Universität Ulm

Fakultät für Informatik

On the Correspondence between Neural Folding Architectures and Tree Automata

Andreas Küchler

Universität Ulm

Nr. 98-06

Ulmer Informatik-Berichte

Mai 1998

On the Correspondence between Neural Folding Architectures and Tree Automata

Andreas Küchler

Department of Neural Information Processing

University of Ulm, 89069 Ulm, Germany

kuechler@neuro.informatik.uni-ulm.de

Abstract

The folding architecture together with adequate supervised training algorithms is a special recurrent neural network model designed to solve inductive inference tasks on structured domains. Recently, the generic architecture has been proven as a universal approximator of mappings from rooted labeled ordered trees to real vector spaces.

In this article we explore formal correspondences to the automata (language) theory in order to characterize the computational power (representational capabilities) of different instances of the generic folding architecture. As the main result we prove that simple instances of the folding architecture have the computational power of at least the class of deterministic bottom-up tree automata. It is shown how architectural constraints like the number of layers, the type of the activation functions (first-order vs. higher-order) and the transfer functions (threshold vs. sigmoid) influence the representational capabilities. All proofs are carried out in a constructive manner and a detailed analysis of the space complexity of the construction schemes is provided.

We derive bounds for the principled node complexity of folding architecture implementations of tree automata when threshold transfer functions are used.

Tree languages used in former empirical investigations are identified to belong to the class of so-called pattern languages. We show that the folding architecture has the capability to recognize at least interesting subclasses if the defining pattern is constrained to be linear, i.e. if each variable occurs not more than once. Nonlinear pattern languages turn out to be beyond the representational capability of tree automata. The relationships between pattern languages, tree automata and the folding architecture give a new view on the interpretation of known empirical results. Further, this leads to the formulation of conjectures about the adequacy of the classical automata framework to characterize the full computational power of the folding architectures.

The construction schemes presented here can be effectively used to inject prior knowledge given in form of tree automata (or implicitly in form of the corresponding grammars or pattern descriptions) into an equivalent folding architecture with sigmoid transfer functions which is ready to be inductively refined on data by known training algorithms like backpropagation through structure. We briefly discuss the possible embedding of the folding architecture into a general knowledge injection-refinement-extraction framework.

Keywords Theory of Neural Networks, Recurrent Architectures, Computational Power, Tree Automata, Node Complexity, Pattern Languages, Knowledge Injection.

Contents

1 Introduction 1

1.1 Motivation . . . 1

1.2 Preview . . . 2

2 Preliminaries 4

3 Tree Automata 5

4 Folding Architecture 7

4.1 The Learning Tasks . . . 7

4.2 The Static View . . . 8

4.3 Processing Dynamics . . . 9

4.4 Approximation Capabilities . . . 10

4.5 VC-Dimension . . . 11

5 Single Neurons and Boolean Functions 12

5.1 First-Order Connections . . . 12

5.2 Higher-Order Connections . . . 15

6 Correspondences between DTA and FA 16

6.1 Two-Layered Folding Part . . . 17

6.2 One-Layered Folding Part . . . 19

6.2.1 Higher-Order Connections . . . 19

6.2.2 One Layer with First-Order Connections Cannot Implement Arbitrary State Transitions . . . 20

6.2.3 The State-Splitting Approach for First-Order Connections . . . 22

6.3 The Computational Power of the FA in the Threshold Case and the Relation to Context-Free Word Languages . . . 25

7 Bounds on the Node Complexity of FA Implementations of DTA 27

7.1 Upper Bounds . . . 27

7.2 A Way Directed to a Lower Bound . . . 28

8 On Pattern Languages and the Adequacy of the Automata Framework 32

8.1 Pattern Languages and the Language Boundary Induced by Occurrence Constraints on Variables . . . 32

8.2 The Experimental Setup . . . 37

8.3 Results . . . 37

8.4 Implications and Conjectures about the Full Computational Power of the FA . . . 39

9 Remarks on Combining Symbolic and Neural Learning 42

9.1 The Injection-Refinement-Extraction Framework . . . 42

9.2 Robust Injection of a DTA into the FA 43

10 Related Work 46

11 Conclusions 48

References 51

1 Introduction

1.1 Motivation

Classical neural network models – like feedforward multilayer networks – are successfully applied in static classification and recognition tasks. These models are not always appropriate in interesting domains with inherent dynamics like speech processing, plant control and time series prediction. Therefore there has been growing interest in dynamic recurrent neural network models which are capable to model nonlinear dynamic systems, but are limited to process variable-length sequences of continuous or discrete values as inputs (a selection of topics can be found in Gori et al. [27] and Giles et al. [23]). However, several application areas like molecular biology, language and document processing, geometric reasoning and symbolic computation require to deal with more structured objects like graphs or terms.

Inspired by this observation, recently the *folding architecture* (FA) provided with adequate supervised training algorithms (e.g. *backpropagation through structure*) was designed to solve inductive learning tasks on structured objects (Küchler and Goller [46, 26]). The generic architecture has been proven as a universal approximator of mappings from labeled rooted ordered trees to real vector spaces (Hammer [29], Hammer and Sperschneider [32]). First application perspectives already emerged – computational chemistry (Schmitt and Goller [61], Bianucci et al. [6]), pattern recognition (Francesconi et al. [15]), search control in theorem proving (Goller [25], Schulz et al. [62]), but up to now there is only few knowledge about the in principle representational capabilities of this architecture.

The FA may be viewed as a generalization of a class of discrete-time recurrent neural network models (RNN) which includes the so-called *simple-recurrent networks* (Elman [13]). In the past few years much work has been done in characterizing the computational power of RNN by linking them to concepts known from the theory of formal languages and automata, especially *finite-state automata* (FSA) (e.g. Kremer [45, 44], Omlin and Giles [55], Frasconi et al. [16], Goudreau et al. [28], Minsky [51]). Thus, a plausible access might be in the attempt to lift some of the known results and concepts from the RNN to the FA.

In this article we explore some formal correspondences between the FA and a certain class of *tree automata* (Gécseg and Steinby [20]) (which is a natural generalization of finite-state automata). The focus is on the *computational power* and the representational capabilities of different instances and variations of the generic architecture. Note that this issue is different from the question of *learnability*, but a necessary prerequisite for the latter. As the main result we prove that simple instances of the folding architecture have the computational power of at least the class of *deterministic bottom-up tree automata* (DTA) (Doner [12]). Proofs are developed in a constructive manner. We analyze and compare the space complexity of different construction schemes.

These construction schemes can be effectively used to inject prior knowledge given in form of tree automata (or in form of the corresponding grammars) into equivalent FA. The possible embedding into a general *injection-refinement-extraction* framework (as introduced by Shavlik [63]) is briefly addressed.

A further topic to be discussed is the so-called *node complexity* of folding architecture implementations of tree automata, i.e. the number of neurons required to implant a given tree automaton into the FA. Known bounds in the case of RNN with threshold transfer functions (see Horne and Hush [35, 36]) are lifted to the FA.

In the past a bunch of empirical investigations on the learnability of artificially generated term classification tasks have been conducted with the FA and gradient-based learning procedures (Schmitt [60], Goller [25], Schulz et al. [62], Küchler and Goller [46]). Even tasks intuitively considered to be hard for the FA were solved with fairly high generalization accuracy. We identify those term sets used in former experiments belonging to the class of so-called *pattern languages*. The relationship between pattern languages, tree automata and the FA gives a new interpretation of the known experimental results and makes the FA a very promising candidate for interesting inductive inference tasks in pattern recognition applications (Steinby [72], Schalkoff [59]). We are lead to the conjecture that the full computational power of the FA (equipped with sigmoid transfer functions) is far beyond that of DTA.

Throughout this article one can observe that most concepts and results known in the case of discrete-time recurrent neural networks, finite-state automata and sequence processing can be successfully lifted to tree processing and applied to explore the relationship between folding architectures and tree automata.

1.2 Preview

This paper is organized as follows. After having fixed the basic terminology required for later investigations we give a brief introduction into the concept of *bottom-up tree automata* (DTA).

The generic *folding architecture* (FA) is described in Section 4. Taking a static point of view the FA is roughly characterized by two multi-layer feedforward network components – the *folding part* and the *transformation part*. The folding part computes a non-linear combination of the input and a sequence of prior network states yielding the new state while the latter is used to map a network state to the output. We consider different kinds of activation functions (first-order vs. higher-order) and transfer functions (sigmoid vs. threshold) but allow only a homogeneous usage. A known result on the *approximation capability* of the FA is briefly reviewed. We show how a known upper bound result on the *sample complexity* – strictly speaking the so-called *VC-dimension* – of discrete-time recurrent neural networks (RNN) can be easily applied to the folding architecture.

The basic tools to be applied in later proof constructions are developed in Section 5. The key idea is that state transition functions of tree automata can be rewritten to Boolean formulae. We show how certain Boolean functions can be simulated by a single *neuron* (or *unit*) both in the first-order and the higher-order activation function case. For sigmoid transfer functions a discretization scheme has to be applied to the output value of the neuron. The constraints imposed by the discretization and the fact that compositions of Boolean functions are allowed lead to satisfiable sets of linear inequalities in the weights and the thresholds.

The main results concerning the correspondence between DTA and FA are derived in Section 6. We explore the influence of different architectural constraints. Most proofs are carried out in a constructive manner. Each construction schema is completed by a detailed analysis of the *space complexity*. The most important results are:

▷ The folding architecture equipped with a two-layered folding part, one unit as transformation part, first-order activation functions and sigmoid transfer functions suffices to simulate any given deterministic bottom-up tree automata.

▷ The transformation part is superfluous, i.e. the previous proposition also holds for a two-layered FA consisting only of the transformation part where one unit in the second layer is recruited to serve as output.

- ▷ Any given DTA can be simulated by a FA constituted by only one layer if *higher-order connections* are used.
- ▷ One layer with first-order connections *cannot* implement any arbitrary tree automata state transition function.
- ▷ However, the so-called *state-splitting technique* can be applied to transform any given DTA into an equivalent one which can be implemented by a FA with first-order connections and a one-layered folding part. Then, one output neuron is required as transformation part.

Thus, it is shown that the FA equipped with sigmoid transfer functions has at least the computational power of DTA. In the case of threshold functions the both machine models are equivalent (in terms of computational power). By a known result in the field of tree automata we illustrate an interesting relationship of the folding architecture to the class of *context-free word languages*.

A discussion of the so-called *node complexity* of FA implementations – i.e. the number of neurons required to implant a given DTA into an equivalent FA – follows in Section 7. In the case of threshold functions known facts can be borrowed from the field *circuit complexity theory* of *linear threshold circuits* and we are able to lift results known in the context of RNN and FSA to the FA and DTA. Upper bounds (for a two-, three- and four-layered folding part) are derived. A way for the estimation of a lower bound is worked out.

Section 8 reconsiders *inductive inference tasks* on artificially generated term languages which were formerly used in empirical investigations about the learnability and generalization capabilities of the FA respectively gradient-based learning procedures. We identify those term sets belonging to the class of so-called *pattern languages*. The FA is proven to possess the representational capability of at least the class of *linear* pattern languages. *Non-linear* pattern languages are beyond the class of *regular tree languages* (RTL) (which correspond to DTA). However, empirical results show that inductive inference tasks on non-linear pattern languages can be solved by the FA with high generalization accuracy. We formulate the *conjecture* that the full computational power of the FA is *beyond* that of DTA and discuss supporting arguments. This section is closed by a critical view on the adequacy of the formal automata/language framework to characterize the "full" computational power of the FA. We briefly discuss how the FA may be viewed from a *non-linear dynamic systems* perspective.

The possible embedding of the FA (and adequate training procedures) in the known knowledge *injection-refinement-extraction* framework is the subject of Section 9. The proof constructions developed in Section 6 can be effectively used to *inject* prior knowledge given in form of tree automata (or in form of corresponding grammars or defining patterns) into an equivalent FA which is ready for an inductive refinement. The proposed injection scheme shows *robustness* against *weight perturbations* and a small *noise* level on the output of the neurons.

Section 10 is the attempt to fit our own work into the existing research landscape. Most results and concepts developed in this article have been fruitfully lifted from the the context of discrete-time recurrent neural network models. The proof technique for implementing DTA by FA was essentially inspired by an existing system for the combination of symbolic decision tree learning with supervised training in multilayer feedforward neural networks.

We conclude this article with a summary on an abstract level, discuss open questions and future directions and reflect some consequences for the practical application of the folding architecture in pattern recognition tasks.

2 Preliminaries

In order to eliminate ambiguities and to facilitate detailed discussions let us first introduce and fix the basic concepts and terminology used throughout this paper in a formal way. The neural network architectures considered here were designed to operate on *terms* (or *trees*) — a special kind of structured domain.

Definition 1 (Term) Let Σ be a *ranked alphabet*, r be the corresponding ranking function $r : \Sigma \to 2^{\mathbb{N}_0}$ and $\mathcal{V}$ be a finite set of *variables* with $\Sigma \cap \mathcal{V} = \emptyset$. The enumerable set $\mathcal{T}(\Sigma, \mathcal{V})$ of *terms over* Σ *and* $\mathcal{V}$ is inductively defined as:

1. $t \in \mathcal{T}(\Sigma, \mathcal{V})$ if $t \in \mathcal{V}$ or $t \in \Sigma$ with $r(t) = 0$,
2. $f(t_1, t_2, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ if there is a symbol $f \in \Sigma$ with $r(f) = n$ and for each t_i $(i = 1, \ldots, n)$ holds $t_i \in \mathcal{T}(\Sigma, \mathcal{V})$,
3. nothing else is in the set $\mathcal{T}(\Sigma, \mathcal{V})$.

