

# Recurrent networks for structured data – a unifying approach and its properties

Barbara Hammer

*Universität Osnabrück, Department of Mathematics/Computer Science, D-49069  
Osnabrück, Germany, e-mail: hammer@informatik.uni-osnabrueck.de*

---

## Abstract

We consider recurrent neural networks which deal with symbolic formulas, terms, or, generally speaking, tree-structured data. Approaches like the recursive autoassociative memory, discrete-time recurrent networks, folding networks, tensor construction, holographic reduced representations, and recursive reduced descriptions fall into this category. They share the basic dynamics of how structured data are processed: the approaches recursively encode symbolic data into a connectionistic representation or decode symbolic data from a connectionistic representation by means of a simple neural function. In this paper, we give an overview of the ability of neural networks with these dynamics to encode and decode tree-structured symbolic data. The correlated tasks, approximating and learning mappings where the input domain or the output domain may consist of structured symbolic data, are examined as well.

---

**Keywords:** Hybrid systems, recurrent networks, recursive autoassociative memory, holographic reduced representation, approximation ability, learnability

## 1 Introduction

An ubiquitous characteristic of real-life data is compositionality and structure: written English language is composed of 26 characters and a few additional symbols which are combined to words and sentences with a specific structure; visual scenes decompose into single figures which are grouped together, each figure characterized by a certain shape, texture, color, edges, *etc.*; computer programs decompose into functions, procedures, objects, methods, *etc.* In all cases, essential information lies in the basic primitives as well as their respective role with regard to the whole structure. The human brain, being a highly efficient biological neural network, obviously processes structured data easily – we can understand and produce speech; we can recognize visual scenes. Moreover, since synthetic data produced

by humans is often structured – programs, web sites, mathematical formulas, for example – it is likely that the human brain indeed uses the principles of compositionality and structure for efficient processing.

In contrast, standard artificial neural networks as used in practice are mostly restricted to very simple data: the inputs or outputs, respectively, usually consist of simple real vectors of a specified finite and fixed dimension. That means, standard artificial neural networks commonly process and produce only flat and unstructured data. Since artificial neural networks constitute a universal and efficient learning mechanism with successful applications in various areas of application, they could probably add a valuable tool for areas where structured or symbolic data are involved. For this purpose, a generalization such that structured data can be processed automatically with neural tools is essential. Moreover, artificial neural networks have been designed such that they mimic important aspects of biological neural networks. Hence it is very interesting to understand in which way artificial neural networks can process and evolve symbolic data so that eventually more insight into biological networks performing these tasks can be gained.

Hence lots of effort has been done in order to extend connectionistic systems with symbolic aspects. Of course, there exist various possibilities: on the one hand, symbolic aspects allow a further insight into the way in which neural networks process data. They connect networks to the way in which human experts would formalize their behavior; rule extraction or rule insertion mechanisms enable us to deal with prior expert knowledge in connectionistic systems or to translate the connectionistic behavior to a behavior understandable by experts, respectively [7,46,47]. Furthermore, subsymbolic processing on its own is faced to severe limitations for modeling complex behavior; commonly the success of standard neural networks depends crucially on the way of how data are presented in practical applications. Raw data are to be preprocessed and appropriate feature extraction mechanisms are to be added for example in image classification, time series prediction, or language processing [4,24,28]. Hence standard connectionistic applications usually contain some prior knowledge or preprocessing in order to be successful. For short: symbolic methods and the integration of structured data enable the understanding of network behavior and they may speedup and boost neural network training considerably.

On the other hand, connectionistic systems provide a very simple, efficient, and problem-independent way of learning an unknown behavior from a finite set of examples and hence they could add a valuable learning tool to symbolic domains. Concerning the standard scenario with real vectors, theoretical results ensure that neural networks can represent (almost) any unknown behavior with an adequate neural architecture; they are universal approximators as shown in [19] as an example. Furthermore, the unknown behavior can be learned from a finite set of examples. The necessary size of the training set depends on the network architecture and the required accuracy; the theoretical background is provided by so-called statisti-

cal learning theory [1,2,49]. Assumed connectionistic methods and the respective theoretical guarantees could be established for symbolic domains, too, they could speedup symbolic processing of data and they could model behavior for which a precise logical description is hard to obtain. Moreover, the structure which is provided by the symbolic data may allow to interpret the behavior of the network. In addition, data may propose an adequate representation of the knowledge learned by the network [29,45]. As a consequence, neural methods could add an efficient and automatic learning tool to classical symbolic methods.

In the following, we consider neural networks with structured input or output domain, *i.e.* neural networks that learn mappings on symbolic data in order to speedup and accompany symbolic data processing. Problems occur in this context since standard data for neural networks are real vectors of a fixed and finite dimension. We are interested in structured data such as terms and symbolic formulas of a priori unlimited length. Unlike fixed dimensional vectors, the size of these complex data and their informational content are not limited a priori. Moreover, not only the data's primitives like the variables and operators in a logical formula carry important information but their respective relation is important, too. The standard way of further processing with neural methods is as follows: a finite set of real-valued features is extracted which constitutes an adequate representation of symbolic objects in a finite dimensional vector space. Of course, this approach requires problem-dependent knowledge and it is limited in principle because finite dimensional vectors carry only an a priori limited amount of information. An alternative possibility is as follows: the network structure could be changed so that symbolic data can be processed directly. Then the a priori unlimited size of structured data is mapped to a priori unlimited processing time of the networks with an appropriate dynamics. We will focus on this latter possibility.

Naturally, various different dynamics have been proposed for this purpose. Complex neural systems like LISA, SHRUTI, or INFERNET allow complex analogical reasoning and symbol processing [20,37,42]. They are based on a specific network topology as well as appropriate dynamical behavior such as synchronous oscillation of the neurons in order to encode complex structures. Alternatively, one can use the fine temporal structure for adequate representation of symbols as in networks of spiking neurons [25] or complex dynamical behavior for data processing in recurrent networks like the competitive layer model [50]. All these approaches constitute particular solutions to parts of the general so-called binding problem, *i.e.* the problem of how complex structured data are represented in biological neural networks such that rapid and faithful processing is possible [27,33].

However, we will not consider this interesting problem and solutions in general but focus on one particular possibility to process structured data with neural networks: symbolic data can be incorporated into connectionistic systems through recurrent dynamics. Since classical symbolic data like terms and logical formulas can be represented by a tree structure in a natural way, a canonic method is induced by

the standard recursive definition of trees. Recursive connections are added according to the recursive structure of trees. These connections enable us to deal with a priori unlimited complexity, *i.e.* unlimited height of the trees. The dynamical behavior of the network directly mirrors the dynamical structure of the data. The whole networks commonly decompose into three parts: a recursive neural encoding part where structured data are encoded into a connectionistic representation, a standard feedforward network used for processing, and a recursive neural decoding part where structured data are decoded from connectionistic representations.

This approach has several advantages: several concrete implementations of this idea and concrete learning algorithms have been proposed in the literature. In particular, successful real-life applications exist. Various realizations are for example the recursive autoassociative memory (RAAM) [32] and labeled RAAM (LRAAM) [44], holographic reduced representations (HRR) [31], and recurrent and folding networks [12]. The approaches differ in the method in which they are trained and in their areas of application. The basic recurrent dynamics are the same for all approaches: the possibility to deal with symbolic data relies on some either fixed or trainable recursive encoding and decoding of data. Some of the above mentioned methods show a surprisingly good performance in different areas of application such as time series prediction, automated theorem proving, picture processing, drug design, *etc.* [13,14,34,35]. They are often used for tasks which cannot be solved with standard feedforward networks in a natural way due to the complex data structures.

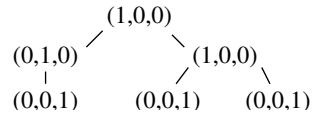
A second advantage consists in the fact that the approach is very natural and constitutes a generalization of well established recurrent networks unlike other specific solutions to the binding problem. Hence lots of standard methods and arguments from feedforward and recurrent networks can be transferred to this new scenario. In particular, it is possible to establish a general theory which will be the main topic of this article. Two key properties of feedforward networks are their universal approximation ability and their information theoretical learnability [1,19,49]. The same properties can be established for recurrent networks processing symbolic data. That means, the proposed mechanisms fulfill some kind of universal approximation ability and, furthermore, they yield valid generalization for an appropriate training set size. We will present an overview about the main results on these two topics in this article. Altogether, the results establish these type recurrent networks as a valuable and well-founded method for processing symbolic data with neural methods.

We first define the dynamics in a mathematically precise way in section 2. It is shown how the specific approaches from the literature fit into the general framework. Afterwards, we consider the capability of coding and the closely related approximation capability. The main results are as follows: encoding is possible with very small networks. Decoding requires an a priori unlimited amount of resources. As a consequence it can be shown that mappings with symbolic input domain or

output domain can be approximated in an appropriate sense. Finally, we deal with the learnability of the network architecture. Since the techniques are based on involved methods from so-called statistical learning theory, we shortly explain the necessary background in a somewhat informal manner and point out the consequences for the scenario in which we are interested: it is possible to derive explicit bounds on the number of examples which guarantee valid generalization of learning algorithms for recursive mappings. Compared to standard feedforward networks, these bounds necessarily depend on the data which is used for training. This fact is due to the complex data structures which are involved in this scenario.

## 2 Network dynamics

First, the basic recurrent dynamics which enable standard networks to deal with symbolic inputs and outputs, respectively, are defined. Note that terms and logical formulas possess a natural representation via a tree structure: the single symbols, *i.e.* the variables, constants, function symbols, predicates, and logical symbols are encoded as unique values in some real vector space; these values correspond to the labels of the nodes in a tree. The tree structure directly corresponds to the structure of the term or formula; *i.e.*, subterms of a single term correspond to subtrees of a node. The node itself is equipped with the label which encodes the function symbol. As an example, the term  $f(g(a), f(a, a))$  would be represented by the tree



where  $(1, 0, 0)$  encodes the symbol  $f$ ,  $(0, 1, 0)$  encodes the symbol  $g$ , and  $(0, 0, 1)$  encodes the symbol  $a$ . In a similar way, logical formulas can be represented by tree structures where the logical connectives are encoded in the nodes.

