution. *IEEE Transactions on Neural Networks*, 6(4):829-836.
- Goller, C. (1997). *A connectionist approach for learning search control heuristics for automated deduction systems*. PhD thesis, Technische Universität München.
- Goller, C. and Kuchler, A. (1996). Learning task-dependent distributed structure-representations by backpropagation through structure. In: *IEEE International Conference on Neural Networks*, pp. 347-352.
- Hammer, B. (2001). Generalization ability of folding networks. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):196–206.
- Hammer, B. (2000). *Learning with recurrent neural networks*. Springer Lecture Notes in Control and Information Sciences 254, Springer.
- Hammer, B. and Steil, J.J. (2002). Perspectives on learning with recurrent neural networks. In M. Verleysen, editor, *ESANN'02*, pages 357-368. D-side publications.
- Haussler, D. (1999). *Convolution kernels on discrete structure*. Technical report, UC Santa Cruz.
- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359-366.
- Jaakkola, T., Diekhans, M., and Haussler, D. (2000). A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7:95–114.
- Kremer S. (2001). Spatio-temporal connectionist networks: A taxonomy and review. *Neural Computation*, 13(2):249-306.

- Leslie, C., Eskin, E., and Noble, W. (2002). The spectrum kernel: A string kernel for SVM protein classification. *Proc. Pacific Symposium on Bio-computing*, pages 564-575.
- Lodhi, H., Shawe-Taylor, J., Cristianini, N., and Watkins, C.J.C.H. (2000). Text classification using string kernels. *NIPS*, pages 563-569.
- Mauro, C. de, Diligenti, M., Gori, M., and Maggini, M. (2003). Similarity learning for graph-based image representations. *Pattern Recognition Letters* 24(8):1115-1122.
- Micheli, A. Recursive Processing of Structured Domains in Machine Learning. PhD thesis, Technical Report TD-13/03, Department of Computer Science, University of Pisa.
- Micheli, A., Sona, D., and Sperduti, A. (2003a). Formal definition of context in contextual recursive cascade correlation networks. In E.Alpaydin, E.Oja, and L.Xu (eds.), *Proceedings of ICANN/ICONIP 2003*, LNCS 2714, pages 173-130.
- Micheli, A., Sona, D., and Sperduti, A. (2003b). Contextual processing of structured data by recursive cascade correlation. To appear in *IEEE Transactions on Neural Networks*.
- Micheli, A., Sona, D., and Sperduti, A. (2002). Recursive cascade correlation for contextual processing of structured data, In: *Proceedings of the International Joint Conference on Neural Networks 2002*, 1, pp.268-273.
- Micheli, A., Sona, D., and Sperduti, A. (2000). Bi-causal recurrent cascade correlation. In: *Proceedings of the International Joint Conference on Neural Networks*, vol.3, pp.3-8.
- Pollastri, G., Baldi, P., Vullo, A., and Frasconi, P. (2002). Prediction of protein

- topologies using GIOHMMs and GRNNs. In: *Advances of Neural Information Processing Systems 2002*.
- Scarselli, F. and Tsoi, A.C. (1998). Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, 11:15–37.
- Sontag, E.D. (1992). Feedforward nets for interpolation and classification. *Journal of Computer and System Sciences*, 45:20-48.
- Sperduti, A. (1997). On the computational power of neural networks for structures. *Neural Networks*, 10(3):395-400.
- Sperduti, A., Majidi, D., and Starita, A. (1996). Extended cascade-correlation for syntactic and structural pattern recognition. In: P.Perner, P.Wand, A.Rosenfeld (eds.), *Advances in Structured and Syntactical Pattern Recognition*, Springer, pp.90-99.
- Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3): 714-735.
- Sturt, P., Costa, F., Lombardo, V., and Frasconi, P. (2003). Learning first-pass structural attachment preferences with dynamic grammars and recursive neural networks. *Cognition*, 88(2):133-169.
- Sun, R. (2001). Introduction to sequence learning. R. Sun, C.L. Giles (eds.), *Sequence Learning: Paradigms, Algorithms, and Applications*, pp. 1-10, Springer.
- Vullo, A. and Frasconi, P. (2003). A recursive connectionist approach for predicting disulfide connectivity in proteins. *Proceedings of the Eighteenth*

*Annual ACM Symposium on Applied Computing (SAC 2003)*, Melbourne, FL.