Symbols $a \in \Sigma$ with rank (or *arity*) $r(a) = 0$ are also called *constants*, the pair of alphabet Σ and ranking function r is often denoted as *signature*, the rank of a symbol is often marked by subscript. A term in $\mathcal{T}(\Sigma, \emptyset)$ (or shorthand: $\mathcal{T}(\Sigma)$) is called a *ground term*. If Σ and $\mathcal{V}$ are fixed by context of usage we will abbreviate $\mathcal{T}(\Sigma, \mathcal{V})$ by $\mathcal{T}$. Let s,t be two terms in $\mathcal{T}(\Sigma, \mathcal{V})$. s is called *subterm* (written $s \sqsubseteq t$) of t iff $s = t$ or $t = f(t_1, t_2, \ldots, t_n)$ and s is a subterm of one of the terms t_i $(i = 1, 2, \ldots, n)$. We write $s \sqsubset t$ if s is a *proper* subterm of t, i.e. $s \sqsubset t$ iff $s \sqsubseteq t$ and $s \neq t$. If $t = f(t_1, t_2, \ldots, t_n)$ then the term t_i is called the i-th *immediate subterm*.

Any term t in $\mathcal{T}(\Sigma, \mathcal{V})$ may be viewed as a *rooted labeled ordered tree*, the external nodes (or *leaves*) of which are labeled with variables and constants and the internal nodes of which are labeled with function symbols f $(f \in \Sigma$ with $r(f) \geq 1)$ and having an outdegree equal to the arity of the label. The *frontier* of a tree (term) is defined as the sequence of external nodes (leaves) built up in a left-to-right order.

A *position* within a term may be represented as a sequence of positive integers, describing the path (queued argument position numbers) from the *root* (*head*, outermost symbol) to the root (head) of a subtree (subterm). By $t|_p$ we denote the *subterm* of t rooted at position p. A tree (term) s is said to *occur* in t iff there is a position p in t with $t|_p = s$.

A *substitution* is a mapping $\gamma : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$, extended to $\gamma : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathcal{T}(\Sigma, \mathcal{V})$ in such a way that $f(t_1, t_2, \ldots, t_n)\gamma = f(t_1\gamma, t_2\gamma, \ldots, t_n\gamma)$. A term s *matches* a term t if there is a substitution γ with $t\gamma = s$. The functions *root* and *succ* have their usual meaning. Further let $\lambda(v) \in \Sigma$ map the node v of a tree (term) to its label (head symbol), and let $k = \max_{a \in \Sigma}(r(a))$ be the maximum rank of Σ. A subset of $\mathcal{T}(\Sigma, \emptyset)$ will be synonymously called *tree language*, *term language* or *forest*.

The computational power of neural network models is often constrained by certain types of transfer functions (e.g. see Siegelmann and Sontag [66]). Here, the following types of functions are of particular interest.

binary threshold	$\sigma_t : \mathbb{R} \mapsto \{0,1\}$	$\sigma_t(x) = \begin{cases} 1 & \text{if } x \geq \theta, \\ 0 & \text{else.} \end{cases}$	$\theta \in \mathbb{R}$
classical sigmoid	$\sigma_c : \mathbb{R} \mapsto]0,1[$	$\sigma_c(x) = \dfrac{1}{1+e^{-\mu x}}$	$\mu > 0 \in \mathbb{R}$
general sigmoid	$\sigma_g : \mathbb{R} \mapsto]\mu^-, \mu^+[$	$\lim_{x \to +\infty} \sigma_g(x) = \mu^+$ $\lim_{x \to -\infty} \sigma_g(x) = \mu^-$ and σ_g is continuous.	$\mu^- < \mu^+ \in \mathbb{R}$

Note that σ_c is a special case of σ_g and σ_t is approximated arbitrarily well by σ_c for $\lim_{\mu \to +\infty}$. In our constructions σ_c will be configured by choosing $\mu = 1$.

Sometimes it is convenient to index certain elements in an enumerable set. Then, we assume an arbitrary but a priori fixed total ordering on elements in a set M and use the function $\iota : M \to \mathbb{N}$ to compute the index number, i.e. $\iota(a) = i$ for $a = a_i$ and $M = \{a_1, a_2, \ldots, a_i, a_{i+1}, \ldots\}$.

3 Tree Automata

The concept of *tree automata* (and that of the corresponding tree grammars) is well-understood and applied in several fields of computer science like compiler construction (code generation and optimization, see Wilhelm and Maurer [81]), syntactical pattern recognition (e.g. Schalkoff [59] or Miclet [50]), linguistics, computational biology, computational logic and game theory. It may be viewed as a straight forward generalization of the well-known finite-state automata (FSA), see Hopcroft and Ullman [34]. Informally spoken, a state transition is decided upon a character read from the input and a tuple of prior states (instead of exactly one prior state in the case of finite state automata). There are several types of tree automata and several ways to introduce them, e.g. consult Doner [12], Gésceg and Steinby [20, 21] or Common et al. [10]. We are particularly interested in finite bottom-up tree automata and essentially follow the formalization of Schalkoff [59] or Miclet [50].

Definition 2 (Bottom-Up Tree Automata) A *finite bottom-up tree automata* (or *frontier-to-root automata*) is a quadruplet

$$(\Sigma, Q, F, R)$$

where Σ is a ranked alphabet, r the corresponding ranking function $r : \Sigma \to 2^{\mathbb{N}_0}$, Q a finite set of states, $F \subseteq Q$ a set of final states and $R = \{\delta_a : a \in \Sigma\}$ a set of relations of the type $\delta_a \subseteq Q^{r(a)} \times Q$.

If R is a set of functions of the type $\delta_a : Q^{r(a)} \to Q$ we speak of a *deterministic* finite bottom-up tree automata (DTA).

A given DTA $\mathcal{A}$ operates as a *tree acceptor* in the following way. First, each leaf node v with label $a = \lambda(v)$ of a given input tree t is assigned (if possible) an initial state belonging to $\delta_a \in Q$. Then $\mathcal{A}$ examines all the paths of t towards the root by looking at each node and its immediate successors. If the sequence of states already assigned to the successors of a node v with label a is

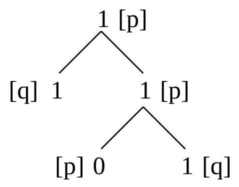

Figure 1: An example input tree processed by the DTA $\mathcal{A}_p$. Assigned states are denoted in square brackets.

an element of $Q^{r(a)}$ in the domain of the mapping δ_a, then the corresponding image state for the mapping is assigned to v. The process continues either it becomes impossible to assign states to nodes or the root is reached. The tree t is *accepted* by the automaton $\mathcal{A}$ iff the root of t is assigned to a final state.

As observed the transition functions of a DTA do not necessarily have to be defined on the whole domain. In order to be able to present a formal specification of the recognition operation we introduce a *undefined* state $\perp \notin F$ and define $\delta_a(q_1, q_2, \ldots, q_n) = \perp$ if there is a q_i $(i = 1, 2, \ldots, n)$ with $q_i = \perp$.

Let $t \in \mathcal{T}(\Sigma)$ be a given tree with $a = \lambda(root(t))$ and $t_1, \ldots, t_{r(a)}$ be the immediate subtrees. The transition functions in R can be extended to a function $\tilde{\delta} : \mathcal{T}(\Sigma) \to Q$ in the following way:

$$\tilde{\delta}(t) = \delta_a(\tilde{\delta}(t_1), \tilde{\delta}(t_2), \ldots, \tilde{\delta}(t_{r(a)})) \quad (1)$$

The *tree language $L(\mathcal{A})$ recognized* by a given DTA $\mathcal{A}$ is the set of all trees accepted by this automaton, i.e. $L(\mathcal{A}) = \{t \mid \tilde{\delta}(t) \in F\}$. The following DTA does not only serve as explanation but will also be used as a counterexample in Section 6.

Example 1 (Even-parity Tree Recognizer) Let $\mathcal{A}_p = (\Sigma, Q, F, R)$ be a DTA where $\Sigma = \{0, 1\}$, $r(0) = r(1) = \{0, 2\}$, $Q = \{q, p\}$, $F = \{p\}$ and $R = \{\delta_0, \delta_1\}$ with

$$\begin{aligned} \delta_0 : &\quad \mapsto p, \quad (p,p) \mapsto p, \quad (p,q) \mapsto q, \quad (q,p) \mapsto q, \quad (q,q) \mapsto p \\ \delta_1 : &\quad \mapsto q, \quad (p,p) \mapsto q, \quad (p,q) \mapsto p, \quad (q,p) \mapsto p, \quad (q,q) \mapsto q \end{aligned}$$

Figure 1 shows the automaton $\mathcal{A}_p$ accepting the the input tree $t = 1(1, 1(0, 1))$, assigned states are denoted in square brackets. The accepted language is the enumerable set of all binary trees with the number of nodes labeled by '1' is even. p acts as "even", q as "odd" state, a node of a tree is assigned a state depending on the parity of its immediate subtrees.

Henceforth for reasons of simplicity we consider DTA that are "filled up" to the maximum rank of the given signature.

Definition 3 (Maximum-Rank DTA) Let $\mathcal{A}' = (\Sigma', Q', F', R')$ be an arbitrary DTA. $\mathcal{A}'$ is transformed to a *maximum-rank automaton* $\mathcal{A} = (\Sigma, Q, F, R)$ in the following canonical way.

We introduce a void state $q_0 \notin Q'$. $\Sigma = \Sigma'$, $Q = Q' \cup \{q_0\}$, $F = F'$ and R is R' extended to the maximum rank k of Σ', i.e. each $\delta'_a \in R', \delta'_a : Q'^n \to Q'$ with $n < k$ is extended to $\delta_a : Q^k \to Q$ and each $(q_1, \ldots, q_n, q) \in \delta'_a$, $\delta'_a \in R'$ becomes $(q_1, \ldots, q_n, \underbrace{q_0, \ldots, q_0}_{k-n}, q) \in \delta_a$, $\delta_a \in R$.

Remarks The two classes of deterministic and nondeterministic bottom-up tree automata are known to have the same recognition power (see Doner [12]), i.e. the recognizable class of languages is identical. A tree automaton which processes the trees starting at the root proceeding then towards the leaves is called a *top-down* (or *root-to-frontier*) tree automaton. The nondeterministic version of the top-down tree automaton has the same power as the bottom-up counterpart, but the deterministic one is considerably weaker and defines a proper subfamily of recognizable languages (Doner [12], Gésceg and Steinby [20]).

As in the case of other automata types each type of tree automaton corresponds to certain kinds of *tree grammars* which can be alternatively used to describe the recognition power. The so-called *regular tree grammars* belong to DTA, nondeterministic bottom-up and nondeterministic top-down tree automata. The recognized language class is often called *regular tree language* (RTL).

By restricting the maximum rank to $k = 1$ tree automata are reduced to the well-known *finite state automata* (FSA), see Hopcroft and Ullman [34].

Several important concepts and results in the field of tree automata necessary to explore the correspondence to neural networks are still not addressed, but we will introduce them on demand and in context in the later sections.

4 Folding Architecture

The neural folding architecture (FA) together with a supervised learning procedure has been designed to inductively infer mappings from structured objects to the real vector space by given examples (Küchler and Goller [46]). In this section we will first explain the type of learning tasks to be solved by the FA in a more precise way. After the presentation of the layout of the generic architecture and the processing dynamics, the approximation and the generalization capabilities of the FA are briefly discussed.

4.1 The Learning Tasks

Precisely, the *inductive learning task* (ILT) intended to be solved by the FA is defined as a tuple $(\Xi, \mathcal{P})$, where Ξ is an unknown function of the type $\Xi : \mathcal{T}(\Sigma) \rightarrow I\!R^q$ with $q \in I\!N$ and $\mathcal{T}(\Sigma)$ is the (possibly infinite) domain of rooted labeled ordered trees (ground terms) over a given ranked alphabet Σ (see Section 2). $\mathcal{P}$ is a large but finite set of examples partially describing Ξ, i.e. $\mathcal{P} = \{(t_1, \Xi(t_1)), (t_2, \Xi(t_2)), \ldots, (t_p, \Xi(t_p))\}$ where $t_i \in \mathcal{T}(\Sigma)$ and $i, p \in I\!N, 1 \leq i \leq p$. The learning task is to infer an *approximation* Ξ_A (as good as possible, w.r.t. a given error measure) to the unknown function Ξ based on the given finite example set $\mathcal{P}$ only.

A special case is the function Ξ restricted to $\Xi : \mathcal{T}(\Sigma) \rightarrow \{c_1, c_2, \ldots, c_p\}$ with $p \in I\!N$. This is a description of inductive p-class *classification* tasks where the c_i $(i \in I\!N, 1 \leq i \leq p)$ are the corresponding class labels. The objective here is to infer a classifier Ξ_C with a high accuracy, i.e. with a high probability of correctly classifying a randomly selected instance $(t, \Xi(t)), t \in \mathcal{T}(\Sigma)$.

The principled idea in designing a neural network architecture for solving a given ILT is to combine a component for encoding elements of $\mathcal{T}(\Sigma)$ into suitable connectionist *distributed representations* (points in the real vector space) with a component to compute (approximation or classification) tasks on these distributed representations.

4.2 The Static View

Our generic *folding* architecture is layered and the static view is that of a specially scaled multilayer feedforward network (see Figure 2). The first r layers constitute the *folding* part, the next layers (including layer numbered r) the *transformation* part, where $s \geq 0$ and $r \geq 1$.