In the following, we restrict the maximum arity of functions and predicates to some fixed value  $k$  which is known a priori. Therefore, data in which we are interested are trees where each node has at most  $k$  successors. We go a step further and assume that every node except for the empty tree has precisely  $k$  successors. Note that this is no restriction since we can simply expand nodes with less than  $k$  successors by empty trees without changing the information with respect to the symbolic term represented by the tree. Moreover, we assume that the alphabet the labels are taken from is the same for every node. This need not be the case if we are representing Boolean formulas with more than one domain for the variables, for example. However, we can use the union of all possible domains as a universal domain for the nodes. Again, we do not lose information. Therefore we will not consider terms or logical formulas but we will deal with tree structures with fixed fan-out  $k$  and node labels from a fixed real vector space in the following.

**Definition 1** A  $k$ -tree with labels in some set  $\Sigma$  is either the empty tree which we denote by  $\perp$ , or it consists of a root labeled with some  $a \in \Sigma$  and  $k$  subtrees, some of which may be empty,  $t_1, \dots, t_k$ . In the latter case we denote the tree by  $a(t_1, \dots, t_k)$ . We denote the set of  $k$ -trees with labels in  $\Sigma$  by  $(\Sigma)_k^*$ , and the restriction to  $k$ -trees of height at most  $t$  by  $(\Sigma)_k^{\leq t}$ .

The recursive nature of trees induces a natural dynamics for recursively encoding or decoding trees. More precisely, we can define for each mapping with appropriate arity an induced encoding or decoding, respectively, which consists in recursively applying the mapping to the single parts of a tree. A precise definition reads as follows:

**Definition 2** Denote by  $\Sigma$  a set. Any mapping  $f : \Sigma \times (\mathbb{R}^m)^k$  and initial context  $y \in \mathbb{R}^m$  induce a recursive encoding

$$f_y^{\text{enc}} : \Sigma_k^* \rightarrow \mathbb{R}^m, t \mapsto \begin{cases} y & \text{if } t = \perp \\ f(a, f_y^{\text{enc}}(t_1), \dots, f_y^{\text{enc}}(t_k)) & \text{if } t = a(t_1, \dots, t_k). \end{cases}$$

Any mapping  $g = (g_0, g_1, \dots, g_k) : \mathbb{R}^m \rightarrow \Sigma \times (\mathbb{R}^m)^k$  and set  $Y \subset \mathbb{R}^m$  induce a recursive decoding

$$g_Y^{\text{dec}} : \mathbb{R}^m \rightarrow \Sigma_k^*, x \mapsto \begin{cases} \perp & \text{if } x \in Y \\ g_0(x)(g_Y^{\text{dec}}(g_1(x)), \dots, g_Y^{\text{dec}}(g_k(x))) & \text{otherwise.} \end{cases}$$

Hence the encoding recursively applies a mapping in order to obtain a code for a tree in a real vector space. One starts at the leaves and recursively encodes the single subtrees at each level using the already computed codes of the respective subtrees as context. The recursive decoding is defined in a similar manner: the recursive application of some decoding function to a real vector yields the label of the root and codes for the  $k$  subtrees. Note that the mapping  $g_Y^{\text{dec}}$  might be only partial if the set  $Y$  is not reached during decoding. See Fig. 1 for an example of recursive encoding and decoding. Hence a simple mapping together with the recursive dynamics induced by the data structures provide an encoding or decoding, respectively. In the connectionistic setting, the two mappings used for encoding or decoding, respectively, can be computed by standard feedforward neural networks. For convenience, we provide a formal definition for feedforward networks:

**Definition 3** A standard feedforward neural network consists of a set of neurons  $\{1, \dots, N\} \subset \mathbb{N}$  and a connection structure  $\rightarrow$  so that  $i \rightarrow j$  only if  $i < j$ . The neurons without predecessor  $1, \dots, n$  are called input neurons, the neurons without successor  $i_1, \dots, i_o$  are called output neurons, the remaining neurons are hidden neurons. Each connection  $i \rightarrow j$  is equipped with a weight  $w_{ij} \in \mathbb{R}$  and each neuron  $i$  is equipped with a bias  $\theta_i \in \mathbb{R}$  and an activation function  $f_i : \mathbb{R} \rightarrow \mathbb{R}$ .

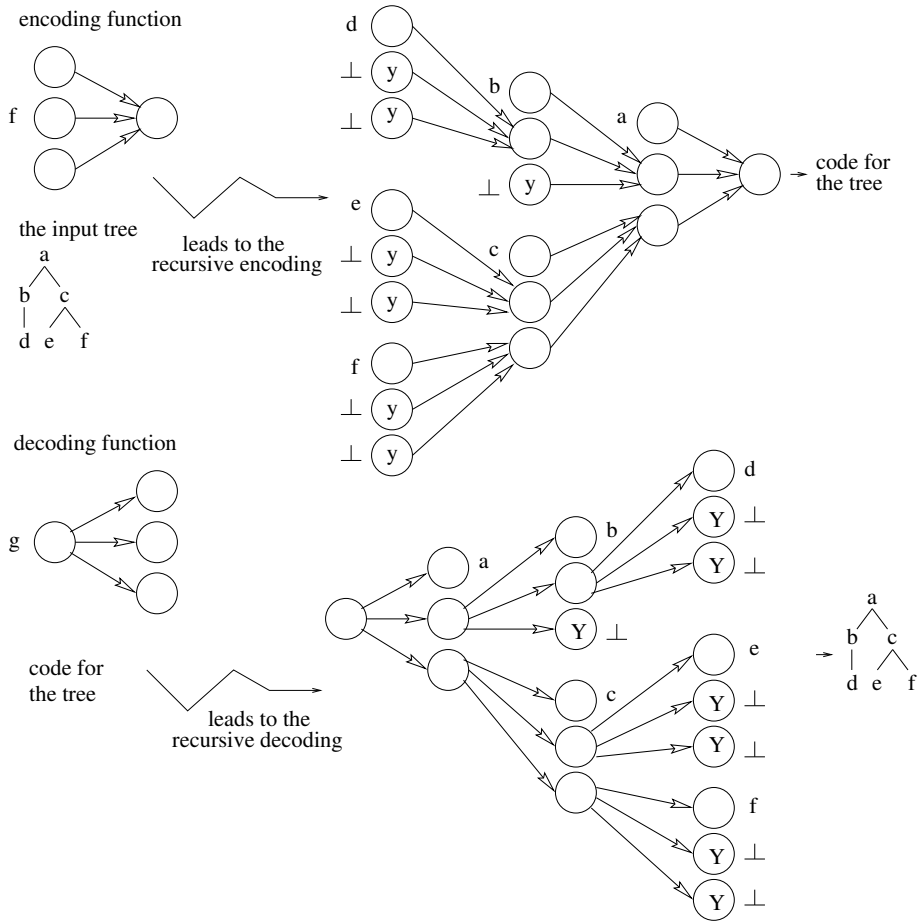


Fig. 1. Example for the recursive coding: applying an encoding mapping recursively to the subtrees yields a code for the tree in a real vector space. Applying a decoding mapping recursively to a real value yields the labels of the encoded tree.

Such a network computes a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}^o$ ,  $\mathbf{x} \mapsto (o_{i_1}, \dots, o_{i_o})$  where the activation  $o_i$  is recursively defined as

$$o_i = \begin{cases} x_i & \text{if } i \leq n \\ f_i(\sum_{j \rightarrow i} w_{ji} o_j + \theta_i) & \text{otherwise.} \end{cases}$$

Often, the network graph has a multilayered structure; i.e., there exists a division of the neurons into sets  $N_0, \dots, N_h$  so that  $i \rightarrow j$  iff  $i \in N_l$  and  $j \in N_{l+1}$  for some  $l$ . This is called a multilayer network with  $h - 1$  hidden layers.

A network architecture determines only the network graph without specifying the precise weights and biases. In a natural way, a network architecture stands for the set of functions which can be computed by a network of the specified architecture.

A typical multilayer feedforward network is depicted in Fig. 2. The circles stand for the single neurons. Note that each single neuron computes a very simple function: it

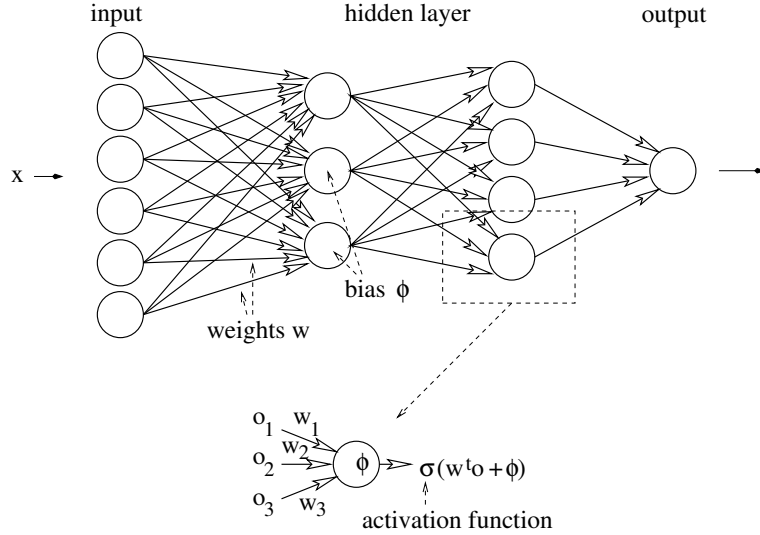


Fig. 2. A typical multilayer feedforward network

computes the weighted sum of the outputs of its predecessors; afterwards, it applies some activation function to this value. Popular choices for the activation function are the identity  $\text{id}(x) = x$ , the Heaviside function  $H(x) = 0$ , if  $x < 0$ ,  $H(x) = 1$ , if  $x \geq 0$ , the standard logistic activation function  $\text{sgd}(x) = 1/(1 + e^{-x})$ , or some general *squashing function*, *i.e.* a monotonic function  $\sigma$  which approaches 0 for small inputs and 1 for large inputs. Complex behavior arises in a network from the connection of the neurons. A connectionistic system dealing with structured data is obtained if we define the encoding or decoding function in Def. 2 via a neural network. We obtain networks with tree-structured inputs or outputs, respectively, in this way which encode symbolic data into a connectionistic representation or decode symbolic data from a connectionistic representation, respectively.