Watkins, C. (1999). *Dynamic alignment kernels*. Technical report, UL Royal Holloway.

Wakuya, H. and Zurada, J. (2001). Bi-directional computing architectures for time series prediction. *Neural Networks*, 14:1307-1321.

## Appendix

### Proof of Theorem 3.5

Assume a stationary CRecCC network is given. Consider the two graphs depicted in Fig. 3. We refer to the vertices by the numbers as indicated in the figure. Thereby, enumeration of children and parents is done from left to right as they appear in the figure. I.e. the upper left box of a vertex refers to parent number one, the upper right box to parent number two, the lower left box indicates child number one, the lower right box refers to child number two. The labels of all vertices are identical, say  $l$ . Note that the structures are highly symmetric: each pair of vertices  $i$  and  $i'$  has the same number of children and parents and the same label. Moreover,  $i$  and  $i'$  are pointing to vertices which have the same structure in the left and the right DPAG. We would like to point out one aspect of these structures: consider the edges  $(0, 4)$  and  $(0', 4')$ .  $4$  and  $4'$  constitute the respective *second* child of the root, whereas  $0$  and  $0'$  constitute the *first* parent of  $4$  and  $4'$ , respectively. For all other edges, the positioning of the respective parent coincides with the positioning of the respective child.

As a first step, we show by induction over  $i$  that the following equalities hold

for the hidden neurons' activation:

$$\begin{aligned}x_i(1) &= x_i(4), x_i(2) = x_i(5), x_i(3) = x_i(6), \\x_i(1') &= x_i(4'), x_i(2') = x_i(5'), x_i(3') = x_i(6')\end{aligned}$$

This indicates that a large symmetry can already be found *within* each structure.

$i = 1$ :

$$\begin{aligned}x_1(3) &= \tau_1(l, 0_c^1, 0_c^1) = x_1(6), \\x_1(2) &= \tau_1(l, x_1(3), 0_c^1) = \tau_1(l, x_1(6), 0_c^1) = x_1(5), \\x_1(1) &= \tau_1(l, x_1(2), x_1(3)) = \tau_1(l, x_1(5), x_1(6)) = x_1(4)\end{aligned}$$

and

$$\begin{aligned}x_1(3') &= \tau_1(l, 0_c^1, 0_c^1) = x_1(6'), \\x_1(2') &= \tau_1(l, x_1(3'), 0_c^1) = \tau_1(l, x_1(6'), 0_c^1) = x_1(5'), \\x_1(1') &= \tau_1(l, x_1(2'), x_1(6')) = \tau_1(l, x_1(5'), x_1(3')) = x_1(4')\end{aligned}$$

Induction step for  $i$ :

$$\begin{aligned}x_i(3) &= \tau_i(l, x_1(3), \dots, x_{i-1}(3), 0_c^1, \dots, 0_c^i, 0_c^1, \dots, 0_c^i, x_1(2), \dots, x_{i-1}(2), x_1(1), \dots, x_{i-1}(1)) \\&= \tau_i(l, x_1(6), \dots, x_{i-1}(6), 0_c^1, \dots, 0_c^i, 0_c^1, \dots, 0_c^i, x_1(5), \dots, x_{i-1}(5), x_1(4), \dots, x_{i-1}(4)) \\&= x_i(6)\end{aligned}$$

$$\begin{aligned}x_i(2) &= \tau_i(l, x_1(2), \dots, x_{i-1}(2), x_1(3), \dots, x_i(3), 0_c^1, \dots, 0_c^i, x_1(1), \dots, x_{i-1}(1), 0_p^1, \dots, 0_p^{i-1}) \\&= \tau_i(l, x_1(5), \dots, x_{i-1}(5), x_1(6), \dots, x_i(6), 0_c^1, \dots, 0_c^i, x_1(4), \dots, x_{i-1}(4), 0_p^1, \dots, 0_p^{i-1}) \\&= x_i(5)\end{aligned}$$