The number $q \in I\!N$ of *units* (neurons) in the output layer as well as the maximum rank k is defined by the given ILT $= (\Xi, \mathcal{P})$ (i.e. the maximum rank of the domain of Ξ). The number of *hidden* layers and the number of neurons in each layer concerning the folding and transformation part is not predefined. Neither is m, the dimension of the *representation layer* (the layer numbered by r). The input layer is constituted by $n + km$ units, i.e. one block of n units followed by k blocks of equal size m.

The architecture is homogeneous, i.e. all neurons are of the same type. The output of a layer $l + 1$ is computed from the output $\vec{o}^{\,(l)}$ of the previous one according to

$$\vec{o}^{\,(l+1)} = \sigma\left(\nu(\vec{o}^{\,(l)})\right)$$

where ν computes the activation of a neuron and σ is the transfer function. A popular instance is the classical sigmoid function $\sigma = \sigma_c$ (see Section 2) and $\nu = \nu_1$ with

$$\sigma_c(x) = \frac{1}{1 + e^{-x}} \qquad \nu_1(\vec{o}^{\,(l)}) = W^{(l+1)}\vec{o}^{\,(l)} + \vec{\theta}^{\,(l+1)}$$

ν_1 specifies so-called *first-order connections*, $W^{(l+1)}$ denotes the connections from layer l to $l + 1$ in matrix form (with real-valued components), $\theta^{(l)}$ the individual *biases* in layer l.

An other possibility is to use *higher-order connections* (also called *sigma-pi units*). For example consider an instance of the FA with a one-layer folding part ($r = 1$), no transformation part ($s = 0$), some units in the representation layer recruited as output units and the function $\nu = \nu_h$ of order $k + 1$ which computes the activation of neuron i as

$$\nu_h(\vec{x}) = \theta_i + \sum_{j=1}^{n} \sum_{l_1,\ldots,l_k=1}^{m} w_{ijl_1\ldots l_k} \cdot x_j \cdot \prod_{u=0}^{k-1} x_{n+um+l_u}$$

where $\vec{x} \in I\!R^{n+km}$ denotes the input vector, θ_i the bias of the neuron, x_i the i-th component of $\vec{x}$ and $w_{ijl_1\ldots l_k} \in I\!R$ is the weight of the connection (of order $k + 1$) that leads to neuron i by a multiplicative combination of the input from neuron j and from the neurons indexed $l_1, \ldots, l_k$. In this special higher-order case a fully-connected layer yields nm^{k+1} weight connections.

Remarks The FA may be viewed as a generalization of some types of *discrete-time continuous-space recurrent neural networks* (RNN) used for temporal sequence processing (e.g. see Giles et al. [28]). By setting $k = 1$, specifying one layer as folding part ($r = 1$), one layer as transformation part ($s = 1$) and choosing first-order connections the FA is instantiated to the *simple recurrent network* of Elman [13]. The configuration $k = 1, r = 1, s = 0, \nu = \nu_h$ gives the well-known *second-order recurrent* architecture.

Occasionally we will describe the FA by a 8-tuple [1] $(r, s, m, n, k, q, \nu, \sigma)$ although this notation does not specify the number of units in the hidden layers of the folding and transformation part.

[1]If one component is not relevant for certain propositions it will be filled by the anonymous underscore '_' character.

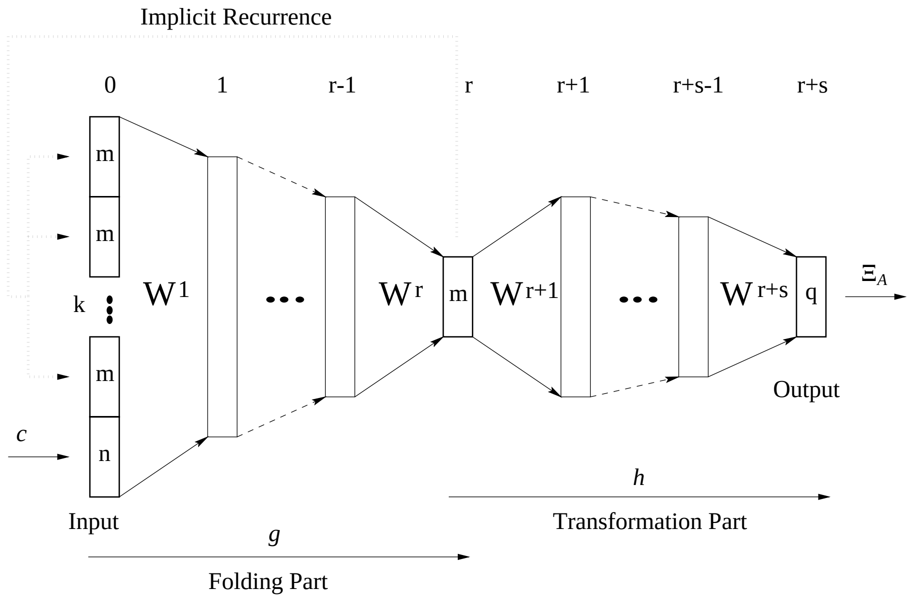

Figure 2: The generic folding architecture

4.3 Processing Dynamics

There are no explicit feedback connections in the FA. The recurrent processing is completely driven by the inherent recursive structure of the objects mapped from the tree (term) domain.

Let Σ be a ranked alphabet with maximum rank k, $\mathcal{T}(\Sigma)$ be the induced set of trees, c be an arbitrary, but fixed encoding function $c : \Sigma \rightarrow I\!R^n$ that maps labels to numeric codes and $nil \in I\!R^m$ be a special encoding for the empty tree.

Informally spoken the folding part is used to "fold" (or encode) a given tree into a distributed representation (in $I\!R^m$). This is achieved by recursively setting the previously computed representations of the immediate subtrees together with the label coding of the root node at the input layer and propagating them through the folding part. This process starts at the leaves (by using the label coding and k times the coding of the empty tree) and ends up with a representation of the whole tree which is then mapped by the transformation part yielding the mapping of the input tree.

Let us now define the dynamics of the folding architecture in a formal way. The folding part may be viewed as a function $g : I\!R^{n+k \cdot m} \rightarrow I\!R^m$ and the transformation part as a function $h : I\!R^m \rightarrow I\!R^q$. Let $t \in \mathcal{T}(\Sigma)$ be an arbitrary tree with $0 \leq d \leq k$ immediate subtrees $\{t_1, t_2, \ldots, t_d\}$. Then the mapping $\tilde{g} : \mathcal{T}(\Sigma) \rightarrow I\!R^m$ is recursively defined by g in the following way:

$$\tilde{g}(t) = g\big(c(\lambda(root(t))) \oplus \tilde{g}(t_1) \oplus \tilde{g}(t_2) \oplus \cdots \oplus \tilde{g}(t_d) \oplus \underbrace{nil \oplus \cdots \oplus nil}_{k-d}\big), \tag{2}$$

where '$\oplus$' is the vector concatenation operator. Finally the mapping $\Xi_A : \mathcal{T}(\Sigma) \rightarrow I\!R^q$ computed by the folding architecture is defined by

$$\Xi_A = h \circ \tilde{g}.$$

Precisely, this definition only allows mappings to the range of the transfer function. If one wants to use $I\!R^q$ as potential range the last layer of the transformation part can be replaced by a layer of neurons (or to be extended by an additional one) carrying identity transfer functions. By this measure, the last layer computes an affine mapping A and we get

$$h = A \circ h'$$

where h' is the function computed by the original part of the transformation layers.

Remarks If no transformation part is specified for the FA ($s = 0$) one can simply recruit some units from the representation layer (r) and define h to extract the corresponding components from the result vector delivered by the mapping $\tilde{g}$.

Let the network *state* be defined as the output of the representation layer (i.e. the layer numbered by r). Note that in our model state transitions depend on the current node of the input tree to be processed, i.e. autonomous state transitions are not allowed [2]. This modus operandi (described by Equation 2) essentially differs from that of other recurrent models (e.g. see some models investigated by Siegelmann and Sontag [66]), where an arbitrary number of autonomous state transitions are allowed (after the input sequence is completely fed into the network) before the output of the network is computed.

A supervised learning procedure can be developed to solve a given ILT $= (\Xi, \mathcal{P})$ with the FA in a common way. Let E be an (appropriately differentiable) error measure in the weight space w.r.t the training data $\mathcal{P}$ and the mapping Ξ_A. Obviously, the FA gives a good approximation to Ξ w.r.t. $\mathcal{P}$ if the error E is minimized. In principle any suitable numerical optimization procedure can be used for this minimization task. A simple gradient-based training algorithm called *Backpropagation Through Structure* (BPTS) has been developed and published by Goller and Küchler [26], later refined by Küchler and Goller [46] and worked out in detail by Goller [25, chapter 3].

In this article we concentrate on the computational capabilities of the folding architecture and for that purpose it suffices to know about the existence of an effective learning method. For the detailed mathematical derivation, the application and experimental evaluation of BPTS for the FA we refer to Goller [25], Küchler and Goller [46, 26], Schulz et al. [62], Schmitt [60], Calmbach [8].

Other froms of dynamics than expressed by Equation 2 might be realizable and compatible with the static view of the folding architecture. For example, by allowing to compute an output each time a node of a given input tree is processed the overall mapping constituted by the FA will be a special *structure-to-structure* mapping $\Xi_A : \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma)$ where the *shape* of tree remains invariant (see also Frasconi et al. [18]).

Further note that for the processing of trees by the FA it makes no difference whether the tree labels are taken from a discrete set Σ (and encoded by $c : \Sigma \to \mathbb{R}^n$) or directly drawn from $\mathbb{R}^n$.

4.4 Approximation Capabilities

If we want to apply the FA as approximation device one might ask about the class of functions that can be approximated under the assumption that an effective and appropriate learning method will be available. Recently, Hammer [29] has proven the FA being a *universal approximator* for functions of the above type $\Xi : \mathcal{T}(\Sigma) \to \mathbb{R}^q$. Here, we briefly recall this result.

Let the transfer function σ be the general sigmoid σ_g (see Section 2), monotonuous and further constrained by the existence of a point x_1 in $]\mu^-, \mu^+[$ so that σ is two times continuously differentiable in an environment of x_1 and $\sigma''(x_1) \neq 0$. Let $f : \mathcal{T}(\Sigma) \to \mathbb{R}^q$ be an arbitrary function and

[2] This property of models of computation is also called *real-time processing*.

P be an appropriate probability measure on $\mathcal{T}(\Sigma)$. Further let $\mathcal{F}$ denote the class of functions that are implementable by the generic FA equipped with neurons using σ as transfer function.

Theorem 1 (Approximation) *For any $\varepsilon > 0$ there exists a function $f^\varepsilon \in \mathcal{F}$ so that for all $t \in \mathcal{T}(\Sigma)$ holds*

$$P\left(|f^\varepsilon(t) - f(t)| > \varepsilon\right) < \varepsilon$$

Hammer [29] gives a proof construction that requires $O(\log_2 k)$ layers with less than $O(k)$ units in each layer in the folding part and a three-layer feedforward network as transformation part. k is the maximum rank of Σ, the dimension m of the representation layer can be chosen less than 3. The theorem also holds for radial basis functions as transfer function. Using the threshold function σ_t, approximation can be accomplished with a number of units that is exponential to the maximum depth of the trees in a finite domain.

4.5 VC-Dimension

One aspect in characterizing the generalization capabilities of learning devices like neural networks is the theoretical investigation of the *sample complexity*, i.e. the amount of training data necessary to achieve a given level of generalization accuracy. The sample complexity is known to be closely related to the so-called *Vapnik-Chervonenkis-dimension* (or *VC-dimension*), which can be regarded as a measure of the "expressive power" of a set of functions $\mathcal{F}$. In the case of neural networks, $\mathcal{F}$ is defined as the set of functions implementable by a given architecture. For the exact relationship between sample complexity and VC-dimension, the application of these concepts to neural networks and a survey of recent work on that field the reader may be referred to Anthony [4].

In the following we show that an upper bound for the VC-dimension of the FA can be trivially derived by directly applying the result of Karpinski and Macintyre [38] (VC-dimension of multilayer feedforward networks) and the extension of Koiran and Sontag [42] (VC-dimension of RNN).

Theorem 2 (VC-Dimension) *For folding architectures with first-order connections being constrained (by discretized interpretation of the output) to implement functions $\Xi \in \mathcal{F}$ of the type $\Xi : \mathcal{T}(\Sigma) \to \{0, 1\}$ holds:*

The VC-dimension given classical sigmoid transfer functions σ_c is $O(u^2 w^4)$,

where w is the number of weights (including thresholds) in the FA and u is the size (number of nodes) of the input terms (trees) received by the FA.

Proof (VC-Dimension, Upper Bound) By *virtually unfolding* (see also Section 4.3, Equation 2) the FA for a given input tree of size u the recurrent architecture can be simulated by an equivalent multilayer feedforward network with $un_f + n_t$ units (neurons), where n_f resp. n_t be the number of units in the folding resp. transformation part of the original architecture, and the same number of *programmable parameters* (weights) $w = w_f + w_t$ in the folding (w_f) resp. transformation part (w_t).

By Karpinski and Macintyre [38, Theorem 7] there is an $O((un_f + n_t)^2 w^2)$ upper bound on the VC-dimension of that feedforward architecture. Assuming $w > w_f > n_f$ and $w > w_t > n_t$ this is $O(u^2 w^4)$ as claimed. ■

Remarks Koiran and Sontag [42] also derived upper bounds on the VC-dimension of RNN instantiated by several other classes of transfer functions. These results seem to be transferrable to the FA in a similar way as demonstrated by the proof of Theorem 2. Hammer [30] explores recurrent neural network models using the framework of PAC learning and give lower bounds on the VC-dimension of the FA.