**Definition 4** An encoding network computes a function  $f_y^{\text{enc}}$  where  $f$  is computed by a feedforward network. A decoding network computes a function  $g_Y^{\text{dec}}$  where  $g$  is computed by a feedforward network.

In applications, data are to be processed further in an adequate way. We would like to learn some specific behavior rather than an encoding or decoding, only. However, note that the above encoding enables us to deal with real vectors of a finite dimension instead of symbolic data. Therefore we can apply standard connectionistic techniques to the encoded data. That means, we can combine neural encoding or decoding with a standard feedforward network in order to model complex behavior on symbolic data. Mathematically speaking, assumed we somehow manage to find mappings  $f$  and  $g$  so that  $f_y^{\text{enc}}$  and  $g_Y^{\text{dec}}$  constitute proper encoding and decoding, respectively, then we can compose those mappings with a standard network  $h$ .  $h \circ f_y^{\text{enc}}$  yields mappings with structured inputs,  $g_Y^{\text{dec}} \circ h$  yields mappings with structured outputs.

Now the question arises of how can we find appropriate encoding or decoding networks? There exist several possibilities in the literature. As an example, assume we

would like to learn a mapping which maps Boolean formulas without variables to their logical values. Assume the example  $(1 \wedge 1) \mapsto 1$  is available, 1 denoting the value TRUE. Since we deal with trees, we first encode the input as a tree structure, say  $2(1, 1)$ , the root label 2 encoding the connective  $\wedge$ . To be precise, we had to denote the tree by  $2(1(\perp, \perp), 1(\perp, \perp))$ . However, we will drop the empty node  $\perp$  in the following. Hence our training set consists of the pattern  $2(1, 1) \mapsto 1$ . Of course, we would need more examples for proper training in practice. However, this single example will do for demonstration purposes. We would like to find an encoding network  $f$  so that  $f_y^{\text{enc}}(2(1, 1))$  is a proper connectionistic representation of the tree; in addition we would like to find a feedforward network  $h$  which maps this encoded value to the desired output 1.

Various different methods for training have been proposed in the literature. An abstract description of the general recursive dynamics which we defined in a mathematically precise way in Def. 2 can be found in [32]; however, the focus of this so-called recursive reduced descriptions lies on the properties of the dynamics and no concrete implementation is suggested. The *tensor construction* (T) proposed by Smolensky in [40] provides a concrete implementation for the encoding and decoding. These are fixed mappings  $f$  and  $g$ , respectively, based on the tensor product of two vectors. Smolensky decomposes the problem of encoding structures into three subproblems: decomposing the structure via roles, *e.g.* the label and left and right subtree of a binary tree; implementing a superposition of the several roles, this is realized via direct sums and embedding in a vector space of appropriate dimension in Smolensky's tensor algebra; and implementing role/filler bindings, this is performed via the tensor product of specific vectors for the roles and the vector which represents the distributed representation of the filler. Altogether, a faithful encoding of structured data can be obtained. However, it is not necessary to know how exactly the tensor product looks like at this point. The property which is of interest for us is as follows: the mappings induced by  $f$  and  $g$  are proper encodings or decodings, respectively, that means they allow to encode and restore all data, *i.e.* all possible trees, perfectly. Hence we can simply use these encodings, encode the above input tree  $2(1, 1)$ , and train a standard feedforward network with conventional methods so that it outputs 1 for the connectionistic representation of the tree. However, Smolensky's tensor algebra has a significant drawback: the dimensionality of the connectionistic representation for the data increases in this encoding and decoding using tensor constructions. More precisely, the tensor product of two vectors of dimension  $n$  and  $m$ , respectively, has the dimension  $n \cdot m$ . Hence dimensionality increases very rapidly for nested structures. Therefore, the applicability of this approach is severely limited.

*Holographic reduced representation* which has been proposed by Plate in [31] uses ideas from the tensor construction. It proposes a fixed encoding and decoding, too. Again, the encoding problem decomposes into three subtasks: identification of the several roles; superposition of the various roles, which is implemented via the direct sum of their representations; and implementation of role/filler

bindings. The latter is implemented with so-called circular convolution or correlation, respectively. It has the advantage that this method does not suffer from the problem of increasing dimensionality. More precisely, the circular convolution of two  $n$ -dimensional vectors  $x = (x_0, \dots, x_{n-1})$  and  $y = (y_0, \dots, y_{n-1})$  is the vector  $x \otimes y = (\sum_{i=0}^{n-1} x_i y_{j-i})_{j=0}^{n-1}$  and their correlation is the vector  $x \odot y = (\sum_{i=0}^{n-1} x_i y_{j+i})_{j=0}^{n-1}$  where indices are taken modulo  $n$  in both cases. For vectors  $x$  and  $y$  it holds that  $x \odot x \otimes y \approx y$  plus some noise. If the coefficients of  $x$  and  $y$  are independently and normally distributed then the additional noise in the latter equality can be estimated. It turns out that circular correlation is an approximate inverse of circular convolution. Hence the encoding can have the form  $f : a(t_1, t_2) \mapsto r_1 \otimes a + r_2 \otimes t_1 + r_3 \otimes t_2$ ,  $r_i$  representing the respective roles, and decoding can be computed through  $g : x \mapsto (\text{clean-up}(r_1 \odot x), r_2 \odot x, r_3 \odot x)$ . The mapping ‘clean-up’ is usually an associative memory which compares the label which is slightly corrupted by noise to all possible labels in order to suppress the noise. The properties which are of interest for us are as follows: the encoding  $f$  and decoding  $g$  proposed by HRR are with high probability proper encodings for general trees. That means, one can encode and almost perfectly restore all data with these mappings  $f$  and  $g$ . More precisely, the output will be subject to some noise which arises from the encoding and decoding procedure. Usually, the noise is small enough so that the correct labels of the tree can be recovered if the height of the encoded tree is sufficiently small and some clean-up mechanism as mentioned above is added to the process. Hence we can use HRR as long as we do not have to deal with highly structured data. It allows us to find a proper connectionistic representation which can be used for further processing with standard networks. Note that there exist efficient ways to compute convolution and correlation using Fast Fourier transforms as pointed out in [31]. Moreover, one can prevent total corruption by noise for highly nested structures through so-called chunking, *i.e.* storing parts of the structure using pointers and additional memory.

The *labeled recursive autoassociative memory* [32,44] or LRAAM chooses neural networks for encoding and decoding, which have to be trained properly. The training procedure is very intuitive: the two networks are combined and trained so that the composition yields the identity on the given training patterns. That means,  $g_Y^{\text{dec}} \circ f_y^{\text{enc}} \approx \text{id}$  should hold on the given training data; in our case this reads as  $g_Y^{\text{dec}}(f_y^{\text{enc}}(2(1, 1))) \approx 2(1, 1)$ . Usually, this can be done via minimizing an appropriate error function, for example the quadratic distance between the desired and the real outputs of the mapping  $g_Y^{\text{dec}} \circ f_y^{\text{enc}}$ . Minimization can be performed with some standard minimization procedure such as a gradient descent. [32,44] use a so-called truncated gradient descent method which leads to formulas which are similar to the backpropagation (BP) procedure for feedforward network training. Details can be found in [32,44]. Assumed the above equality could be satisfied then the networks would perform proper encoding and decoding on our training data. Since neural networks generalize to unseen data, data which are similar to the training examples could be properly encoded and decoded, too. Afterwards, we could use the connectionistic representations for training standard feedforward networks on the specific

tasks. Note that the coding is adapted to our training data in this case. Hence small networks will be sufficient if our data are simple. We need larger networks for encoding complex data. Due to the training, the encoding focuses on regions of our data in which we are interested.

Folding architectures (FA) [12,43] or recursive networks combine an encoding part with a standard feedforward network directly. *I.e.*, they compute functions  $h \circ f_y^{\text{enc}} : (\mathbb{R}_k^m)^* \rightarrow \mathbb{R}^o$ , where  $h$  and  $f$  are given by standard feedforward networks. Hence they consider mappings with structured input domain; they do not consider symbolic outputs. How can we find a proper encoding in this situation? Given our training data, we know the desired output of certain values. In our case the equality  $h(f_y^{\text{enc}}(2(1, 1))) \approx 1$  should hold. One can define an appropriate error measure, *e.g.* the quadratic distance of the desired and the real outputs of the mapping  $h \circ f_y^{\text{enc}}$  which should be minimal. Again, this can be minimized with standard methods such as a gradient descent. [35] proposes a method called backpropagation through structure (BPTS) which constitutes a very efficient way of computing the gradients. Details can be found in this reference. Note, that the encoding part and the feedforward network are trained simultaneously. This yields an encoding which is not universal, *i.e.* it will not work for arbitrary input trees. Certain trees, which are to be mapped to similar values with  $h$ , may have similar or the same encoding under  $f_y^{\text{enc}}$ . This means that the encoding is fitted to the specific data and it is fitted to the specific learning problem [14]. Parts which are not relevant for  $h$  may be dropped in the encoded representations.