$$\begin{aligned}x_i(1) &= \tau_i(l, x_1(1), \dots, x_{i-1}(1), x_1(2), \dots, x_i(2), x_1(3), \dots, x_i(3), x_1(0), \dots, x_{i-1}(0), 0_p^1, \dots, 0_p^{i-1}) \\&= \tau_i(l, x_1(4), \dots, x_{i-1}(4), x_1(5), \dots, x_i(5), x_1(6), \dots, x_i(6), x_1(0), \dots, x_{i-1}(0), 0_p^1, \dots, 0_p^{i-1}) \\&= x_i(4)\end{aligned}$$

and

$$\begin{aligned}
& x_i(3') \\
&= \tau_i(l, x_1(3'), \dots, x_{i-1}(3'), 0_c^1, \dots, 0_c^i, 0_c^1, \dots, 0_c^i, x_1(2'), \dots, x_{i-1}(2'), x_1(4'), \dots, x_{i-1}(4')) \\
&= \tau_i(l, x_1(6'), \dots, x_{i-1}(6'), 0_c^1, \dots, 0_c^i, 0_c^1, \dots, 0_c^i, x_1(5'), \dots, x_{i-1}(5'), x_1(1'), \dots, x_{i-1}(1')) \\
&= x_i(6')
\end{aligned}$$

$$\begin{aligned}
& x_i(2') \\
&= \tau_i(l, x_1(2'), \dots, x_{i-1}(2'), x_1(3'), \dots, x_i(3'), 0_c^1, \dots, 0_c^i, x_1(1'), \dots, x_{i-1}(1'), 0_p^1, \dots, 0_p^{i-1}) \\
&= \tau_i(l, x_1(5'), \dots, x_{i-1}(5'), x_1(6'), \dots, x_i(6'), 0_c^1, \dots, 0_c^i, x_1(4'), \dots, x_{i-1}(4'), 0_p^1, \dots, 0_p^{i-1}) \\
&= x_i(5')
\end{aligned}$$

$$\begin{aligned}
& x_i(1') \\
&= \tau_i(l, x_1(1'), \dots, x_{i-1}(1'), x_1(2'), \dots, x_i(2'), x_1(6'), \dots, x_i(6'), x_1(0'), \dots, x_{i-1}(0'), 0_p^1, \dots, 0_p^{i-1}) \\
&= \tau_i(l, x_1(4'), \dots, x_{i-1}(4'), x_1(5'), \dots, x_i(5'), x_1(3'), \dots, x_i(3'), x_1(0'), \dots, x_{i-1}(0'), 0_p^1, \dots, 0_p^{i-1}) \\
&= x_i(4')
\end{aligned}$$

Next, we show by induction over  $i$  that  $x_i(n) = x_i(n')$  holds for all  $n \in \{0, \dots, 6\}$ .

In particular, any CRecCC network yields the same output for both graphs independently of the precise form of the network functions  $\tau_i$ .

$i = 1$ :

$$\begin{aligned}
x_1(3) &= x_1(6) = \tau_1(l, 0_c^1, 0_c^1) = x_1(3') = x_1(6') \\
x_1(2) &= x_1(5) = \tau_1(l, x_1(3), 0_c^1) = \tau_1(l, x_1(3'), 0_c^1) = x_1(2') = x_1(5') \\
x_1(1) &= x_1(4) = \tau_1(l, x_1(2), x_1(3)) = \tau_1(l, x_1(2'), x_1(6')) = x_1(1') = x_1(4') \\
x_1(0) &= \tau_1(l, x_1(1), x_1(4)) = \tau_1(l, x_1(1'), x_1(4')) = x_1(0')
\end{aligned}$$