5 Single Neurons and Boolean Functions

The smallest building block of the FA is the single unit (neuron). In Section 6 we will utilize the fact that the transition function of a DTA can be rewritten in a propositional way. Thus, it is convenient to explore the computational capabilities of a single neuron first and to investigate how certain Boolean functions can be simulated by certain types of single neurons. The results to be developed here will enable us to establish constructive proofs for propositions concerning the correspondence between the FA and DTA (to be presented in later sections).

5.1 First-Order Connections

Lemma 1 *A single neuron provided with first-order connections and the classical sigmoid transfer function σ_c can simulate the following Boolean functions:*

a. *The n-ary Boolean $\wedge$,*

b. *the n-ary Boolean $\vee$,*

c. *the function $\alpha \wedge (\beta^1_1 \vee \ldots \vee \beta^1_{n_1}) \wedge \ldots \wedge (\beta^k_1 \vee \ldots \vee \beta^k_{n_k})$, where $\alpha, \beta^i_j \in \{0,1\}$, $1 \leq i \leq k$, $1 \leq j \leq n_i$ and for all i there exists at most one j with $\beta^i_j = 1$.*

In order to simulate Boolean functions by neurons equipped with transfer functions of a continuous range a special discretization scheme has to be applied. The following idea is borrowed from Ivanova and Kubat [37].

Proof Since σ_c has a continuous range of $]0,1[$ we choose an $\varepsilon \in]0, 0.5[$ and interpret the output of a neuron in the following way. We say that (the output of) a neuron is *high* if $\sigma(x) \geq 1-\varepsilon$, *low* if $\sigma(x) \leq \varepsilon$ and undefined, else (x being the activation). Thus, the Boolean values $\{0,1\}$ correspond to $\{low, high\}$. Let $\psi = \ln \frac{1-\varepsilon}{\varepsilon}$. Due to the monotonicity and symmetry of σ_c the output will be high if the activation of the neuron is $x \geq \psi$, low if $x \leq -\psi$ and undefined, else.

a. A neuron receiving input from n neurons connected by identical weights $w \in I\!R, w > 0$ behaves as a n-ary Boolean $\wedge$ iff the following constraints can be satisfied:

$$(1-\varepsilon) \cdot n \cdot w + \theta \quad \geq \quad \psi \qquad (3)$$

$$1 \cdot (n-1) \cdot w + \varepsilon \cdot 1 \cdot w + \theta \quad \leq \quad -\psi \qquad (4)$$

Constraint 3 states that the neuron should be high even if all input neurons are (minimal) high. Further it should be low even if $n-1$ input neurons are (maximal) high and one is (minimal) low (Constraint 4). By turning these constraints into equations we get at least one solution:

$$w = \psi \cdot \frac{2}{1-\varepsilon(n+1)} \qquad \theta = \psi \cdot \frac{1-\varepsilon(n+1) - 2n(1-\varepsilon)}{1-\varepsilon(n+1)} \qquad \varepsilon < \frac{1}{n+1}$$

b. For simulating a n-ary Boolean $\vee$ we get the following conditions ($w > 0$):

$$\begin{aligned} (1-\varepsilon)\cdot 1\cdot w + 0\cdot(n-1)\cdot w + \theta &\geq \psi && (5)\\ \varepsilon\cdot n\cdot w + \theta &\leq -\psi && (6)\end{aligned}$$

Constraint 5 postulates that the neuron should give a high output if one input neuron is (minimal) high and all other input neurons are (maximal) low. The neuron has to be low, even if all input neurons are minimal low (Constraint 6). We get at least one solution:

$$w = \psi\cdot\frac{2}{1-\varepsilon(n+1)} \qquad \theta = -\psi\cdot\frac{1-\varepsilon(n+1)+2n\varepsilon}{1-\varepsilon(n+1)} \qquad \varepsilon < \frac{1}{n+1}$$

c. We assume that the neuron receives input from one neuron by the weight connection $w_\alpha > 0$ and from $n = \sum_{i=1}^{k} n_i$ neurons by identical weight connections $w_\beta > 0$. The Boolean function $\alpha \wedge (\beta_1^1 \vee \ldots \vee \beta_{n_1}^1) \wedge \ldots \wedge (\beta_1^k \vee \ldots \vee \beta_{n_k}^k)$ is simulated if the following constraints hold:

$$\begin{aligned} (1-\varepsilon)\cdot 1\cdot w_\alpha + k\cdot(1-\varepsilon)\cdot 1\cdot w_\beta + \theta &\geq \psi && (7)\\ \varepsilon\cdot 1\cdot w_\alpha + [1\cdot 1 + \varepsilon(n_1-1)]\cdot w_\beta + \ldots & && \\ \ldots + [1\cdot 1 + \varepsilon(n_k-1)]\cdot w_\beta + \theta &\leq -\psi && (8)\\ 1\cdot 1\cdot w_\alpha + \varepsilon\cdot n_1\cdot w_\beta + [1\cdot 1 + \varepsilon(n_2-1)]\cdot w_\beta + \ldots & && \\ \ldots + [1\cdot 1 + \varepsilon(n_k-1)]\cdot w_\beta + \theta &\leq -\psi && \\ &\vdots && \\ 1\cdot 1\cdot w_\alpha + [1\cdot 1 + \varepsilon(n_1-1)]\cdot w_\beta + \ldots & && \\ \ldots + [1\cdot 1 + \varepsilon(n_{k-1}-1)]\cdot w_\beta + \varepsilon\cdot n_k\cdot w_\beta + \theta &\leq -\psi && (9)\end{aligned}$$

Constraint 7 postulates that the neuron should be high if α is set (minimal) high and for all $1 \leq i \leq k$ there is exactly one β_j^i, $1 \leq j \leq n_i$ (minimal) high and all other $n_i - 1$ (maximal) low. Further the neuron has to be low if α is set (minimal) low and each β-disjunction is tuned to a maximal permissable value (Condition 8). k inequalities are expressed by Condition 9. The neuron should be low if α is set (maximal) high, $k-1$ β-disjunctions are tuned to a maximal value and for one β^i-disjunction all n_i literals are (minimal) low. The above $k+2$ constraints may be compactified to:

$$\begin{aligned} (1-\varepsilon)\cdot w_\alpha + k\cdot(1-\varepsilon)\cdot w_\beta + \theta &\geq \psi && (10)\\ \varepsilon\cdot w_\alpha + [k + \varepsilon\cdot n - k\cdot\varepsilon]\cdot w_\beta + \theta &\leq -\psi && (11)\\ w_\alpha + [k - 1 + \varepsilon\cdot n - (k-1)\cdot\varepsilon]\cdot w_\beta + \theta &\leq -\psi && (12)\end{aligned}$$

By turning these conditions to equations and computing (11)-(12) we get

$$w_\alpha = w_\beta \qquad (13)$$

Resubstitution into Condition 10 and Condition 11 yields

$$\begin{aligned} [(k+1)\cdot(1-\varepsilon)]\cdot w_\alpha + \theta &= \psi && (14)\\ [\varepsilon + k + \varepsilon\cdot n - k\cdot\varepsilon]\cdot w_\alpha + \theta &= -\psi && (15)\end{aligned}$$

and leads to at least one solution:

$$w_\alpha = w_\beta = \psi \cdot \frac{2}{1-\varepsilon(n+2)} \qquad \theta = -\psi \cdot \frac{2k\cdot(1-\varepsilon)+1+\varepsilon\cdot n}{1-\varepsilon(n+2)}$$

$$\varepsilon < \frac{1}{n+2} \qquad n = \sum_{i=1}^{k} n_i$$

■

One might have observed that this proof has been designed to obtain a tool that possesses more power than really needed to prove Lemma 1. Although the parameters for the Boolean functions being constrained to discrete values from the set $\{0,1\}$ by Lemma 1, the construction permits to apply the developed discretization scheme for both input and output values (presented to and computed by the single neuron). This gives rise to the following lemma.

Lemma 2 (Circuits) *Any logical circuit composed by the Boolean gates given in Lemma 1 (a.,b.,c.) can be simulated by a network of neurons with first-order connections and the transfer function σ_c.*

Proof (Circuits) Substitute each logical gate by a neuron. Let n' be the maximum fan-in found in the circuit. Choose the discretization variable ε as $\varepsilon = \frac{1}{n'+3}$. Set the thresholds and weights according to the recipe given in the proof construction for Lemma 1. Obviously, the resulting neural network simulates the given logical circuit. ■

Lemma 1 and Lemma 2 can be generalized in the following way.

Lemma 3 *Lemma 1 and Lemma 2 hold substituting σ_c by the general sigmoid function σ_g*

Proof Due to the asymptotic behavior of σ_g for any given ε with $0 < \varepsilon < \frac{1}{2}(\mu^+ - \mu^-)$ there exist ψ_+ and ψ_- such that $\sigma_g(x) \geq (\mu^+ - \varepsilon)$ if $x \geq \psi_+$ and $\sigma_g(x) \leq (\mu^- + \varepsilon)$ if $x \leq \psi_-$. By choosing $\psi = \max(|\psi_-|, |\psi_+|)$ we can apply the discrete interpretation scheme and the constructions used to prove Lemma 1. ■

Lemma 4 *Lemma 1 and Lemma 2 hold when σ_c is substituted by the threshold function σ_t.*

Proof The range of σ_t is $\{0,1\}$ so that the Boolean functions may directly be implemented without discretization. Let $\psi \in I\!R^+$ be an arbitrary constant. We apply the general construction given in the proof of Lemma 1 by setting $\varepsilon = 0$.

a. Obeying the constraints for the n-ary $\wedge$, $n \cdot w \geq \theta$ and $(n-1)\cdot w < \theta$ we get one possible solution:

$$w = 2\psi \qquad \theta = \psi(2n-1)$$

b. For the n-ary $\vee$ we have to satisfy $w \geq \theta$ and $0 < \theta$:

$$w = 2\psi \qquad \theta = \psi$$

c. The Boolean function $\alpha \wedge (\beta_1^1 \vee \ldots \vee \beta_{n_1}^1) \wedge \ldots \wedge (\beta_1^k \vee \ldots \vee \beta_{n_k}^k)$ requires $w_\alpha + k \cdot w_\beta \geq \theta$, $k \cdot w_\beta < \theta$ and $w_\alpha + (k-1) \cdot w_\beta < \theta$. We get one possible solution by

$$w_\alpha = w_\beta = 2\psi \qquad \theta = \psi(2k+1)$$

■

5.2 Higher-Order Connections

As specified in Section 4.2 the FA can be configured with higher-order activation functions. The next lemma shows that this enables the simulation of a more complex (than given in Lemma 1 c.) Boolean function by using just a single neuron.

Lemma 5 *A single neuron equipped with k-th-order connections, the classical sigmoid transfer function σ_c and the activation function $\nu_h(\vec{\beta}) = \sum_{i=1}^{l} w_i \prod_{j=1}^{k} \beta_j^i$ (where $l, k \geq 1$ and β_j^i are the inputs to the neuron) can simulate the Boolean function*

$$(\beta_1^1 \wedge \ldots \wedge \beta_k^1) \vee (\beta_1^2 \wedge \ldots \wedge \beta_k^2) \vee \ldots \vee (\beta_1^l \wedge \ldots \wedge \beta_k^l) \tag{16}$$

Proof We apply the same discretization scheme as used for proving Lemma 1. Since each conjunct in Equation 16 carries the same number k of literals the weights are assumed to be identical $w_i = w > 0$, $i = 1, \ldots, l$. A neuron simulates the above Boolean function if the following constraints can be satisfied:

$$\begin{aligned} (1-\varepsilon)^k \cdot w + 0^k \cdot w \cdot (l-1) + \theta &\geq \psi && (17) \\ \varepsilon \cdot 1^{k-1} \cdot w \cdot l + \theta &\leq -\psi && (18) \end{aligned}$$

Constraint 17 (Constraint 18) expresses the minimal (maximal) activation of the neuron that should result in a high (low) output. By turning these constraints into equations we get at least one solution:

$$w = \psi \cdot \frac{2}{(1-\varepsilon)^k - l\varepsilon} \qquad \theta = -\psi \cdot \frac{(1-\varepsilon)^k + l\varepsilon}{(1-\varepsilon)^k - l\varepsilon}$$

In order to fulfill $w > 0$ the discretization parameter $\varepsilon \in]0, 0.5[$ has to be chosen such that $(1-\varepsilon)^k > l\varepsilon$. Let $y(\varepsilon) = (1-\varepsilon)^k$ be substituted by $\hat{y}(\varepsilon) = 1 + \left[\frac{\partial y}{\partial \varepsilon}\right]_0 \varepsilon = 1 - k\varepsilon$, i.e. the tangent on y in $\varepsilon = 0$. Thus, $\varepsilon < \frac{1}{k+l}$ guarantees $w > 0$. ■

Remarks Lemma 5 also holds for the general transfer function σ_g and the threshold function σ_t. The Boolean function given by Equation 16 subsumes the k-ary $\wedge$ (by choosing $l = 1$) and the l-ary $\vee$ (by choosing $k = 1$). Furthermore, units with higher-order connections can be composed to larger Boolean circuits (using the same argumentation as applied to prove Lemma 2 for the case of first-order activation functions).

6 Correspondences between DTA and FA

Our intuition tells us that there must be a tight correspondence between DTA and FA. Both machines are deterministic, both process trees from the leaves to the root and both compute new states (the output of the representation layer of the FA may be interpreted as state) by combining a sequence of prior states with symbols read from the input (compare Equation 1 with Equation 2).