Table 1 summarizes the properties of the approaches we have mentioned. Note that *recurrent neural network* [14], which are widely used for times-series prediction, speech processing, forecasting, *etc.*, constitute a special case of folding networks.

|                   | T  | HRR  | LRAAM  | FA                                      |
|-------------------|--|--|--|---|
| encoding          | fixed  | fixed  | data-adapted   | data-adapted<br>task-adapted            |
| dimension         | increasing   | fixed  | fixed  | fixed                                   |
| training          | $f_y^{\text{enc}}, g_Y^{\text{dec}}: \text{fixed}$<br>$h: \text{BP}$ | $f_y^{\text{enc}}, g_Y^{\text{dec}}: \text{fixed}$<br>$h: \text{BP}$ | $g_Y^{\text{dec}} \circ f_y^{\text{enc}}: \text{BP}$<br>$h: \text{BP}$ | $h \circ f_y^{\text{enc}}: \text{BPTS}$ |
| structured input  | yes  | yes  | yes  | yes                                     |
| structured output | yes  | yes  | yes  | no                                      |

Table 1

Properties of various approaches which obey the proposed dynamics, the methods differ in the way in which the single network parts are trained. The models are: T = tensor construction, HRR = holographic reduced representations, LRAAM = labeled recursive associative memory, FA = folding architecture; the training algorithms refer to: BP = backpropagation, BPTS = backpropagation through structure.

Recurrent networks deal with time series or sequences, *i.e.* input trees with fan-out 1. They compute a function of the form  $h \circ f_y^{\text{enc}} : (\mathbb{R}^m)_1^* \rightarrow \mathbb{R}^o$ . They are commonly trained with a gradient descent method such as backpropagation through time and variants assumed the sequences are given a priori, or real time recurrent learning for on-line learning in robotics, for example [14,51].

These approaches provide concrete tools for processing structured data with neural networks. We believe that a thorough theoretical investigation is mandatory in order to establish these methods. Several points are worth considering: each method comes with a specific training method which complexity determines our training time. We will not consider this point in the following, although it is a very interesting topic. Actually, there are correlated questions which are not yet solved for standard feedforward networks or recurrent networks and, certainly, the investigation in our situation will not become easier: several results show the NP-hardness of neural network training, see *e.g.* [8,9,17,39] and references therein. Additionally, training recurrent networks suffers from numerical problems as discussed *e.g.* in [3]. Recent approaches identify specific situations for recurrent or folding networks where training is easy due to the lack of local minima [11]. However, since the complexity of training is not yet understood for standard feedforward and recurrent networks, we will not tackle this question in the context of structure processing networks in the following.

Two other questions are of particular interest: are the approaches universal approximators, *i.e.* can they represent everything they are intended to represent? Is valid generalization guaranteed, *i.e.* can the methods learn everything from a finite set of examples they are intended to learn? These questions can be answered in a slightly different way for the various approaches. Assumed the encoding and decoding is fixed as in T and HRR, it is to be shown that precisely the proposed encoding and decoding are proper codings. Since this question is specific for the respective approach we refer to the respective specific literature [31,40]. Assumed the encoding and decoding are trainable as in LRAAM and FA, it is to be shown that networks can perform proper encoding with sufficient resources. We will deal with this question in Section 4. Assumed the encoding and decoding is fixed then every learning task is a standard learning task for standard feedforward networks on the connectionistic representation of the data. Hence we can simply refer to the respective investigation for standard feedforward networks. Assumed encoding or decoding can be trained then we have to provide guarantees for proper generalization. We will consider this topic in Section 5.

### 3 Capacity of the architecture

Having investigated the dynamics, we introduce a quantity which is useful in various different mathematical considerations. Formally speaking, we have defined

function classes in Def. 2 capable of dealing with symbolic inputs or outputs, respectively: the class of functions of the form  $h \circ f_y^{\text{enc}}$  where  $f$  and  $h$  are neural networks with at most, say, 42 neurons, as an example. Now it is interesting to see how these function classes look like. An interesting quantity is the richness or capacity of the class, for example. How could it be measured? There exists a very useful quantity, the so-called Vapnik-Chervonenkis dimension or VC dimension, for short, for this purpose [1,41,49]. Informally speaking, the VC dimension is the largest number of points so that every possible binary function on these points is contained in the function class or in its restriction to these points, respectively. *I.e.* restricted to these points, every possible binary function can be implemented with the function class and hence the class possesses the maximum capacity with respect to these points.

**Definition 5** *The VC dimension  $VC(F)$  of a function class  $F$  mapping from some set  $X$  into binary values  $\{0, 1\}$  is the largest cardinality (possibly infinite) of a set of points which can be shattered. A set of points is shattered if for every binary valued function  $d$  on the points a function  $f$  from the class exists which coincides with  $d$  on these points.*

Note that  $VC(F)$  is defined for binary-valued function classes only. For real-valued function classes we will identify outputs larger than 0 with 1 and negative outputs with 0 and refer to this associated binary valued class. We will use the same notation, for convenience. Note that the VC dimension is not defined if we deal with tree-structured outputs.

What is the benefit of this definition? As already said, the VC dimension measures the maximum number of points with maximum richness of the function class. This number enables us to estimate the overall richness of the function class: so-called Sauer's Lemma (see *e.g.* [49]) tells us that the number of functions which can be implemented on an arbitrary number of points (say  $m$ ) can be estimated as a polynomial in  $m$  of degree given by the VC dimension. As a consequence an estimation of the overall capacity for arbitrary points is possible based on the VC dimension.

In particular, the VC dimension is a key quantity in statistical learning theory which is due to the following reason: statistical learning theory considers the question of proper generalization. Assumed we would like to learn an unknown function from a specific function class based on a finite set of examples; then we would like to have guarantees that we indeed found the desired function or some function which is similar. Obviously, this is possible if and only if the finite number of examples provides enough information in order to determine the desired function almost uniquely in the class. Assumed we search for a function in a small class; a small number of examples will be sufficient to distinguish the desired function from the remaining class members. Assumed we search in a very rich class. Then we will need a huge example set in order to determine what we are searching for. Since the capacity of the function class can be measured in terms of the VC dimension, the

number of examples which are required for valid generalization is related to the VC dimension of the function class, too.

Luckily, upper and lower bounds on the VC dimension for recurrent or folding architectures can be found in the literature or they can be derived from the arguments given therein, respectively [10,17,23]. Denote by  $F$  the function class in which we are interested. We assume that it is given by a recurrent or folding architecture with  $N$  neurons and  $W$  weights. Denote by  $\sigma \in \{H, \text{id}, \text{sgd}, g\}$ ,  $g$  being some piecewise polynomial function of degree at least 2, the activation function of the neurons,  $k$  the fan-out of the input trees, and  $t$  the maximum height of input trees. Then one can obtain bounds of the form

$$VC(F|(\mathbb{R}^m)_k^{\leq t}) \leq \begin{cases} p_1(W, N, \ln t) & \text{if } \sigma = \text{id}, \text{ or } k = 1, \sigma = H, \\ p_2(W, N, t) & \text{if } k \geq 2, \sigma = H \text{ or } k = 1, \sigma \in \{\text{sgd}, g\}, \\ p_3(W, N, 2^t) & \text{otherwise} \end{cases}$$

for some polynomials  $p_1, p_2, p_3$ . Here,  $F|(\mathbb{R}^m)_k^{\leq t}$  denotes the function class restricted to input trees of fan-out  $k$  with height at most  $t$  and labels in  $\mathbb{R}^m$ . The above activation functions cover most activation functions which are used in practical applications. Note that different activation functions or fan-out  $k$  lead to differences in the bounds with respect to the order of  $t$ . This is due to the various situations which arise: if the perceptron activation  $H$  is used, recurrent networks constitute finite automata or tree automata, respectively, since the number of interior states is limited. If the identity is used, linear recursive filters arise. For the logistic function the capacity of both scenarios can be combined since the logistic function contains a linear part at the origin as well as saturated parts if the inputs approach  $\pm\infty$ . Moreover, a fan-out  $k \geq 2$  allows branching, *i.e.* an exponential increase of the number of input labels with respect to the height  $t$  compared to  $k = 1$ , *i.e.* sequences. Lower bounds in the respective cases are

$$VC(F|(\mathbb{R}^m)_k^{\leq t}) \geq \begin{cases} p_4(W, \ln t) & \text{if } \sigma = \text{id}, \text{ or } k = 1, \sigma = H, \\ p_5(W, t) & \text{otherwise} \end{cases}$$

for other polynomials  $p_4$  and  $p_5$  of degree at least one. Note that the lower bounds depend on  $t$ , *i.e.* the maximum input height, in the above cases and hence yield an infinite VC dimension for arbitrary inputs, *i.e.* inputs of arbitrary height. We will use these bounds in order to obtain bounds on various resources and the generalization ability later on.

## 4 Approximation ability

We are interested in the representation capability of encoding and decoding networks. It is possible to focus on various aspects: one can consider the question as to whether proper encoding and decoding can be performed. Alternatively, possibilities to approximate mappings where either the inputs or the outputs may be symbolic data can be examined. Of course, the second question is related to the first one: assumed we could manage proper encoding and decoding then a combination of the encoding and decoding with a standard feedforward network could approximate any complex mapping. This is due to the fact that standard feedforward networks are universal approximators and the problem is an approximation task for feedforward networks on the connectionistic representations after encoding.

Actually, a couple of results concerning the approximation capabilities exist in the literature. Since the approaches [17,18] mainly focus on the capability of approximating functions, the consequences concerning the encoding and decoding capabilities are often hidden in the proofs. We will explain the main ideas of encoding and decoding possibilities and summarize the consequences for the approximation ability. Proofs which are not yet contained in the literature can be found in the appendix.