Induction step for  $i$ :

$$\begin{aligned}
& x_i(3) \\
&= \tau_i(l, x_1(3), \dots, x_{i-1}(3), 0_c^1, \dots, 0_c^i, 0_c^1, \dots, 0_c^i, x_1(2), \dots, x_{i-1}(2), x_1(1), \dots, x_{i-1}(1)) \\
&= \tau_i(l, x_1(3), \dots, x_{i-1}(3), 0_c^1, \dots, 0_c^i, 0_c^1, \dots, 0_c^i, x_1(2), \dots, x_{i-1}(2), x_1(4), \dots, x_{i-1}(4)) \\
&= \tau_i(l, x_1(3'), \dots, x_{i-1}(3'), 0_c^1, \dots, 0_c^i, 0_c^1, \dots, 0_c^i, x_1(2'), \dots, x_{i-1}(2'), x_1(4'), \dots, x_{i-1}(4')) \\
&= x_i(3') = x_i(6') = x_i(6)
\end{aligned}$$

$$\begin{aligned}
& x_i(2) \\
&= \tau_i(l, x_1(2), \dots, x_{i-1}(2), x_1(3), \dots, x_i(3), 0_c^1, \dots, 0_c^i, x_1(1), \dots, x_{i-1}(1), 0_p^1, \dots, 0_p^{i-1}) \\
&= \tau_i(l, x_1(2'), \dots, x_{i-1}(2'), x_1(3'), \dots, x_i(3'), 0_c^1, \dots, 0_c^i, x_1(1'), \dots, x_{i-1}(1'), 0_p^1, \dots, 0_p^{i-1}) \\
&= x_i(2') = x_i(5') = x_i(5)
\end{aligned}$$

$$\begin{aligned}
& x_i(1) \\
&= \tau_i(l, x_1(1), \dots, x_{i-1}(1), x_1(2), \dots, x_i(2), x_1(3), \dots, x_i(3), x_1(0), \dots, x_{i-1}(0), 0_p^1, \dots, 0_p^{i-1}) \\
&= \tau_i(l, x_1(1), \dots, x_{i-1}(1), x_1(2), \dots, x_i(2), x_1(6), \dots, x_i(6), x_1(0), \dots, x_{i-1}(0), 0_p^1, \dots, 0_p^{i-1}) \\
&= \tau_i(l, x_1(1'), \dots, x_{i-1}(1'), x_1(2'), \dots, x_i(2'), x_1(6'), \dots, x_i(6'), x_1(0'), \dots, x_{i-1}(0'), 0_p^1, \dots, 0_p^{i-1}) \\
&= x_i(1') = x_i(4') = x_i(4)
\end{aligned}$$

$$\begin{aligned}
& x_i(0) \\
&= \tau_i(l, x_1(0), \dots, x_{i-1}(0), x_1(1), \dots, x_i(1), x_1(4), \dots, x_i(4), 0_p^1, \dots, 0_p^{i-1}, 0_p^1, \dots, 0_p^{i-1}) \\
&= \tau_i(l, x_1(0'), \dots, x_{i-1}(0'), x_1(1'), \dots, x_i(1'), x_1(4'), \dots, x_i(4'), 0_p^1, \dots, 0_p^{i-1}, 0_p^1, \dots, 0_p^{i-1}) \\
&= x_i(0')
\end{aligned}$$

Hence the structures cannot be differentiated by any CRecCC network regardless of the chosen specific weights of the network and form of the neurons.  $\square$

### Proof of Theorem 3.7

Assume a DPAG  $\mathcal{D}$  is given. If  $\mathcal{D}$  is empty, the result is obvious. Assume  $\mathcal{D}$  is not empty. The first construction steps can be done for any limited fan-in and fan-out at

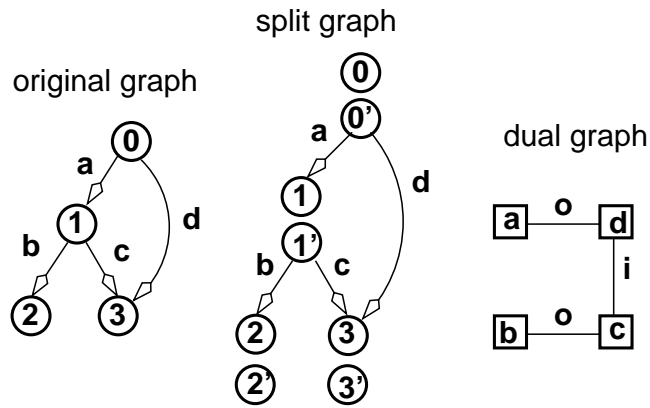


Figure 4: Example of the transformation of a graph (left) for which a compatible positioning has to be found to the graph where each vertex is split into a parent vertex denoted by  $0, \dots, 3$  and a child vertex denoted by  $0', \dots, 3'$  (middle). Thereby, connections to empty children and connections from empty parents are dropped. If the connections are considered as undirected connections, the dual graph can be constructed (right). Now an enumeration of the vertices in the dual graph which corresponds to a compatible positioning of children and parents in the original DPAG can be constructed.