The relation between FSA and RNN has been intensively studied during the last years (e.g. Kremer [45], Omlin and Giles [55], Frasconi et al. [16], Goudreau et al. [28], Minsky [51]). One can observe that DTA/FA is a straight-forward generalization of the FSA/RNN concept. This section is to show how known results about the relationship between FSA/RNN can be transferred and applied to explore the correspondences between the DTA and FA.

The approximation theorem of Hammer [29] may give a first hint on the computational power of the FA and its relationship to DTA.

Theorem 3 *The FA provided with first-order connections and the classical sigmoid transfer function σ_c can approximate any DTA w.r.t. a finite set of input trees arbitrarily well.*

Proof A DTA may be viewed as an acceptor function $\Xi : \mathcal{T}(\Sigma) \rightarrow \{0,1\}$. According to [29] (see also Theorem 1, Section 4.4) any such function Ξ can be approximated arbitrarily well by the generic FA. The construction of the proof requires $\mathrm{O}(\log_2 k)$ layers with $\mathrm{O}(k)$ units each in the folding part and a multilayer network as transformation part with $q = 1$, where k is the maximum rank found in Σ. ■

This result supports the conjecture that the FA is at least as powerful as DTA. However, a DTA is defined on a enumerable set of trees (that cannot a priori be limited in size) and the Theorem 3 gives us little practical help for implementing a concrete DTA into a finite-size FA with the exactly identical behavior.

We are going to show that indeed the FA is at least as powerful as DTA. The investigation of the relationship between FA and DTA is guided by three principles:

Architectural Constraints The generic folding architecture developed in Section 4.2 allows several degrees of freedom. We want to explore how architectural constraints like the number of layers in the folding and transformation part, the type of the activation and the transfer function will take influence on the relationship to the DTA.

Constructiveness in Proof All proofs of positive propositions will be carried out in a constructive manner. The tools developed in Section 5 will be applied to give construction schemes for the implementation of given DTA by specific folding architectures.

Analysis of Complexity For practical purposes (for example see Section 9) one might also be interested in the complexity of the different alternative "neural DTA-implementations" to be developed. The space complexity of a DTA $\mathcal{A} = (\Sigma, Q, F, R)$ can be characterized by the number of states $m = |Q|$, the size of the alphabet $n = |\Sigma|$, the maximum rank k of Σ, the number of final states $n_F = |F|$ and the number of explicitly specified transitions $n_R = \sum_{\delta \in R} |\delta|$. If it is demanded that each possible combination of states and input symbols has to be explicitly specified we get $n_R^\star = m^k \cdot n$ different transitions.

The complexity of a specific neural implementation will be measured by the number of neurons (also called *node complexity* of the implementation, not counting the input layer) and the number of weights (inclusive biases) used in the construction. Here, it will be convenient to discriminate between the number of non-zero weights in the neural network architecture and the total number of weights if layers are assumed to be fully-connected.

6.1 Two-Layered Folding Part

We begin the investigations by proving that an architecture $\mathrm{FA}(2, 1, |Q|, n, k, 1, \nu_1, \sigma_g)$, i.e. two layers in the folding part, one unit as transformation part with the first-order activation function ν_1 and the general sigmoid transfer function σ_g, suffices to simulate any given DTA.

Theorem 4 *The FA with first-order connections and the general transfer function can simulate any DTA using two layers in the folding part and only one unit in the transformation part.*

Proof Let $\mathcal{A} = (\Sigma, Q, F, R)$ be an arbitrary DTA with $\Sigma = \{a_1, \ldots, a_n\}$, $t \in \mathcal{T}(\Sigma)$ be a given tree and v a node in t whose successors are already assigned the states $(q_1, \ldots, q_k)$. By Definition 2 we can rewrite the conditions for state transitions in a *propositional* way. $\mathcal{A}$ assigns v the state q iff

$$\bigvee_{\substack{(p_1,\ldots,p_k,q)\in\delta_a \\ a\in\Sigma}} \left((a = \lambda(v)) \bigwedge \left(\bigwedge_{i=1}^{k} (q_i = p_i) \right) \right) \tag{19}$$

$\mathcal{A}$ is mapped into an architecture $\mathrm{FA}(2, 1, |Q|, n, k, 1, \nu_1, \sigma_g)$ by the following construction (see also Figure 3). We choose the coding function c for labels (symbols) as $c(a_i) = e_i$, $a_i \in \Sigma$, i.e. the i-th symbol is mapped to the i-th unit vector (the i-th component is high, all others low in the sense of Lemma 1, proof construction). The second layer in the folding part consists of $m = |Q|$ units, the i-th unit corresponds to the i-th state $q_i \in Q$. For each $(p_1, \ldots, p_k, p) \in \delta_a, a \in \Sigma$ we put a unit into the first layer receiving connections from the $\iota(a)$-th unit of the label block and from each $\iota(p_j)$-th unit of the j-th block of the input layer ($j = 1, \ldots, k$). That unit is further connected to the $\iota(p)$-th unit in the second layer.

By Lemma 3 (*Circuit Lemma*) all units in the first layer can be tuned (weights and biases) to act as $(k+1)$-ary logical $\wedge$, each unit i in the second layer to act as logical $\vee$ with arity $n_i = |\{(p_1, \ldots, p_k, q_i) \in \delta, \delta \in R\}|$. By Equation 19 it is obvious that the folding part implements the state transition function of $\mathcal{A}$. Finally we add a single output unit in transformation part of the FA acting as $|F|$-ary logical $\vee$ receiving input from each unit $\iota(q), q \in F$ of the second layer. The empty tree (corresponding to state q_0) is coded by e_0.

In case where the transition function of $\mathcal{A}$ is only partially defined the missing connections (to a full connectivity between layers) will be inserted with weight $w = 0$. Thus, the *undefined* state $\perp \notin F$ (see Definition 2 and Equation 19) is represented by $\vec{0}$.

It is clear (compare Equation 2 and Equation 19) that the tree t is mapped by the FA to a high output iff the DTA $\mathcal{A}$ accepts t. ■

Table 1 summarizes the results of the space complexity analysis for the "neural implementation" of a given DTA into the folding architecture (as demonstrated by the proof construction for Theorem 4, see also Figure 3).

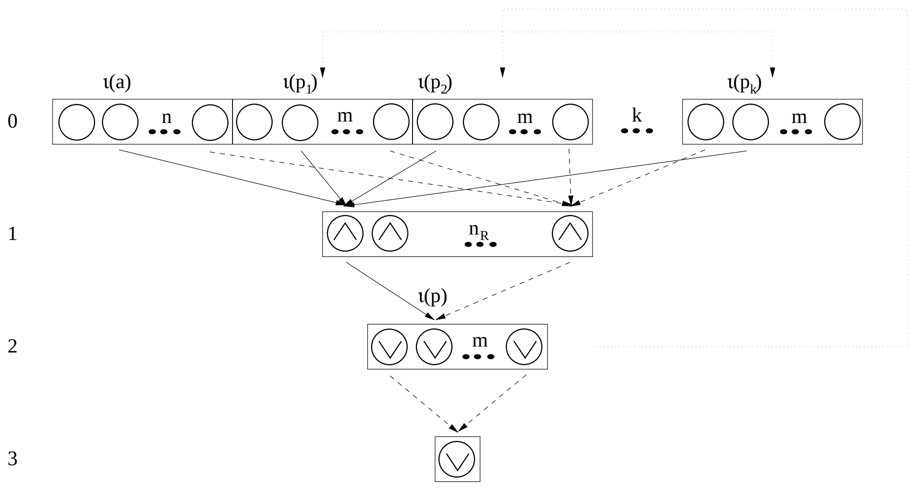

Figure 3: Layout of the folding architecture with two layers in the folding part, one layer in the transformation part for simulating DTA. Neurons are denoted by circles, those acting as Boolean functions are filled with $\wedge$ and $\vee$, groups of semantically similar neurons are enclosed by boxes. The solid lines with arrows exemplify the connections to be inserted when a transition $(p_1, p_2, \ldots, p_k, p) \in \delta_a, \delta_a \in R$ is embedded.

	n_R	$n_R^\star$
neurons	$n_R + m + 1 =$ $O(n_R)$	$m^k n + m + 1 =$ $O(m^k n)$
weights (full connectivity)	$(n + km)n_R + n_R m + m +$ $+ n_R + m + 1 = O(n_R(n + km))$	$(n + km)m^k n + m^k nm + m +$ $+ m^k n + m + 1 = O(km^{k+1} n + m^k n^2)$
weights (non-zero)	$n_R(k + 2) + n_F + n_R + m + 1 =$ $O(kn_R)$	$m^k n(k + 2) + n_F + m^k n + m + 1 =$ $O(km^k n)$

Table 1: Complexity of the DTA implementation given by Theorem 4, proof construction.

By a small modification of the proof construction given above (regarding Theorem 4) we can show that the transformation part of the FA is superfluous.

Eliminate the output unit of the transformation part and all connections leading to. Instead, extend layer two by one unit acting as logical $\vee$ and the new designated output of the network. Connect this unit with all units of the first layer that belong to a transition leading to a final state ($\{(p_1, \ldots, p_k, p) \in \delta \mid \delta \in R, p \in F\}$). It is obvious that this unit is high iff the DTA accepts the tree t. The complexity of this construction (see Table 2) differs only marginally from the proof of Theorem 4. Let n_f describe the number of transitions leading to a final state, i.e. $n_f = |\{(p_1, \ldots, p_k, p) \in \delta \mid \delta \in R, p \in F\}|$.

Corollary 1 *The FA with first-order connections and the general transfer function can simulate any DTA using two layers in the folding part. No transformation part is required.*

	n_R	$n_R^\star$
neurons	$n_R + m + 1 =$ $O(n_R)$	$m^k n + m + 1 =$ $O(m^k n)$
weights (full connectivity)	$(n + k(m+1))n_R + n_R(m+1) +$ $+ n_R + m + 1 = O(n_R(n + km))$	$(n + k(m+1))m^k n + m^k n(m+1) +$ $+ m^k n + m + 1 = O(km^{k+1}n + m^k n^2)$
weights (non-zero)	$n_R(k+2) + n_f + n_R + m + 1 =$ $O(kn_R)$	$m^k n(k+2) + n_f + m^k n + m + 1 =$ $O(km^k n)$

Table 2: Complexity of the DTA implementation given by Corollary 1.

6.2 One-Layered Folding Part

The next conclusive question is whether one layer in the folding part of the FA suffices to simulate any DTA. The answer seems to depend on the representability of arbitrary DTA state transition functions and on the type of activation function implanted into the FA.

6.2.1 Higher-Order Connections

The condition for state transitions in a DTA as formulated by Equation 19 (Theorem 4, proof) can be immediately identified as a Boolean function of the type presented by Equation 16 (Lemma 5). This observation can be utilized to simulate any given DTA by a FA with higher-order activation functions and only one layer as folding part.

Theorem 5 *The FA with higher-order connections and the general transfer function can simulate any given DTA using one layer in the folding part and no transformation part.*

Proof Let $\mathcal{A} = (\Sigma, Q, F, R)$ be an arbitrary DTA, let k be the maximum rank of Σ. We apply the same construction scheme as presented by the proof of Theorem 4 and Corollary 1, but now using neurons with higher-order (defined by Lemma 5) instead of first-order connections to implement Equation 16 into an architecture $\text{FA}(1, 0, |Q|+1, n, k, 1, \nu_h, \sigma_g)$ (see also Figure 4).

Symbols in Σ are encoded by unit vectors. The folding part consists of one layer carrying $|Q|+1$ neurons, one for each state in $|Q|$ and one neuron as output of the network. There is no transformation part.

For each state $q \in Q$ the disjunction of conjunctions (as expressed by Equation 19) is modeled by the $\iota(q)$-th neuron receiving connections of order $k+1$ from the corresponding units of the input layer. The conditions for transitions leading to final states are implemented by the designated output neuron in the same manner.

Let l' be the maximum fan-in found in the network. Thus, the Circuit Lemma (Lemma 3) can be applied to neurons with an activation function of order $k+1$ by choosing the discretization parameter ε as $\varepsilon = \frac{1}{l'+k+2}$ and setting the weights of the connections according to Lemma 5.

Following the same line of argumentation as worked out for the proof of Theorem 4 it is obvious that this FA construction yields a simulation of the given DTA $\mathcal{A}$.

■

Table 3 summarizes the resource consumption for the proof construction developed above. The "$\sim$" sign inside a column indicates that the complexity aspect does not change when all possible

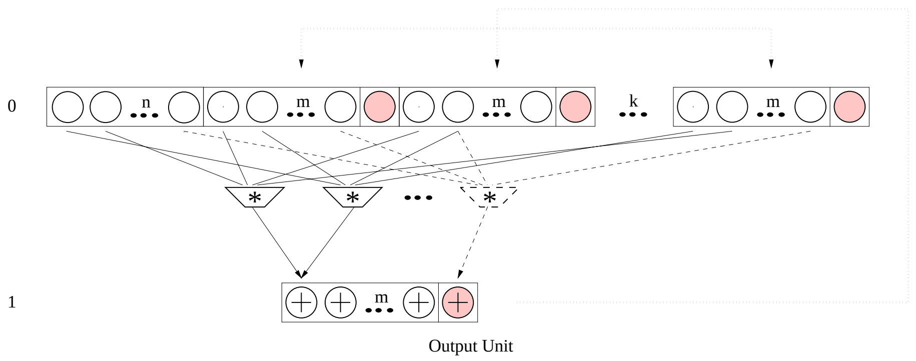

Figure 4: Sketch of the layout of the folding architecture constituted by one layer as folding part, no transformation part and by neurons with connections of order $k+1$ for simulating a given DTA. Neurons are denoted by circles, the connections of order $k+1$ by trapeze filled with asterisk "$*$" and linked to circles with a plus sign "$+$" inside, groups of semantically similar neurons are enclosed by boxes. The output neuron is marked in a hatched manner. The solid lines exemplify the implementation of the transition conditions leading to the state represented by the first neuron. The implementation of the test on the final state condition is indicated by dashed lines leading to the output unit.

transitions are explicitly specified for the DTA. Again, n_f denotes the number of automaton transitions leading to a final state.

	n_R	$n_R^\star$
neurons	$m+1 =$ $O(m)$	$\sim$
weights (full connectivity)	$n(m+1)^k(m+1)+m+1 =$ $O(n(m+1)^{(k+1)})$	$\sim$
weights (non-zero)	$n_R+n_f+m+1 =$ $O(n_R)$	$m^k n+n_f+m+1 =$ $O(m^k n)$

Table 3: Complexity of the DTA implementation given by the proof construction to Theorem 5.