It turns out that two relevant situations of different complexity can be identified. Data may come from a symbolic domain, *i.e.* the labels of the trees come from a finite alphabet, say  $\Sigma = \{1, \dots, B\}$ . These values enumerate the symbols for constants, functions, *etc.* Assumed we are interested in Boolean formulas without variables,  $\Sigma$  could enumerate the two possible truth values and all logical connectives. We refer to data from  $(\Sigma)_k^*$ ,  $\Sigma$  being a finite alphabet, as *purely symbolic* or *discrete data*. Alternatively, the labels may consist of real vectors. One possible example are logical formulas with truth values from the interval  $[0, 1]$  as in fuzzy logic. A formula could have the form  $(0.5 \wedge 0.8)$  as an example with overall truth value in  $[0, 1]$  depending on the interpretation of  $\wedge$ . Trees representing such formulas would be contained in  $(\mathbb{R})_2^*$ . Labels in  $[0, 1]$  correspond to truth values, operators like  $\wedge$ ,  $\vee$ , *etc.* can be encoded using numbers 2, 3, *etc.*, for example. In general, data could be contained in  $(\mathbb{R}_m)_k^*$ . We refer to these kind data as *hybrid* or *real-valued data*.

How can those type trees be encoded in a finite dimensional real vector space? A major problem consists in the fact that the height of the trees may be a priori unlimited, hence an arbitrary amount of information may be stored in a tree. We can take this into account using either an a priori unlimited dimensionality of the real vector space. Alternatively, we can use unlimited precision of the computation and somehow store the information in the digits of real numbers. It is obvious that the capacity of each finite dimensional vector space with finite values, *i.e.* values computed up to a limited precision, is restricted. Therefore, computer simulation

can only approximate the situation with unlimited information, of course.

A canonic and ‘flat’ representation of trees is their prefix representation: the tree  $1(2, 2)$  would be represented by the sequence

$$[1, 2, \perp, \perp, 2, \perp, \perp],$$

the tree  $0.5(1(2), 0.3(0.2, 0.2))$  would be represented by the sequence

$$[0.5, 1, 2, \perp, \perp, \perp, 0.3, 0.2, \perp, \perp, 0.2, \perp, \perp].$$

Both representations include the empty tree  $\perp$ . This representation is unique assumed the fan-out  $k$  is fixed, in our case 2, *i.e.* we enlarge the trees by the empty tree  $\perp$  at appropriate places. Can these prefix representations be embedded into a finite dimensional vector space? There are mainly two possibilities: we can write the single elements of the sequence into the coefficients of one vector. The dimension of this vector is a priori unlimited according to the unlimited height of the trees. However, assumed we limited the height, a vector space of a fixed, possibly huge dimension would be sufficient:

$$(0.5, 1, 2, \perp, \perp, \perp, 0.3, 0.2, \perp, \perp, 0.2, \perp, \perp, 0, 0, \dots, 0)$$

would encode the latter tree. We refer to this encoding as *vector encoding*. Alternatively, we can write the elements in the digits of one real number provided the elements come from a finite alphabet. This latter condition is necessary in order to ensure that each element requires only a finite number of digits. Assumed the above tree contains labels in  $\{0, 0.1, \dots, 2\}$ , 0 representing the empty tree, then an alternative representation would be the number

$$0.05102000000003020000020000$$

where each label occupies two digits in the representation. We refer to this encoding as *real-value encoding*. Obviously, the same method could not be applied to hybrid data, *i.e.* real-valued labels with infinite precision.

Can these encodings or an approximation thereof be computed with encoding and decoding networks, respectively, and what are the required resources? Yes, they can be computed and the resources depend on the respective situation. The technical details behind the proofs are only of minor interest. The main ideas are the following: standard networks can approximately compute standard operations like a linear map, multiplication, truncation, Boolean operations, *etc.* This is due to the fact that linear maps are given for free – each neuron first computes a weighted sum of the predecessor’s outputs. It is well known that truncation and Boolean operation can be computed with combinations of Heaviside functions. Common network activation functions are squashing functions, *i.e.* they approximate the Heaviside function if restricted to an appropriate input domain. Appropriate scaling of the

inputs, *i.e.* appropriate linear mappings allow us to do so. Moreover, arbitrary polynomials, in particular linear functions and multiplication, can be approximated with standard activation functions like the logistic function as well. This is due to the fact that the Taylor expansion is an infinite sum of monomials of all possible degrees. An appropriate scaling and linear combination allows us to focus on the required degree. Details can be found in [17], for example.

We recall the main results concerning the possibility of encoding which are implicitly contained in [17]. The dimensionality of the vector space which contains the connectionistic representations is of particular interest. We refer to this dimension as *encoding dimension*. For convenience, we assume that the activation function is the standard logistic function or a similar function.

**Theorem 6** *There exist encoding networks  $f_y^{\text{enc}}$  for the following tasks:*

- *Every finite set of purely symbolic data in  $(\Sigma)_k^*$  can be encoded with encoding dimension 2 in a real-value encoding. The number of neurons depends on the fan-out  $k$ , only.*
- *Every finite set of hybrid data in  $(\mathbb{R}^m)_k^*$  can be encoded with encoding dimension 2 in a real-value encoding. The number of neurons depends on the fan-out  $k$  and the maximum height of the trees or, alternatively, the number of trees to be encoded.*
- *Every set of hybrid data with limited height  $t$  in  $(\mathbb{R}^m)_k^{\leq t}$  can be encoded with encoding dimension which is exponential in  $t$ . The encoding is a vector encoding. The number of neurons is exponential in  $t$ .*

*It is not possible in principle to encode hybrid data of unlimited height in a real-value encoding or vector encoding if the activation function of  $f_y^{\text{enc}}$  is continuous.*

Hence encoding is possible and requires only a very small amount of resources for purely symbolic data. For binary trees, for example, 11 neurons with the logistic activation function are sufficient [17]. The latter limitation in the theorem arises from the well-known mathematical fact that a real vector space cannot faithfully be embedded in a vector space of smaller dimension with a continuous function. However, we had to do so in order to represent all labels of a tree of unlimited height properly.

The capability of encoding yields to immediate results concerning the possibility to approximate mappings with structured inputs. In fact, the above encoding results are hidden in the approximation results from [17] which read as follows:

**Theorem 7** *Assume  $F : (\mathbb{R}^m)_k^* \rightarrow \mathbb{R}$  is a (measurable) function. Assume an accuracy  $\epsilon > 0$ , confidence  $\delta > 0$ , height  $t$ , and probability measure  $P$  on  $(\mathbb{R}^m)_k^*$  are chosen. Then there exists an encoding network  $f_y^{\text{enc}}$  and a feedforward network  $h$  with the following properties:*

- $h \circ f_y^{\text{enc}}$  approximates  $F$  up to inputs of small probability:

$$P(x \mid |h(f_y^{\text{enc}}(x)) - F(x)| > \epsilon) < \delta,$$

*i.e. the probability of trees where the network's output  $h(f_y^{\text{enc}}(x))$  differs more than  $\epsilon$  from the desired output  $F(x)$  is smaller than  $\delta$ . The encoding dimension is two. If data are purely symbolic, the number of neurons of  $f$  can be restricted with respect to  $k$ . Assumed the number of inputs  $x$  was finite, say  $p$ , an order of  $p + k$  neurons would be sufficient for purely symbolic data and an order of  $p^2$  neurons would be sufficient for real-valued data.*

- Assumed  $F$  is continuous,  $h \circ f_y^{\text{enc}}$  approximates  $F$  on compact inputs of height at most  $t$ , *i.e.*

$$|h(f_y^{\text{enc}}(x)) - F(x)| \leq \epsilon$$

*for all inputs  $x$  of height at most  $t$  with labels from a compact set. The encoding dimension is two for purely symbolic data and exponential in  $t$  for real-valued data. Less neurons will not do in general.*

- Assumed the inputs come from  $(\{1\})_1^*$  (*i.e. unary sequences*) then

$$|h(f_y^{\text{enc}}(x)) - F(x)| \leq \epsilon$$

*for all inputs  $x$ . It is not possible to approximate every mapping on symbolic inputs of unlimited height with labels from a binary alphabet.*

We do not want to explain the term measurability at this point. Which is of interest for us is the fact that measurability is a very weak condition. Every mapping which occurs in practice is measurable. Hence universal approximation is possible. Explicit bounds on the resources exist if a finite set of data is dealt with. Again, the bounds are particularly well behaved for purely symbolic data. Approximation for arbitrary height is not possible in general.

Actually, this latter topic touches on computability issues due to the following reason: assumed we dealt with linear trees, *i.e.* sequences. Assumed the length of the sequences would be unlimited. Assume the network behavior would be important after each time step, no matter the length of the sequence. Then one can consider the network as a classical computing mechanism like a Turing machine and the entries of the sequence as input. Actually, there exists a couple of approaches which relate recurrent networks to classical mechanisms like finite automata or Turing machines. It has been shown in the literature that Turing machines can be simulated with networks with various activation functions [21,22,38]. In addition, direct simulations of automata are possible which even show some robustness to noise [6,26,30]. A generalization of some of the latter approaches to folding networks can be found in [5]: one can relate folding networks to so-called tree automata.

However, we do not want to go into details concerning this topic. We are mainly interested in approximating and learning functions which do not come from a symbolic mechanism or where it is not known whether they come from a symbolic

mechanism. We are interested in approximation capabilities as they occur in the so-called PAC framework (we will explain this framework in Section 5). This framework focuses on the possibility of learning mappings from a finite set of examples so that the output is close to the desired function. In particular, the term ‘close to’ is interpreted in a probabilistic framework. This implies that the focus does not lie on arbitrarily long sequences or high trees, but on data with restricted length or height as it occurs in practice. In all practical applications the maximum height is limited; moreover even in theory, one can find for all probabilities  $p$  and small values  $\epsilon$  a height  $t$  such that trees which are higher than  $t$  have the probability at most  $\epsilon$ .