most  $k$ , thus we describe the steps for general  $k$ .

Given  $\mathcal{D}$ , consider the following related (split) graph  $\mathcal{D}'$  where the role of parents and children is explicitly assigned to the vertices: vertices in  $\mathcal{D}'$  are  $\{u_p | \exists(u, v) \in \text{edge}(\mathcal{D})\}$  and  $\{v_c | \exists(u, v) \in \text{edge}(\mathcal{D})\}$ , edges in  $\mathcal{D}'$  are  $\{(u_p, v_c) | \exists(u, v) \in \text{edge}(\mathcal{D})\}$ . See Fig. 4 (middle) for an example. This graph can be further transformed to its dual graph  $\mathcal{D}'^\vee$  with labeled edges with labels in  $\{i, o\}$  which we obtain as follows: the dual graph introduces a vertex  $v_e$  for each edge  $e$  in  $\mathcal{D}'$  and an edge  $(v_e, v_f)$  if a vertex in  $\mathcal{D}'$  exists such that  $e$  and  $f$  are both connected to this vertex. I.e.  $e = (u_p, v_c)$  and  $f = (u'_p, v_c)$  (the edges are connected via a child) or  $e = (u_p, v_c)$  and  $f = (u_p, v'_c)$  (the edges are connected via a parent) for some vertices  $u_p, v_c, u'_p, v'_c$  in  $\mathcal{D}'$ . An edge  $(v_e, v_f)$  in  $\mathcal{D}'^\vee$  is labeled with  $i$  if  $e$  and  $f$  are connected via a child in  $\mathcal{D}'$ , i.e. they are both ingoing edges in  $\mathcal{D}'$ . An edge  $(v_e, v_f)$  in  $\mathcal{D}'^\vee$  is labeled with  $o$  if  $e$  and  $f$  are connected via a parent in  $\mathcal{D}'$ , i.e. they are both outgoing edges in  $\mathcal{D}'$ . See Fig. 4(right) for an example.

Now, we can relate positionings of  $\mathcal{D}$  to positionings of  $\mathcal{D}'$  and positionings of  $\mathcal{D}'$  to enumerations of  $\mathcal{D}'^\vee$  to prove the theorem. First consider  $\mathcal{D}$  and  $\mathcal{D}'$ .  $\mathcal{D}'$  has fan-in and fan-out  $k$  like  $\mathcal{D}$ . Since the positioning of the children of a fixed vertex can be done independently from the positioning of the parents of this vertex, the following holds: there is a one-to-one connection between compatible positionings of  $(P'_v, C'_v)$  of the vertices  $v$  in  $\mathcal{D}'$  and  $(P_v, C_v)$  of the vertices in  $\mathcal{D}$ . Thereby,  $\mathcal{D}'$  is no longer a rooted DPAG, however, the notion of a compatible positioning in the above form can also be applied for unrooted DPAGs. The one-to-one connection is given by the identities  $P'_{v_c}(u_p) = P_v(u)$  and  $C'_{u_p}(v_c) = C_u(v)$ . This holds because positioning of the (empty) children of  $v_c$  and of the (empty) parents of  $u_p$  can be done arbitrarily.