Remarks When considering the space complexity the higher-order one-layered implementation of a DTA seems to be superior to the first-order two-layered case presented in Corollary 1. However, the time complexity for recognizing a given tree of size u with the FA gives a different picture. Folding architectures with first-order connections require $\Theta(un_w)$ multiplicative operations which is in contrast to $\Theta((k+1)u\ n_w)$ multiplicative operations for architectures with connections of order $k+1$, where n_w is the number of weights found in the architecture.

6.2.2 One Layer with First-Order Connections Cannot Implement Arbitrary State Transitions

We have seen that a FA with one layer in the folding part (and no transformation part) suffices to simulate an arbitrary DTA if higher-order connections are provided. The next question is whether one-layered FA with first-order connections have the same computational power. The first idea is to investigate the representability of automata state transition functions in a folding part consisting of only one layer.

Theorem 6 *One layer as folding part of a FA with first-order connections and the threshold transfer function σ_t cannot compute arbitrary DTA state transition functions.*

This statement is easily proven by applying known results about the relationship between FSA and RNN.

Proof Deterministic Finite State Automata are special DTA where each new state is computed by combining each symbol of the alphabet with exactly one previous state, i.e. FSA are DTA $\mathcal{A} = (\Sigma, Q, F, R)$ with the maximum rank of Σ is $k = 1$. Goudreau et al. [28] proved that a single layer with first-order connections and the threshold transfer function cannot compute arbitrary finite state transition functions. ■

It turns out that the situation does not change when using more powerful transfer functions.

Theorem 7 *Theorem 6 even holds for monotonous general sigmoid transfer functions.*

Proof Here we provide an explicit example of a DTA state transition function which cannot be computed by a single layer FA with first-order connections. Take the "even-parity" DTA $\mathcal{A}_p$ from Example 1.

Without loss of generality let $\vec{q}$ resp. $\vec{p}$ be outputs at layer 1 that represent the "even" resp. "odd" parity state of $\mathcal{A}_p$. We assume that p and q are distinguishable, i.e. $\vec{q} \neq \vec{p}$. Further let the input symbols be coded by $c('0') = \vec{x}_0$ and $c('1') = \vec{x}_1$ with $\vec{x}_0 \neq \vec{x}_1$. In order to compute the state transitions correctly the following constraints have to be satisfied (compare Example 1):

$$\begin{array}{cccc}
 & A & B & C \\
0 & g(\vec{x}_0,\vec{p},\vec{p},W) = \vec{p} & g(\vec{x}_0,\vec{p},\vec{q},W) = \vec{q} & g(\vec{x}_0,\vec{q},\vec{q},W) = \vec{p} \\
1 & g(\vec{x}_1,\vec{p},\vec{p},W) = \vec{q} & g(\vec{x}_1,\vec{p},\vec{q},W) = \vec{p} & g(\vec{x}_1,\vec{q},\vec{q},W) = \vec{q}
\end{array}$$

Thresholds $\vec{\theta}$ may be integrated into the weight matrix W by supplying additional input units with the constant input $\vec{1}$. We denote by W^1, W^c, W^l, W^r the columns of W that correspond to the threshold, label coding, left and right input block. The folding part (function g) computes a linear combination of inputs and passes the result through the monotonous sigmoid function. Thus we can strip the transfer function σ_g and get

$$\begin{aligned}
A0 - A1 &\overset{!}{=} B1 - B0 \\
W(\vec{1} \oplus \vec{x}_0 \oplus \vec{p} \oplus \vec{p}) - W(\vec{1} \oplus \vec{x}_1 \oplus \vec{p} \oplus \vec{p}) &\overset{!}{=} W(\vec{1} \oplus \vec{x}_1 \oplus \vec{p} \oplus \vec{q}) - W(\vec{1} \oplus \vec{x}_0 \oplus \vec{p} \oplus \vec{q}) \\
W^c\vec{x}_0 - W^c\vec{x}_1 &\overset{!}{=} W^c\vec{x}_1 - W^c\vec{x}_0 \\
W^c\vec{x}_0 &= W^c\vec{x}_1 \qquad (20)
\end{aligned}$$

by combining the conditions $A0, B1$ with $A1, B0$. Condition $C1$ and Equation 20 leads to

$$\begin{aligned}
\vec{q} &= g(\vec{x_1}, \vec{q}, \vec{q}, W) \\
&= \sigma_g(W(\vec{1} \oplus \vec{x}_1 \oplus \vec{q} \oplus \vec{q}))
\end{aligned}$$

$$\begin{aligned}
&= \sigma_g(W^1\vec{1} + W^c\vec{x}_1 + W^l\vec{q} + W^r\vec{q}) \\
&= \sigma_g(W^1\vec{1} + W^c\vec{x}_0 + W^l\vec{q} + W^r\vec{q}) \\
&= \sigma_g(W(\vec{1} \oplus \vec{x}_0 \oplus \vec{q} \oplus \vec{q})) \\
&= g(\vec{x_0}, \vec{q}, \vec{q}, W) \\
&= \vec{p}
\end{aligned}$$

which is a contradiction to the assumption that p and q are distinguishable states. Therefore no FA consisting only of a one-layered folding part with first-order connections (regardless of the dimensions) and a monotonous sigmoid transfer function can compute arbitrary state transition functions. ■

Remarks This proof is an alternative formulation of the well-known fact that a XOR function cannot be separated by a neural network consisting of only one layer with first-order connections.

6.2.3 The State-Splitting Approach for First-Order Connections

At the first glance, Theorem 6 and Theorem 7 give a rather pessimistic view on the computational power of the FA with first-order connections and only one layer in the folding part. But already Minsky [51] (and later Goudreau et al. [28]) observed that for any finite state automaton that cannot be implemented by a first-order recurrent neural network with one layer there exists another one with identical functional behavior which can be realized. This argument can also be applied to tree automata and the folding architecture.

Theorem 8 *The FA with first-order connections, one layer in the folding part and one unit as transformation part provided with the general sigmoid transfer function can implement any DTA.*

Proof Here we use the so-called *state-splitting* technique of Goudreau et al. [28], where several states in the new automaton play together the same role as a single state in the original one. The idea is to "precompile" information about the direct transition history (only one step back) into the state description.

Let $\mathcal{A} = (\Sigma, Q, F, R)$ be an arbitrary DTA with $\Sigma = \{a_1, \ldots, a_n\}$. We define the DTA $\mathcal{A}' = (\Sigma', Q', F', R')$ as follows:

$$\begin{aligned}
\Sigma' &= \Sigma \\
Q' &= \{\langle q_1, \ldots, q_k, q, a\rangle \mid (q_1, \ldots, q_k, q) \in \delta_a, \delta_a \in R\} \cup \{\langle q_0\rangle\} \\
F' &= \{\langle q_1, \ldots, q_k, q_f\rangle \mid \langle q_1, \ldots, q_k, q_f\rangle \in Q', q_f \in F\} \\
R' &= \{\delta'_a \mid a \in \Sigma'\} \\
\delta'_a &= \{(q'_1, \ldots, q'_k, q') \mid q'_i, q' \in Q', q' = \langle q_1, \ldots, q_k, q, a\rangle, \\
&\qquad \forall i \; (q'_i = \langle\, , \ldots, q_i, \,\rangle \quad \text{or} \quad q'_i = \langle q_0\rangle\,)\}
\end{aligned}$$

The special state $\langle q_0\rangle$ is introduced to capture DTA which are filled up to maximum rank k (see Definition 3). Obviously, the construction of $\mathcal{A}'$ assures the identical functional behavior as the original DTA $\mathcal{A}$. i.e. $L(\mathcal{A}') = L(\mathcal{A})$.

Let $t \in S$ be a given tree and v a node in t whose successors are already assigned the states $(q'_1, \ldots, q'_k)$. By Definition 2 we follow the scheme of the proof to Theorem 4 and reformulate state transitions in a propositional way. $\mathcal{A}'$ assigns v the state $q' = \langle q_1, \ldots, q_j, \underbrace{q_0, \ldots, q_0}_{k-j}, q, a\rangle$ iff

$$(a = \lambda(v)) \bigwedge_{i=1}^{j} \Big(\bigvee_{\substack{u=\langle\, ,\ldots,q_i,\, \rangle \\ u \in Q}} (q'_i = u) \Big) \bigwedge_{i=j+1}^{k} (q'_i = \langle q_0 \rangle) \tag{21}$$

$\mathcal{A}$ can be mapped into FA(1,1,$|Q'|$,n,k,1,ν_1,σ_g) by the following construction (see Figure 5). We choose again the "one-of-n" encoding c for labels with $c(a_i) = e_i$, $a_i \in \Sigma$, i.e. the i-th symbol is mapped to the i-th unit vector. The first and only layer of the folding part consists of $m' = |Q'|$ units each of them representing one state of $\mathcal{A}'$ (the i-th unit corresponds to the i-th state $q'_i \in Q'$). The determinism of $\mathcal{A}$ guarantees that at most one term in each inner disjunction of Condition 21 is true. Thus, Lemma 3 (and Lemma 1c) can be applied to directly translate the Boolean Condition 21 into weight connections for the FA.

The $\iota(q')$-th unit representing $q' = \langle q_1, \ldots, q_k, q, a\rangle$ receives connections from the $\iota(a)$-th unit of the label block and for each block $i = 1, \ldots, k$ from all units which represent the special state $\langle q_0 \rangle$ or states of the scheme $\langle \ldots, q_i, \rangle$ (see Figure 5). The empty tree (corresponding to state $\langle q_0 \rangle$) is coded by e_0. Finally we add a single output unit in the transformation part of the FA which acts as a $|F'|$-ary logical $\vee$ and receives inputs from each unit $\iota(q')$ from the transformation layer with $q' \in F'$. The weights and biases are tuned according to Lemma 3 (and Lemma 1b,c) and the discretization parameter is chosen by applying the Circuit Lemma (Lemma 2). The "undefined state argumentation" for DTA that are not completely specified can be borrowed from the proof construction to Theorem 4.

Obviously (compare Equation 2), the tree t is mapped by the FA to a high output iff the DTA $\mathcal{A}$ accepts t. ∎

Since $m' = |Q'| = n_R + 1$ the complexity of the above FA construction can be expressed using the complexity of the original DTA $\mathcal{A}$ (see Table 4). Again, let n_f be the number of transitions leading to final states, i.e. $n_f = |\{(p_1, \ldots, p_k, p) \in \delta \mid \delta \in R, p \in F\}|$.

	n_R	$n_R^\star$
neurons	$m' = n_R + 1 =$ $O(n_R)$	$m' = m^k n + 1 =$ $O(m^k n)$
weights (full connectivity)	$(n + km')m' + 2m' + 1 =$ $km'^2 + m'(n+2) + 1 =$ $O(kn_R^2 + nn_R)$	$(n + k(m^k n + 1))(m^k n + 1) + 2(m^k n + 1) + 1 =$ $k(m^k n + 1)^2 + (m^k n + 1)(n+2) + 1 =$ $O(km^{2k} n^2)$
weights (non-zero)	see text	$O(km^{2k-1} n^2)$, see Lemma 6

Table 4: Complexity of the DTA implementation given by the state-splitting technique (proof of Theorem 8).

The non-trivial case of counting the number of non-zero weights (see the last row of Table 4) requires a detailed explanation. Equation 21 leads to following formula for the number of non-zero

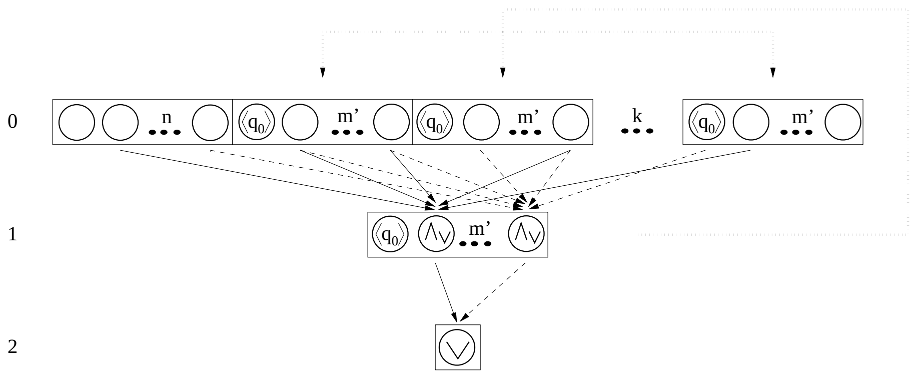

Figure 5: Sketch of the layout of FA's for simulating DTA with first-order connections, one layer in the folding part and one output unit as the transformation part. Neurons are denoted by circles, those acting as Boolean functions are filled with $\wedge_\vee$ (see Condition 21) and $\vee$ signs, groups of semantically similar neurons are enclosed by boxes. The solid lines with arrows exemplify connections to be inserted for a automaton transition function that has maximum rank k. dashed lines the case for a transition with rank $k+1$ using the special state (denoted by "$\langle q_0\rangle$" inside the circles).

weights (here the number of biases are omitted) in the case where not all possible state transitions are explicitely given by the DTA specification.

$$n_f + n_R + \sum_{q\in Q}\left(\sum_{(q_1,\dots,q_j,q_0,\dots,q_0,q)\in\delta,\delta\in R}\left(\sum_{i=1}^{j}|\{(p_1,\dots,p_k,q_i)\in\delta \mid \delta\in R\}| + k - j\right)\right) \tag{22}$$

The number of connections received by the output neuron is n_f. One connection is required from the label block of the input for each state $q'\in Q'$ (there are n_R states in total, not counting the special state). The exact number of non-zero weights needed to implement the inner disjunction of Condition 21 depends on the concrete distribution of transitions to states. However, the number of weights in the case $n_R^\star$ gives also an valid upper bound for the former.