We are interested in the converse direction: is proper decoding possible and can mappings with structured outputs be approximated? We assume that real-value encoding or vector encoding according to Theorem 6 is available. Since we cannot expect to precisely restore the trees, we refer to proper decoding if the following holds: the structure of the decoded trees is correct and the single labels differ from the labels of the encoded tree by only a small value  $\epsilon$ . For convenience, we assume that additional values  $b$  and  $e$  indicate the beginning and end of a tree in the above real-value encoding or vector encoding. The representation

$$[0.5, 1, 2, \perp, \perp, \perp, 0.3, 0.2, \perp, \perp, 0.2, \perp, \perp].$$

for the tree  $0.5(1(2), 0.3(0.2, 0.2))$  would become  $[b, 0.5, b, 1, b, 2, b, \perp, e, b, \perp, e, e, e, b, 0.3, b, 0.2, b, \perp, e, b, \perp, e, e, b, 0.2, b, \perp, e, b, \perp, e, e, e, e]$ . This is very easy to achieve and leads to equivalent, though longer representations. The delimiters  $d$  and  $e$  have not yet been included into the encoding which has been discussed in the beginning of this section. This is due to the fact that the delimiters are not necessary for unique encoding and they are not contained in the respective proofs to which we referred. However, the delimiters could be easily integrated into the respective proofs and they make things much simpler for proper decoding. They enable explicit counting of the respective tree level when decoding.

One can show that proper decoding is possible based on this representation:

**Theorem 8** *Assume a finite set of encoded trees is given. The encoding may consist of real-value codes or vector codes. Then there exists a decoding network  $g_Y^{\text{dec}}$  which properly decodes these trees. The number of neurons is necessarily exponential with respect to the maximum height of the trees.*

Hence decoding is possible in principle. However, decoding requires an increasing amount of resources. This does not only hold for real-valued trees or vector encoding but for symbolic trees encoded arbitrarily, *i.e.* such that the trees correspond to arbitrary but distinct vectors, too. The proof, provided in the appendix, is based on an argument which involves the VC dimension. The rough idea is the following: one can associate to each decoding function a function class which necessarily shatters a huge amount of data, *i.e.* where lower bounds on the VC dimension can be established. Hence every function which is used for decoding necessarily pos-

sesses some minimum complexity. This complexity is not achieved by standard networks with a fixed number of neurons. Note that this argument does not rely on the specific encoding mechanism or the decoding function which is used. It is a general argument which puts restrictions on approaches like the tensor construction (T) or holographic reduced representation (HRR), too. We believe that this can be seen as a theoretical motivation for the necessity of increasing dimension in T and a clean-up mechanism in HRR. Moreover, this result yields a theoretical foundation for the success of folding networks compared to the LRAAM: they only deal with the easier task of encoding, compared to the LRAAM which deals with encoding and decoding. Hence they can solve their tasks with smaller networks. The LRAAM would need an increasing amount of neurons even for purely symbolic data. However, if only a specific form of trees which are almost linear, *i.e.* the number of leaves is restricted a priori, is dealt with, a proper encoding and decoding is possible with restricted resources as shown in [18].

Results on the ability of recurrent networks to approximate mappings with structured outputs can be derived immediately:

**Theorem 9** *Assume  $F : \mathbb{R}^n \rightarrow (\mathbb{R}^m)_k^*$  is a (measurable) function. Assume  $P$  is a probability measure on  $\mathbb{R}^n$ . Assume some accuracy  $\epsilon > 0$  and confidence  $\delta > 0$  are chosen. Then a decoding network  $h_Y^{\text{dec}}$  and a feedforward network  $f$  exist so that the following holds:*

- $h_Y^{\text{dec}} \circ f$  approximates  $F$  on inputs of high probability, *i.e.*

$$P(x \mid |h_Y^{\text{dec}}(f(x)) - F(x)|_m > \epsilon) < \delta,$$

where  $|\cdot|_m$  outputs the maximum Euclidian distance of the labels of the trees. Even if a finite set of data is dealt with, the number of neurons is exponential with respect to the maximum height of the trees.

- Assume  $F$  is continuous. Then  $h_Y^{\text{dec}} \circ f$  approximates  $F$  on all inputs from a compact set up to accuracy  $\epsilon$ . The encoding dimension is exponential with respect to the maximum height of decoded trees.

Hence mappings with structured outputs can be approximated, too. Lower bounds on the resources show that the situation is more difficult than the situation of structured inputs. Hence some kind of universal approximation ability is established for both, encoding and decoding. Bounds on the resources indicate, which situations are difficult to train and which situations are comparably easy. In particular, approaches which deal with encoding only seem more promising than approaches which deal with both, encoding and decoding. Note that the lower bounds hold for all coding mechanisms, *i.e.* folding networks and the LRAAM, where coding can be trained, and for HRR and T, where coding is fixed, too. However, the positive approximation results assume that the networks can be adapted to the specific task – hence they do not transfer to approaches like HRR where the specific fixed encoding and decoding functions in HRR are to be examined.

## 5 Generalization ability

Another important question is the following: can we learn an unknown function from a finite set of data where the inputs or outputs may be structured? In a first step it is necessary to find a formalization of learnability. A popular approach in machine learning is the notion of so-called *probably approximately correct* or *PAC-learnability* as introduced by Valiant [48]: a function class is PAC-learnable if and only if a learning algorithm exists with the following property: the learning algorithm outputs a bad function only with small probability assumed a sufficient number of training examples is provided. That means, PAC-learnability tells us that at least one good learning algorithm exists. Unfortunately it does in general not tell us how to find the algorithm. Hence we would like to go a step further. The standard algorithms used for neural network training minimize the empirical training error on the data. Hence it would be valuable to know that every algorithm which leads to a small training error yields valid generalization. In other words: we would like to have guarantees that the training error is representative for the real error of our function. The real error refers to the distance between the function which we have learned with the algorithm and the underlying function which we would like to learn; this distance is measured taken all possible inputs into account. This property is commonly referred to as the *uniform convergence of empirical distances* or *UCED property* [49].

The standard way to establish the UCED property is as follows: the so-called distribution-independent UCED property holds for real-valued function classes if and only if their VC dimension is finite [1,49]. Moreover, concrete bounds on the training set size such that valid generalization is guaranteed can be found. The bounds constitute a polynomial in the desired accuracy and confidence, and the VC dimension. Here, the notion ‘distribution-independent’ means that the bounds are independent of the concrete underlying and usually unknown input distribution. This method can be applied for standard feedforward networks, as an example: they have a finite VC dimension and hence the UCED property together with concrete distribution-independent bounds are provided.

Unfortunately, the argumentation is more difficult in our situation: the first problem consists in the fact that we do not consider real-valued function classes. We consider complex structured outputs. The second problem is that even if we restricted ourselves to, say, folding networks with real-valued outputs, the VC dimension would be infinite. Hence we can conclude: the distribution-independent UCED property does not hold for the neural architectures considered in this situation. The situation is even worse. The distribution-independent UCED property is equivalent to distribution-independent PAC-learnability [49]. *I.e.* the VC dimension is finite if and only if *one* learning algorithm exists which yields valid generalization with distribution-independent bounds on the generalization error, both facts are equivalent to the fact that the empirical error of *every arbitrary* learning algorithm is

representative for the generalization error with distribution-independent bounds for the difference between both terms. Hence distribution-independent PAC learnability does not hold in our situation, either [17]. The consequences are as follows: assumed we deal with structured inputs of arbitrary height. Then it is impossible to find data-independent bounds on the number of examples which guarantee valid generalization no matter which learning algorithm is used. Due to the difficulty of our data, training may take longer than expected for every learning algorithm and every bound on the time we did expect.

However, since our data are complex, we could agree that training may take more time for particularly complicated data. We would expect that more training data are required if the trees have a height of 1 mio. instead of a height of 10. That means we would be satisfied with data-dependent bounds. Since distribution-independent learnability cannot be guaranteed we can try to establish the distribution-dependent UCED property or UCED property, for short. *I.e.* we would be satisfied with data-dependent bounds on the generalization error where properties of the input distribution or the concrete training data occur in addition to the desired accuracy and confidence and parameters which describe the architecture.

Unfortunately, the arguments from statistical learning theory which we would need at this point are somewhat involved and lie beyond the scope of this article. The main ideas concerning the situation in which we are interested are as follows: it is possible to obtain bounds on the VC dimension if we restrict to input trees of height at most  $t$ , as mentioned before. Hence learnability can be guaranteed for the scenario of restricted input height with standard arguments. In each concrete situation, *i.e.* for each concrete probability measure, high trees become less likely. Assumed we could estimate the likelihood of high trees a priori then we could drop unlikely situations and derive the standard bounds for the remaining situations. This leads to first results:

**Theorem 10** *The UCED property can be established for encoding networks with a standard activation function and each probability measure  $P$  on trees so that the probabilities of trees of height larger than  $t$  can be estimated a priori for each  $t$ .*

‘Standard activation function’ refers to any activation function such that bounds on the VC dimension which depend on the maximum input height can be derived, *e.g.* the standard logistic function. For concrete bounds on the generalization error in this scenario see [16].