Note that any compatible positioning of  $\mathcal{D}'$  corresponds to the assignment of

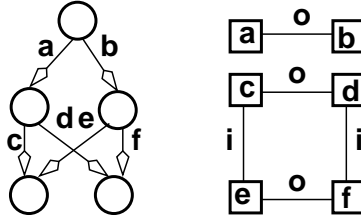


Figure 5: Example of the structure of the dual graph (right) for a given DPAG (left): it decomposes into linear subgraphs and ring structures with an even number of vertices.

a number in  $\{1, \dots, k\}$  to each edge in the graph such that edges connected to the same vertex have different numbers: an edge  $(u_p, v_c)$  is assigned the number  $P'_{v_c}(u_p) = C'_{u_p}(v_c)$ ; these values coincide for a compatible positioning. Since the numbers come from a positioning, edges connected to the same vertex have different numbers. Conversely, any assignment of numbers  $\{1, \dots, k\}$  to the edges of  $\mathcal{D}'$  such that edges connected to the same vertex have different numbers can be transformed to a compatible positioning. If a value  $i$  is assigned to edge  $(u_p, v_c)$  we set  $P'_{v_c}(u_p) = C'_{u_p}(v_c) = i$ . Since edges connected to the same vertex have different numbers, this defines a valid positioning.

Now consider the dual graph  $\mathcal{D}'^V$ . Note the following: Assigning numbers to the edges in  $\mathcal{D}'$  such that edges in  $\mathcal{D}'$  connected to the same vertex have different numbers means to assign numbers to the vertices in  $\mathcal{D}'^V$  such that vertices in  $\mathcal{D}'^V$  which are connected in  $\mathcal{D}'^V$  have different numbers. We have already argued that such an assignment for  $\mathcal{D}'$  leads to a compatible positioning of the graph  $\mathcal{D}$ .

We now use the restriction on  $k$ . Since  $\mathcal{D}'$  has limited fan-in and fan-out  $k$  equal to two, each vertex in  $\mathcal{D}'^V$  has at most one connection labeled with  $i$  and one connection labeled with  $o$ . Thus each vertex in  $\mathcal{D}'^V$  has at most two edges. Hence  $\mathcal{D}'^V$  decomposes into connected components which have a very simple structure: the

components constitute either a linear graph, or they have a ring structure (see Fig. 5). Thereby, each ring has necessarily an even number of vertices because each vertex is connected by one edge labeled with  $i$  and one edge labeled with  $o$ . The assignment of values in  $\{1, 2\}$  can be done independently for each connected component of  $\mathcal{D}'$ . For a linear structure, we can simply assign consecutively alternating values to the vertices. For a ring, we can do the same starting at some arbitrary position because the number of vertices in a ring is even. Thus, an appropriate assignment of numbers to the vertices of  $\mathcal{D}'$  such that a compatible positioning of  $\mathcal{D}$  arises is always possible.  $\square$

### **Proof of Theorem 3.8**

As beforehand, we can construct the graph  $\mathcal{D}'$  from  $\mathcal{D}$  with vertices  $\{u_p \mid \exists(u, v) \in \text{edge}(\mathcal{D})\}$  and  $\{v_c \mid \exists(u, v) \in \text{edge}(\mathcal{D})\}$  and edges in  $\mathcal{D}'$  are  $\{(u_p, v_c) \mid \exists(u, v) \in \text{edge}(\mathcal{D})\}$ , and its dual graph  $\mathcal{D}''$ . Each vertex in  $\mathcal{D}''$  has at most  $2(k - 1)$  connections, since each edge in  $\mathcal{D}'$  is connected to a child with at most  $k - 1$  further ingoing edges in  $\mathcal{D}'$  and a parent with at most  $k - 1$  further outgoing edges in  $\mathcal{D}'$ . Hence we can obviously assign numbers  $\{1, \dots, 2k - 1\}$  to each vertex in  $\mathcal{D}''$  such that no two vertices which are connected in  $\mathcal{D}''$  have the same numbers assigned; even if all  $2(k - 1)$  neighbors of a vertex are already enumerated, there is still one number available. These numbers correspond to a compatible positioning of children and parents in the original DPAG  $\mathcal{D}$  with positions in  $\{1, \dots, 2k - 1\}$ . Hence filling the not yet assigned positions for children and parents with empty children and parents, we obtain a DPAG from  $\mathcal{D}$  with compatible positioning and fan-in and fan-out  $2k - 1$ , thereby expanding with empty children and parents.  $\square$