Lemma 6 *The number of non-zero weights applying the state-splitting implementation technique (proof construction of Theorem 8) for the case when all possible transitions are specified ($n_R^\star$) can be derived to $O(km^{2k-1}n^2)$.*

Proof Let all symbols have the same maximum rank k. For a DTA with m states and n input symbols with maximum rank k there are $n_R^\star = m^k n$ possible transitions which are distributed among m different target states. Let the partition size for state i be denoted by π_i. Thus we get

$$n_R^\star = m^k n = \sum_{i=1}^{m}\pi_i$$

Since the total number of occurrences of each state symbol in all left sides of transitions $(\{(p_1,...,p_k) \mid (p_1,\dots,p_k,q)\in\delta,\delta\in R\}$ is nkm^{k-1}, the Equation 22 can be rewritten to

$$n_f + n_R + n\sum_{i_1,\dots,i_k\in\{1,\dots,m\}}(\pi_{i_1}+\cdots+\pi_{i_k}) \;=\; n_f + m^k n + nkm^{k-1}\sum_{i=1}^{m}\pi_i \;=\; n_f + m^k n + km^{2k-1}n^2$$

resulting in an upper bound for the number of non-zero weights of $O(km^{2k-1}n^2)$, which is not affected by including the number of biases and is still valid when the DTA is filled up (see Definition 3) to the maximum rank k. ■

One might assume that the transformation part is superfluous (in analogy to the two-layered folding part construction by Corollary 1). However, the next theorem gives a negative result.

Theorem 9 *The FA with first-order connections, a monotonous general sigmoid transfer function, one layer in the folding part and no transformation part cannot achieve DTA power.*

Proof We show that it is impossible to add special state neurons to the folding layer to compute arbitrary output functions. Consider the "even-parity tree recognizer" presented as Example 1. The automaton is in the state p (q) if the presented tree is of even (odd) parity. Without loss of generality, we assume there is at least one neuron i in the folding part for which the output vectors $\vec{p}$ and $\vec{q}$ representing p and q ("accepted" and "not accepted") differ. By transition δ_0 node i must assume one value whenever the two predecessor states are identical ((p,p) and (q,q)) and neuron i must get a different value for the other two cases ((p,q) and (q,p)). However, this behavior is that of the function XOR which is known as inseparable by σ_g (see also proof of Theorem 7). This implies that neuron i cannot compute the desired output function. ■

If we want to do it without transformation part, then the modus operandi of tree processing (compare to Section 4.3) has to be changed in the following way.

Let a special output unit be added to the representation layer and connected via $\vee$-gate to all units (of the k-th block) in the input layer that represent final states of the automaton. After a tree is completely processed by the folding part the output of the representation layer is once again fed back to the (k-th block of the) input layer (all other inputs are set to 0) before the output of the network is computed. Clearly, the output unit is high iff the tree is accepted by the DTA.

Corollary 2 *The FA with first-order connections, a monotonous general sigmoid transfer function, one layer in the folding part and no transformation part can achieve DTA power with a one step delay of the output computation.*

6.3 The Computational Power of the FA in the Threshold Case and the Relation to Context-Free Word Languages

In the previous section several simple instances of the FA were proven to simulate arbitrary DTA. The next result shows that the usage of threshold transfer functions in the FA limits the computational power to that of DTA.

Theorem 10 *Let Σ' be a given finite ranked alphabet. Any FA with threshold transfer function σ_t, one output unit and operating on trees $t \in \mathcal{T}(\Sigma')$ can be simulated by a DTA.*

Obviously, those FA can only model a finite number of states and a finite number of different state transitions.

Proof Any given FA(_, _,m,n,k,1,_,σ_t) can be simulated by a DTA $\mathcal{A} = (\Sigma, Q, F, R)$ with

$$\begin{aligned}
\Sigma &= \Sigma' \\
Q &= \{\tilde{g}(t) \mid t \in \mathcal{T}(\Sigma')\} \\
F &= \{\tilde{g}(t) \mid t \in \mathcal{T}(\Sigma'), \Xi_A(t) = 1\} \\
R &= \{\delta_a \mid a \in \Sigma\} \\
&\quad \text{where } \delta_a = \{(q_1, \ldots, q_k, q) \mid q_i \in Q, q = g(c(a) \oplus q_1 \oplus \cdots \oplus q_k)\}
\end{aligned}$$

The the resulting DTA $\mathcal{A}$ has at most 2^m different states and at most $|\Sigma'| \cdot 2^{mk}$ different state transitions. ■

Thus, the FA is equivalent to DTA (and therefore to its nondeterministic counterpart and the nondeterministic top-down tree automaton, see remark at the end of Section 3) in terms of computational power when using threshold transfer functions and moderate constraints on the architecture (as shown in the previous section). In the constructive proofs presented for simulating DTA by FA we applied a discretization scheme, sigmoid transfer functions are only casted into Boolean functions, their full potential power was not explored.

Intuitively, a FA with sigmoid transfer functions and two layers as folding and two layers as transformation part should enable the realization of machines with infinite states and arbitrary state transitions.

But it is still an open question whether the FA equipped with sigmoid transfer functions is strictly more powerful than DTA (see Section 8.4). To describe the interesting relationship between context-free word languages (CFL) and the FA it is convenient to introduce a function which extracts a word from the frontier of a tree.

Definition 4 (Yield) The *yield* $yd(t)$ of a tree $t \in \mathcal{T}(\Sigma)$ is defined inductively as follows:

$$yd(t) = \begin{cases} a & \text{if } a \in \Sigma \text{ with } r(a) = 0, \\ yd(t_1) \ldots yd(t_n) & \text{if } t = f(t_1, \ldots, t_n) \text{ with } f \in \Sigma \text{ and } r(f) = n. \end{cases}$$

The *yield* of a tree language $\mathcal{L}$ is the language $yd(\mathcal{L}) = \{yd(t) \mid t \in \mathcal{L}\}$. We say that a language $\mathcal{L}$ is *yd-recognized* by a tree recognizer $\mathcal{A}$ if $\mathcal{L} = yd(L(\mathcal{A}))$.

Theorem 11 *The FA equipped with sigmoid transfer functions can yd-recognize at least context-free word languages.*

Proof A well-known result tells us even a subclass of DTA yd-recognizable languages being equivalent to CFL (see Doner [12], Gécseg and Steinby [20]).

Informally spoken, that proof uses the fact that all derivation trees of a given CFL are DTA recognizable. On the other hand, the yield of any DTA-recognizable tree language is a context-free language. Thus, with Theorem 4 (or alternatively Theorem 5 or Theorem 8) it is proven that the FA can yd-recognize at least CFL. ■

Definition 4 and Theorem 11 give the impression that the FA with threshold transfer functions should have the capability to represent CFL. However, it is only a device which recognizes the possible syntaxes of a given context-free word and cannot be practically used as recognizer of the word itself.

7 Bounds on the Node Complexity of FA Implementations of DTA

In Section 6 we gave several construction schemes to implement DTA into FA (under different constraints on the architecture). So far, the most economical first-order network implementation was obtained by a FA with two layers in the folding part ($O(m^k n)$, in terms of node complexity). An interesting question is whether the node complexity can in principle be further reduced by using more layers in the folding part. Can we estimate the minimal number of neurons required to accomplish the DTA simulations?

The static FA may be interpreted as a so-called *linear threshold circuit* (LTC) when instantiated by the threshold function σ_t as transfer function (a neuron with first-order connections can be identified as a threshold gate). There has been significant work done on characterizing the node complexity of neural networks by applying known results from the circuit complexity of linear threshold circuits (see Alon et al. [1], Horne and Hush [35, 36], Siu et al. [68, 67]).

Next we show how some of these results can easily be lifted to the context of FA and DTA. One might especially interested in deriving upper and lower bounds on the node complexity of the FA such that arbitrary DTA can be implemented by. The line of argumentation is essentially borrowed from the investigations on the node complexity of RNN implementations of FSA carried out by Horne and Hush [36].

7.1 Upper Bounds

An upper bound on the node complexity can be derived by looking for a LTC (represented by the folding part of the FA) that enables the implementation of arbitrary Boolean functions and thereby arbitrary DTA state transition functions. The following lemmas recall well-known bounds on the node complexity of LTC implementations of Boolean functions.

Lemma 7 (Node Complexity, LTC) *Arbitrary Boolean functions of the form $\phi : \{0,1\}^x \to \{0,1\}^y$ can be implemented in a LTC using r layers with a node complexity of*

(a) $O(\sqrt{y 2^x/(x - \log y)})$, $r = 4$, by Lupanov [47];

(b) $O(y 2^{x/2})$, $r = 3$, multi-output extension of the single-output result of Siu et al. [67, 68];

(c) $O(2^x + y)$, $r = 2$, by Horne and Hush [35].

In the following we consider folding architectures where the transformation part consists of two layers and exactly one output unit.

Theorem 12 *Any given DTA having m states, an alphabet of size n with maximum rank k can be implemented by a FA(r,2,$\lceil \log m \rceil$, $\lceil \log n \rceil$,k,1,ν_1,σ_t) with a node complexity of*

(a) $O\left(\sqrt{\frac{nm^k 2^k \log m}{\log n + k \log m}}\right)$, $r = 4$;

(b) $O(\log m \sqrt{nm^k 2^k})$, $r = 3$;

(c) $O(nm^k 2^k)$, $r = 2$.

Proof A m-state DTA with an alphabet size n of maximum rank k can be implemented in a FA where the folding part performs a mapping of the form [3]:

$$\phi : \{0,1\}^{\lceil \log n \rceil + k \lceil \log m \rceil} \rightarrow \{0,1\}^{\lceil \log m \rceil}$$

When considering upper bounds the original problem can be reduced to the node complexity of Boolean functions where:

$$\begin{array}{rcccl} \log n + k \log m & \leq & x & \leq & \log n + 1 + k(\log m + 1) \\ \log m & \leq & y & \leq & \log m + 1 \end{array}$$

By Lemma 7 we obtain a r-layer LTC with a node complexity of

(a) $O(\sqrt{\frac{(\log m+1)2^{(\log n+1+k(\log m+1))}}{\log n+k\log m-\log(\log m+1)}}) = O(\sqrt{\frac{nm^k 2^k \log m}{\log n+k\log m}})$ $(r = 4)$

(b) $O((\log m + 1)\sqrt{2^{\log n+1+k(\log m+1)}}) = O(\log m \sqrt{nm^k 2^k})$ $(r = 3)$

(c) $O(2^{\log n+1+k(\log m+1)} + \log m + 1) = O(nm^k 2^k)$ $(r = 2)$

The discrimination of accepting automaton states can be implemented by a two-layered transformation part (yielding a node complexity of $O(m)$, see Lemma 7 (c)) which performs a mapping of the type $\phi : \{0,1\}^{\lceil \log m \rceil} \rightarrow \{0,1\}$. Assuming $k \geq 2$, the overall node complexity is not affected.

■

As demonstrated by Corollary 1 we can do it without the transformation part by adding one unit to the layer r which acts as the designated output unit of the network. By this modification the folding part computes a Boolean function of the type $\phi : \{0,1\}^{\lceil \log n \rceil + k(\lceil \log m \rceil + 1)} \rightarrow \{0,1\}^{\lceil \log m \rceil + 1}$.

Corollary 3 *The implementation of DTA into architectures of the type FA(r,0,$\lceil \log m \rceil + 1$, $\lceil \log n \rceil$,k,1,ν_1,σ_t) results in a slight increase of the node complexity to*

$$O(2^k \sqrt{\frac{nm^k \log m}{\log n + k \log m}}) \quad , \quad O(2^k \log m \sqrt{nm^k}) \quad , \quad O(nm^k 2^{2k}) \quad \textit{for} \quad r = 4, 3, 2.$$

7.2 A Way Directed to a Lower Bound

The upper bound is not a very surprising result. One might have expected that it is possible to reduce the resource consumption by using more layers and a more economical way of state and input encoding (in contrast to the "one-of-n" encodings applied in our hand-crafted constructive proofs). But what is the minimum number of neurons required to implement arbitrary DTA? A first naive consideration would lead to $O(\log m)$ since m states can be effectively encoded by $\lceil \log m \rceil$ neurons.

Here we want to follow the way shown by Alon et al. [1] and Horne and Hush [36] in deriving a lower bound for the node complexity of RNN implementations of FSA.

Let $K(m)$ be the smallest number such that every DTA with m or less states can be implemented by a FA using $K(m)$ or fewer neurons. Let $L(m)$ be the number of pairwise *divergent* (see

[3]By log we implicitly mean the logarithm to the base 2.