Assume prior knowledge on the underlying probability measure is not available. Then we could estimate the probability of large input trees from the empirical distribution on our concrete training sample. Assumed training data are short sequences. Then we are lucky and valid generalization is very likely. Assumed a huge amount of long sequences can be found. Then we would demand more examples for proper generalization. This argumentation can be formalized within the so-called luckiness

framework [36]. This framework allows to stratify a learning setting according to a concrete so-called luckiness function. The luckiness function measures important quantities which are related to the expected generalization ability. We expect better generalization if we are lucky, in unlucky situations the generalization ability may be worse. It is important to notice that the luckiness is measured after training, *i.e.* posterior bounds may be derived in this setting. In our case, the height of input trees serves as a luckiness function. Note that the concrete training set is representative for the underlying, though unknown input distribution. Hence we can expect that high trees have a small probability and hence the generalization error depends on small trees only, if only small trees occur in a concrete training setting. One concrete result reads as follows [16]:

**Theorem 11** *Assume an encoding network has been trained on a training set where trees of height at most  $t$  are contained. Then the empirical error on the training data deviates from the generalization error at most by a term which depends on the number of training patterns, the number of weights in the network, the desired confidence, and  $t$ .*

Hence posterior bounds can be obtained which depend on the concrete training data. Still, the results are not yet satisfactory for two reasons: we would like to obtain bounds which do not rely on the worst example in the training set but a representative fraction of the training data. Fortunately, one can modify the above argumentation so that only a large part of the training data has to be taken into account. This yields bounds which depend on the maximum height of a large part of the data and no longer on the worst example [15]. Second, we would like to consider structured outputs. Learning theory provides a formalism which allows us to do so: we have to define a loss function which measures the distance of two trees. As an example, we could use a measure of the form

$$D(X, Y) = \begin{cases} \sum_i p_i |X_i - Y_i| & \text{if } X \text{ and } Y \text{ have the same structure} \\ \max & \text{otherwise} \end{cases}$$

where  $\max$  is some large value,  $X$  and  $Y$  are trees, and  $X_i$  and  $Y_i$  are their labels. The sum  $i$  is taken over all labels of the trees  $X$  and  $Y$ . The  $p_i$  are non-negative values which add up to 1. They emphasize the importance of the respective label in a tree. Usually, most emphasis lies on the root and the importance is smaller for deep nodes corresponding to a small value  $p_i$ . It is required that the values  $p_i$  add up to 1 (or any other positive constant) in order to avoid divergence of the above sum.

Now the original function class with structured outputs is substituted by the real-valued class which we obtain as follows: we compose the functions of the original class with the above loss. Afterwards, one can examine the learnability of the latter class. In doing so, the following result can be established in analogy to the argumentation of [15] (this reference provides the argumentation for recurrent net-

works, only. In order to obtain the same results for structured data, it is sufficient to substitute all bounds on the VC dimension in [15] by the bounds derived for structure-processing networks.) :

**Theorem 12** *Assume a composition of encoding, decoding and feedforward architectures is considered with  $W$  weights,  $N$  neurons, and the logistic activation function. Assume  $m$  training patterns are available with height at most  $t_x$  of all but a fraction  $1/m$  of the training set. Assume  $D$  is the above error measure. Choose  $t$  so that the factors  $p_i$  in  $D$  add up to at least  $1 - 1/(2m)$ . Then the squared distance of the training error and the generalization error is of the order*

$$\frac{p(W, N, 2^{t_x+t}) \lg(m \ln m)}{m},$$

*$p$  being some polynomial.*

Hence we obtain concrete bounds for learning scenarios where both, inputs and outputs are tree-structured. The bounds depend on the number of weights and the concrete training data only. Note that the bounds have to depend on the data due to our complex structures. For simple feedforward networks, valid generalization could be guaranteed even in the distribution-independent case, *i.e.* the bounds would depend on the architectures, only.

## 6 Conclusions

In this paper, we focused on specific dynamics which enable us to use standard connectionistic methods for symbolic data directly. Popular symbolic data like logical formulas and terms are tree-structured. The dynamics mirror the recursive dynamics of tree structures. They constitute a customary approach which is included in various concrete implementations such as RAAM, HRR, folding networks, *etc.* as pointed out earlier. The methods mainly differ in the way in which they are trained.

We believe that theoretical properties of the dynamics should be investigated in order to establish the various methods and to allow a well-founded comparison of their usefulness. Though the various methods differ considerably in the training method, they share several basic limitations and they share several basic guarantees. We have examined the following two key properties: we have considered the universal approximation capability and the generalization ability. The main results were very promising: universal approximation and learnability can be guaranteed, *i.e.* the approaches can be used in principle as universal learning mechanisms. However, decoding is more difficult than encoding and requires more resources – an exponential number compared to a fixed number in an extreme situation, the purely symbolic case. This tells us that we should restrict to encoding whenever we can.

We should use additional mechanisms like a clean-up for HRR and related methods if decoding is mandatory.

Learnability can be guaranteed with bounds which depend on the training data due to the complex dynamics. Though the concrete bounds are still very bad, these are promising results which establish the in-principle learnability as well as the order of the convergence of learning algorithms. Quite remarkably, an approach concerning the possibility of valid generalization could be obtained for structured outputs as well. Here additional appropriate error measures for tree structures should be established. Possible variations could take the symbolic nature of the data into account and incorporate invariances with respect to substitutions in formulas, as an example.

## References

- [1] M. Anthony and P. Bartlett *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.
- [2] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151-165, 1989.
- [3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157-166, 1994.
- [4] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
- [5] R. C. Carrasco and M. L. Forcada. Simple strategies to encode tree automata in sigmoid recursive neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):148-156, 2001.
- [6] M. Casey. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8(6):1135-1178, 1996.
- [7] M. W. Craven and J. W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In: *Proceedings of the Eleventh International Conference on Machine Learning*, Morgan Kaufmann, 37-45, 1994.
- [8] B. DasGupta, B. Hammer. On approximate learning by multi-layered feedforward circuit. in: H. Arimura, S. Jain, A. Sharma, editors, *Algorithmic Learning Theory'2000*, Springer, 264-278, 2000.
- [9] B. DasGupta, H. T. Siegelmann, and E. D. Sontag. On the complexity of training neural networks with continuous activation. *IEEE Transactions on Neural Networks*, 6(6):1490-1504, 1995.
- [10] B. DasGupta and E. D. Sontag. Sample complexity for learning recurrent perceptron mappings. *IEEE Transactions on Information Theory*, 42:1479-1487, 1996.

- [11] P. Frasconi, M. Gori, and A. Sperduti. Learning efficiently with neural networks: a theoretical comparison between structured and flat representations. in: W. Horn, editor, *ECAI 2000*, IOS Press, 301-305, 2000.
- [12] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data sequences. *IEEE Transactions on Neural Networks*, 9(5):768-786, 1997.
- [13] M. Gori. Learning in structured domains. in: M. Verleysen, editor, *European Symposium on Artificial Neural Networks 1999*, D-facto, 27-32, 1999.
- [14] M. Gori, M. Mozer, A. C. Tsoi, and R. L. Watrous. Special issue on recurrent neural networks for sequence processing. *Neurocomputing*, 15(3-4), 1997.
- [15] B. Hammer. On the generalization ability of recurrent networks. in: G. Dorffner, H. Bischof, and K. Hornik, editors, *Artificial Neural Networks – ICANN’2001*, Springer, 731-736, 2001.
- [16] B. Hammer. Generalization ability of folding networks. *IEEE Transactions on Knowledge and Data Engineerin*, 13(2):196-206, 2001.
- [17] B. Hammer. *Learning with recurrent neural networks*. Lecture Notes in Control and Information Sciences 254, Springer, 2000.
- [18] B. Hammer. Limitations of hybrid systems. In: *European Symposium on Artificial Neural Networks*, D-facto publications, 213-218:213-218, 2000.
- [19] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359-366, 1989.
- [20] J. E. Hummel and K. J. Holyoak. Distributed representation of structure: A theory of analogical access and mapping. *Psychological Review*, 104(3):427-466, 1997.
- [21] J. Kilian and H. T. Siegelmann. The dynamic universality of sigmoidal neural networks. *Information and Computation*, 128:48-56, 1996.
- [22] P. Koiran, M. Cosnard, and M. Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132:113-128, 1994.
- [23] P. Koiran and E. D. Sontag. Vapnik-Chervonenkis dimension of recurrent neural networks. *Discrete Applied Mathematics*, 86(1):223-237, 1998.
- [24] R. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, 1:1-38, 1989.
- [25] W. Maass and C. M. Bishop. *Pulsed Neural Networks*, MIT-Press, 1999.
- [26] W. Maass and P. Orponen. On the effect of analog noise in discrete-time analog computation. *Neural Computation*, 10(5):1071-1095, 1998.
- [27] C. von der Malsburg. The what and why of binding: The modeler’s perspective. *Neuron*, 24:95-104, 1994.
- [28] M. Masters. *Neural, Novel, & Hybrid Algorithms for Time Series Prediction*. Wiley, 1995.

- [29] C. W. Omlin and C. L. Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41-52, 41-52, 1996.
- [30] C. Omlin and C. Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 43(2):937-972, 1996.
- [31] T. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623-641, 1995.
- [32] J. Pollack. Recursive distributed representation. *Artificial Intelligence*, 46(1-2):77-106, 1990.
- [33] M. Riesenhuber and T Poggio. Are cortical models really bound by the ‘binding problem’? *Neuron*, 24:87-93, 1999.
- [34] T. Schmitt and C. Goller. Relating chemical structure to activity with the structure processing neural folding architecture. In *Engineering Applications of Neural Networks*, 1998.
- [35] S. Schulz, A. Küchler, and C. Goller. Some experiments on the applicability of folding architectures to guide theorem proving. In *Proceedings of the 10th International FLAIRS Conference*, 377-381, 1997.
- [36] J. Shawe-Taylor, P. L. Bartlett, R. Williamson, and M. Anthony. Structural risk minimization over data dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5), 1998.
- [37] L. Shastri. Advances in SHRUTI - A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. To appear in *Applied Intelligence*.
- [38] H. T. Siegelmann and E. D. Sontag. Analog computation, neural networks, and circuits. *Theoretical Computer Science*, 131:331-360, 1994.
- [39] J. Sima. Back-propagation is not efficient. *Neural Networks*, 9(6):1017-1023, 1996.
- [40] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1-2):159-216, 1990.
- [41] E. D. Sontag. VC dimension of neural networks. In C. Bishop, editor, *Neural Networks and Machine Learning*. Springer, 69-95, 1998.
- [42] J. P. Sougné. *INFERNET: A neurocomputational model of binding and inference*. PhD Thesis, Université de Liège, 1999.
- [43] A. Sperduti. Neural networks for adaptive processing of structured data. in: G. Dorffner, H. Bischof, and K. Hornik, editors, *Artificial Neural Networks – ICANN’2001*, Springer, 5-12, 2001.
- [44] A. Sperduti. Labeling RAAM. *Connection Science*, 6(4):429-459, 1994.
- [45] M.-C. Su, C.-J. Kao, K.-M. Liu, and C.-Y. Liu. Rule extraction using a novel class of fuzzy degraded hyperellipsoidal composite neural networks. In: *Proceedings of the IEEE International Conference on Fuzzy Systems*, volume 1, IEEE Press, 233-238, 1995.

- [46] I. A. Taha and J. Gosh. Symbolic interpretation of artificial neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 11(3):448-463, 1999.
- [47] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich. The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6):1057-1067, 1998.
- [48] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134-1142, 1984.
- [49] M. Vidyasagar. *A Theory of Learning and Generalization*. Springer, 1997.
- [50] H. Wersing, J. J. Steil, and H. Ritter. A competitive layer model for feature binding and sensory segmentation of features. *Neural Computation*, 13(2):357-387, 2001.
- [51] R. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. Rumelhart, editors, *Backpropagation: Theory, Architectures and Applications*. Erlbaum, 433-486, 1992.

## Appendix

### Proof of Theorem 8

**Proof:** We only deal with binary trees, the generalization to other trees is straightforward. We assume that the logistic activation function is used and hence Boolean connectives as well as multiplication, *etc.* can be approximated. The values are encoded via a prefix representation either in one real number, or in a vector with large dimension. In each recursive step, the label and codes for the left and right subtree are to be computed. If encoded in a vector of large dimension, the label can be obtained simply projecting to the respective components. If encoded in one real number, one can reconstruct the single digits recursively via multiplication by 10 and a test in which of the intervals  $[0, 0.05]$ ,  $[0.05, 0.15]$ , *etc.* the resulting number lies.

Afterwards, the left subtree is to be identified. This consists mainly in an identification and counting of the unique symbols indicating the beginning and end of a subtree, respectively. If the two numbers coincide for the first time, the left subtree is finished and the right subtree is represented by the remaining part of the code (up to additional symbols for the end of a tree, which do not alter the computation). Again, the counting of the beginning and end symbols can be performed by Boolean operations and multiplications of a specified neuron by 0.1 for  $b$  or 10 for  $e$ , respectively. The left subtree is completed if the value 1 can be found for the first time.

If we deal with vectors, this computation consists in several projections and a Boolean operations which can be computed via a standard network with a number of layers depending on the maximum number of nodes in the tree. If we deal

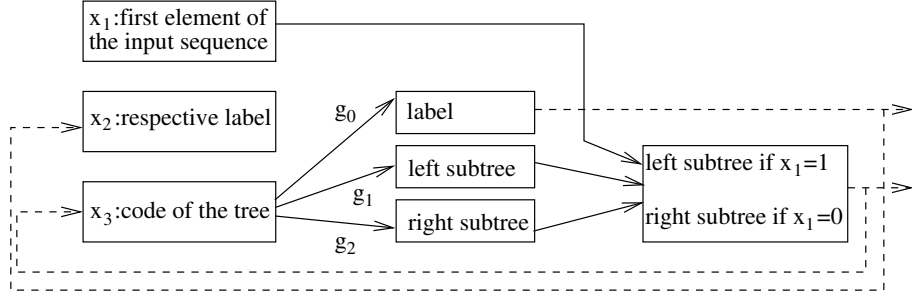


Fig. 3. Recurrent network induced by  $G$ ,  $x_1$  is an element of the input sequence,  $x_2$  only serves as output and provides the actual label of the tree,  $x_3$  stores the code for the actual subtree. The precise label and subtree which are the output in each recursive step depend on the input sequence.

with real numbers, we have to extract the single digits as described above, and we have to store the code for the left subtree in one single number. This can be easily computed by successively adding the respective bits multiplied by  $(0.1)^d$ ,  $d$  being the actual length of the computed part of the left subtree, to the already computed code.

These are the in-principle ideas of the simulation, details with respect to the possibility of counting and approximating standard operations can be found in [17]. We obtain a network with a number of neurons depending exponentially on the maximum height of the trees.

In order to derive a lower bound on the number of neurons, we derive the following general result:

Assume points in  $\mathbb{R}^m$  exist which are approximately decoded to all binary trees of height at most  $T$  with labels in  $\{0, 1\}$  with some  $g_Y^{\text{dec}}$ . If  $g$  is a feedforward network, the number of neurons is bounded from below by  $2^{\Omega(T)}$  if the activation function is the Heaviside function, the standard logistic function or piecewise polynomial, where  $\Omega T$  denotes a function which is at least linear in  $T$ .

In order to prove this result, assume a function  $g$  induces a proper decoding function. The function  $g = (g_0, g_1, g_2) : \mathbb{R}^m \rightarrow \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^m$  gives rise to a standard recurrent network  $G_y^{\text{enc}} : \mathbb{R}_1^* \rightarrow \mathbb{R} \times \mathbb{R}^m$  induced by  $G : \mathbb{R} \times \mathbb{R} \times \mathbb{R}^m \rightarrow \mathbb{R} \times \mathbb{R}^m$ ,

$$(x_1, x_2, x_3) \mapsto (g_0(x_3), (1 - x_1) \cdot g_1(x_3) + x_1 \cdot g_2(x_3)).$$

Note that  $g$  outputs in each step a label and codes for the left and right subtree of a tree. The role of  $G$  (see Fig. 3) is to select one of the two possible codes of the subtrees which are decoded via  $g$ : If the input is 0, the code of the left subtree is the output, if the input is 1, the right subtree is obtained. The first coefficient of the output of  $G$  only stores the actual label of the root, which is propagated back through  $x_2$  without being used in the further computation. Hence recursive application of  $G$  yields a path in the tree encoded in the initial context  $y$ , the respective label stored in  $x_2$ . More precisely, if  $g_Y^{\text{dec}}$  maps the value  $y$  to some tree  $t$  then  $\pi_1 \circ G_{(0,y)}^{\text{enc}}, \pi_1$

being the projection to the first component, maps any binary sequence of length  $i$  to some node in the  $i$ th level of the tree  $t$ ; the exact number of the node depends on the sequence: the sequence containing only entries 0 is mapped to the leftmost node in the  $i$ th level, the sequence with entries 1 is mapped to the rightmost node, the other sequences lead to the nodes in between. Now assume that points in  $\mathbb{R}^m$  exist which are approximately mapped to all trees of height  $T$  in  $\{0, 1\}_2^*$  with  $g_y^{\text{dec}}$ . Then the neural architecture  $\pi_1 \circ G_{(0, \cdot)}^{\text{enc}}$  shatters all binary sequences of length  $T$  with last component 1: one can simply choose the second part of the initial context corresponding to a vector  $y$  which encodes a tree of height  $T$  and leaves according to the dichotomy.

$G$  can be computed adding a constant number of neurons with some at most quadratic activation function and it can be approximated arbitrarily well adding a constant number of units with the logistic activation function. Consequently, the VC dimension of  $\pi_1 \circ G_{(0, \cdot)}^{\text{enc}}$  restricted to inputs of height at most  $T$  is limited by a polynomial in  $T$  and the number of neurons in  $g$ . The lower bound  $2^{T-1}$  on the VC dimension leads to the bound  $N = 2^{\Omega(T)}$  for the neurons in  $g$ .  $\square$

### Proof of Theorem 9

**Proof:** We first address the first item. If we deal with approximation in probability, the following steps do not change the considered situation much: we can divide  $F$  into mappings  $F_i$  each mapping some set to trees of just one fixed structure. We can assume that we have to deal with only a finite number of  $F_i$  since their input domain can be chosen such that the remaining part of the input set has arbitrary small probability. We can assume that each  $F_i$  is continuous since each measurable function can be approximated arbitrarily well by a continuous version. Moreover, we can assume that each  $F_i$  starts only from a compact interval, and hence is uniformly continuous, *i.e.* we can find for each  $\epsilon > 0$  a value  $\delta > 0$  so that the images of inputs which deviate at most  $\delta$  deviate by at most  $\epsilon$ . The mapping which maps the inputs to the code of the output trees which are to be decoded is uniformly continuous, too. Hence it can be approximated by a standard uniformly continuous feedforward network. Now we only have to find proper decodings for the codes. We need to approximate the labels of trees only with accuracy  $\epsilon$ . Hence we can output a constant tree on small intervals of the encoded values. That means: we only have to add a decoding network which decodes a finite set of inputs to the corresponding trees. We have already seen that this is possible. Combining the feedforward network with a test whether the input lies in the respective interval for the codes or the empty tree and this decoding network yields the desired result.

In order to prove the second item: decompose the compact input set  $C$  into compact subsets  $C_i$  so that  $F|_{C_i}$  maps to some fixed structure. Cover every  $C_i$  with a finite number of open intervals so that they do not touch sets  $C_j$  with  $j \neq i$ . The mappings induced by  $F$  on these intervals to the codes of the respective trees can be approximated in the maximum norm with a standard network. These mappings are

combined with a test which interval the input belongs to. The latter is performed in an appropriate way so that on precisely one  $C_i$  an output approximately 1 and for other  $C_j$  an output approximately 0 is obtained. Additionally, we add a decoding network which decodes vector-valued tree representations up to a finite height properly.  $\square$