Definition 5) DTA with m or fewer states and $U(z)$ be the number of different m-state DTA that can be built from a FA using z neurons. Obviously,

$$U(K(m)) \quad \geq \quad L(m) \tag{23}$$

and by deriving an good upper bound for $U(z)$ and a lower bound $L(m)$ we might be able to compute a lower bound for $K(m)$.

Lemma 8 *The number $U(z)$ of different m-state DTA (over an alphabet of size n of maximum rank k) that can be built from a FA with z neurons can be bounded to $O(y2^{xz^2})$, where $x = \lceil \log n \rceil + k(\lceil \log m \rceil + 1)$ and $y = \lceil \log m \rceil + 1$.*

Proof Following Horne and Hush [35, 36] we consider the folding part of a FA constituted by a *lower triangular* network (LTN), i.e. a network where the l-th node is the only node in layer l and receives input from all nodes in the previous layers (including input layer). Multilayer feedforward networks may be viewed as a special LTN where some weights are set to zero.

The folding part of a FA (implementing a DTA) can be interpreted as a function $\phi : \{0,1\}^x \rightarrow \{0,1\}^y$ where

$$\begin{aligned} x &= \lceil \log n \rceil + k(\lceil \log m \rceil + 1) \\ y &= \lceil \log m \rceil + 1 \end{aligned}$$

The total number of Boolean functions which are representable by a LTN with z units, x inputs and y outputs is bounded by the maximum number of functions that the first node can compute, multiplied by the maximum number of logic functions that the second can compute and so on.

According to Muroga [53] the number of logic functions defined over ν vertices of the unit hypercube L_w^ν that can be implemented by a single threshold neuron with w inputs can be bounded by

$$L_w^\nu \quad \leq \quad \frac{2\nu^w}{w!} \tag{24}$$

if $\nu \geq 3w + 2$. The number of inputs in the 0-th layer of the LTN is x and the number of inputs to the i-th node (numbered from 0 to $z-1$) is $x+i$. Thus, by Equation 24 this node can compute at most $2\nu^{(x+i)}/(x+i)!$ logic functions. The maximum number of functions in a LTN with z nodes is given by

$$\prod_{i=0}^{z-1} \frac{2\nu^{(x+i)}}{(x+i)!}$$

We see that ν is at most 2^x, since this is the maximum number of different inputs to the network. Because the outputs are determined by input patterns, 2^x is also the maximum number of different inputs that nodes at higher layers receive even though their fan-in is greater than x. Equation 24 holds for $\nu \geq 3(x+i)+2$, which can be justified for $\nu = 2^x$ and $z \leq 2^x/3$. The total number of outputs of the LTN will be equivalent to y. The last node must be an output or state variable. Of the remaining $z-1$ nodes, $y-1$ must be chosen as outputs or state variables (this requires $z \geq y$) yielding

$$\binom{z-1}{y-1}$$

choices. Since the ordering of outputs is relevant, we get for the maximum number of logic functions that a LTN can compute, i.e. an upper bound for $U(z)$:

$$U(z) \leq y!\binom{z-1}{y-1}\prod_{i=0}^{z-1}\frac{2^{x(x+i)+1}}{(x+i)!} = \frac{y(z-1)!}{(z-y)!}\prod_{i=0}^{z-1}\frac{2^{x(x+i)+1}}{(x+i)!}$$

By taking the logarithm of both sides (and using the inequality $\log x! \geq x\log x - (x - 1/\ln 2)$) we get

$$\begin{aligned}\log[U(z)] \quad &\leq \quad \log y + \log z! - \log[(z-y)!] + \\ &\sum_{i=0}^{z-1}\left(1 + x(x+i) - (x+i)\log(x+i) + \frac{x+i-1}{\ln 2}\right)\end{aligned}$$

Simplification (by summation and elimination of some negative terms, having in mind that $z > x$) of the right side leads to

$$\log[U(z)] \quad \leq \quad \log y + z\log z + z\left(1 + x^2 + \frac{x-1}{\ln 2}\right) + \frac{z(z-1)}{2}\left(x + \frac{1}{\ln 2}\right)$$

Assuming $z \geq x \geq 2$, then there exists a constant c such that

$$\log[U(z)] \quad \leq \quad cz^2x + \log y$$

leading to an upper bound for $U(z)$ of $O(y2^{xz^2})$. ■

In order to find a lower bound for $L(m)$ we have to render more precisely what is meant by "divergent" DTA.

Definition 5 (Divergent DTA) Two DTA $\mathcal{A}_i(\Sigma, Q_i, F_i, R_i)$ $(i = 1, 2)$ are *divergent* iff $L(\mathcal{A}_1) \neq L(\mathcal{A}_2)$, i.e. there exists an input tree $t \in \mathcal{T}(\Sigma)$ with $\tilde{\delta_1}(t) \in F_1$ *iff* $\tilde{\delta_2}(t) \notin F_2$.

Let $\mathcal{A}(\Sigma, Q, F, R)$ be a DTA with $n = |\Sigma|$, $m = |Q|$, $n_R = \sum_{\delta \in R}|\delta|$ and $n_F = |F|$. By considering transition functions as $\delta : Q^k \times \Sigma \rightarrow Q$ we can compute the number of different transition functions by $m^{(nm^k)}$. This gives an appealing approximation to the number of different DTA with m states,

$$\binom{m}{n_F}\frac{m^{(nm^k)}}{m!}$$

but it is obvious that not each pair of them is divergent. So far, the way for estimating a lower bound $K(m)$ for the node complexity of DTA into FA implementations can be done in analogy to the RNN case (Horne and Hush [36]). However, the calculation of $L(m)$ given by Alon et al. [1] is based on a sophisticated construction of Mealy Automata (i.e. FSA where simultaneously to a state transition an output is generated depending on the input and the actual state) and cannot simply be transferred to the DTA case.

The next Lemma and its proof is due to Hammer [31] and gives a rough estimation for the number of divergent DTA.

Lemma 9 *There are at least* $L(m) = 2^{(n-1)(m-1)^k}$ *pairwise divergent DTA with alphabet size* n *of maximum rank* k *and* m *states.*

Proof Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet with $r(a_1) = \{0, 1\}$ and $r(a_i) = k$, $2 \le i \le n$. Further let $Q = \{q_1, q_2, \ldots, q_m\}$ be a set of states, $F = \{q_m\}$ be the set of final states and assume $n, m \ge 2$. Define $q_1 \in \delta_{a_1}$ and

$$\delta_{a_1}(q_i) = \begin{cases} q_{i+1} & \text{if } i < m, \\ q_i & \text{otherwise.} \end{cases}$$

The $(n-1)(m-1)^k$ trees $t_{(i,i_1,\ldots,i_k)}$ of the form

$$a_i(\underbrace{a_1(a_1(\ldots a_1(a_1)}_{i_1}\ldots)), \ldots, \underbrace{a_1(a_1(\ldots a_1(a_1)}_{i_{k-1}}\ldots)), \underbrace{a_1(a_1(\ldots a_1(a_1)}_{i_k}\ldots)))$$

where $i \in \{1, \ldots, n-1\}, i_1, \ldots, i_k \in \{1, \ldots, m-1\}$ have to be mapped by an automaton according to $\tilde{\delta}(t_{(i,i_1,\ldots,i_k)}) = \delta_{a_i}(q_{i_1}, \ldots, q_{i_k})$. For different trees $t_{(i,i_1,\ldots,i_k)}$ the terms $\delta_{a_i}(q_{i_1}, \ldots, q_{i_k})$ are different, therefore each mapping

$$\tilde{\delta} : \{t_{(i,i_1,\ldots,i_k)} \mid i \in \{1, \ldots, n-1\}, i_1, \ldots, i_k \in \{1, \ldots, m-1\}\} \to \{q_1, q_m\}$$

can be implemented by an appropriate choice of δ_{a_i} and thus at least $2^{(n-1)(m-1)^k}$ divergent tree automata exist. ∎

We are now ready to combine the presented results to a statement about a lower bound for the node complexity of DTA into FA implementations.

Theorem 13 *Any given DTA having m states, an alphabet size of n with maximum rank k can be implemented by an architecture FA(_,0,$\lceil \log m \rceil + 1$, $\lceil \log n \rceil$,k,1,ν_1,σ_t) with a node complexity of*

$$\Omega(\sqrt{\frac{nm^k}{\log n + k \log m}}).$$

Proof By combining Lemma 8 and Equation 23 with Lemma 9 we get

$$\begin{array}{rcccl} L(m) & \le & U(z) & \le & cy2^{xz^2} \\ z & = & K(m) & = & \Omega\left(\sqrt{\frac{1}{x}\log\left[\frac{L(m)}{cy}\right]}\right) \end{array}$$

Substitute $L(m)$ by $2^{(n-1)(m-1)^k}$, x by $\lceil \log n \rceil + k(\lceil \log m \rceil + 1)$ and y by $\lceil \log m \rceil + 1$. ∎

Remarks Lemma 9 gives only a very rough estimation of the number of divergent DTA. Thus, we conjecture that the lower bound on the node complexity of DTA into FA implementations (Theorem 13) can be improved.

It seems that an increase in the number of layers allowed in the folding part of the FA results in a decrease of the node complexity (see Section 7.1, Theorem 12 and Siu et al. [67]). One might conjecture that the constraints imposed by a given automaton on the transition function do not allow to cover the whole space of Boolean functions. Our greedy (using a "one-of-n" encoding for labels and states) two-layered construction and Corollary 1) with a node complexity of $O(nm^k)$ is at least competitive to the result given in Theorem 12 (c). It is an open question whether the upper bounds can be further improved (assuming a restricted number of layers r in the folding part).

The implantation of sigmoid transfer functions might lead to a further reduction in the node complexity. This hypothesis is supported by the observation that for certain Boolean functions the size of the implementing network can be reduced by at least a logarithmic factor by using continuous (e.g. sigmoid) instead of threshold gates (Siu et al. [68]).

8 On Pattern Languages and the Adequacy of the Automata Framework

We have seen that FA and DTA are equivalent regarding their computational power if the FA is built of neurons with threshold transfer functions. But what is the situation with sigmoid transfer functions? Theorem 8 tells us that such a FA has the computational power of at least DTA. But is it strictly more powerful than DTA? Can the computational power be adequately characterized by the classical formal language (automata) framework?

In the past, a bunch of empirical investigations on the learnability of artificially generated tree languages (formulated as inductive grammatical inference task given positive and negative examples) have been carried out with the FA and gradient-based learning procedures (see Schmitt [60], Goller [25], Schulz et al. [62], Calmbach [8], Küchler and Goller [46], Goller and Küchler [26]). Even tasks intuitively estimated to be hard for the FA were solved with fairly high generalization accuracy.

In this section we will identify the tree languages used in former experiments belonging to the class of so-called *pattern languages*. We show that the DTA and thereby the FA has the capability to recognize at least interesting subclasses if the defining pattern is constrained to be *linear*. *Nonlinear* pattern languages turn out to be beyond the representational power of the DTA. However, the FA performs surprisingly well on grammatical inference tasks over non-linear pattern languages.

This phenomenon seems to give evidence that the full computational power of the FA with sigmoid transfer functions is strictly stronger than that of DTA. We formulate some conjectures about the adequacy of the formal automata framework to characterize the computational power of the FA. Some experiments carried out with the FA will serve as an illustration of the questions raised above.

8.1 Pattern Languages and the Language Boundary Induced by Occurrence Constraints on Variables

Pattern languages and the inductive inference of pattern languages are well-known formal concepts in computer science (for example see Shinohara and Arikawa [64]) which recently gained growing attention due to the application potential in computational biology (Arikawa et al. [5]). Here, we are specially interested in tree patterns, i.e. terms with variables, and languages induced by tree patterns.

Definition 6 (Pattern) Let Σ be a ranked alphabet and $\mathcal{V}$ be a finite set of variables with $\Sigma \cap \mathcal{V} = \emptyset$. A term t of $\mathcal{T}(\Sigma, \mathcal{V})$ is synonymously called a *pattern*. A pattern is said to be *linear* if no variable occurs in it more than once and *non-linear*, otherwise.

Next, we define a tree language by a pattern occurrence condition.

Definition 7 (Pattern Occurrence Language) Let p be a pattern of $\mathcal{T}(\Sigma, \mathcal{V})$. The *pattern occurrence language* (OL) induced by p is defined as

$$OL[p] = \{t \mid t \in \mathcal{T}(\Sigma),\ \text{there exists a position } i \text{ and a substitution } \sigma \text{ with } t|_i = \sigma(p)\}$$

The language is called *linear* pattern occurrence language (LOL) if the defining pattern is linear.

Lemma 10 *The class of LOL is a proper subset of the class of languages recognizable by DTA.*

The way from linear patterns to tree automata is well-known and for example applied in the compiler construction field to automatically generate pattern matching devices for rewrite rules describing code optimization strategies (see Wilhelm and Maurer [81]).

Proof

1. Wilhelm and Maurer [81] presented a simple construction scheme where a non-deterministic bottom-up tree automaton is built from a given linear pattern π. Since (here) variables do not need to be distinguishable, we replace all the variables in π by $\square$ ("a variable matching everything") resulting in $\pi \in \mathcal{T}(\Sigma, \{\square\})$.

   We define the nondeterministic bottom-up tree automaton $\mathcal{A}_\pi = (\Sigma_\pi, Q_\pi, F_\pi, R_\pi)$ as follows:

$$\begin{array}{rcl} \Sigma_\pi & = & \Sigma \cup \{\square\} \\ Q_\pi & = & \{t \mid t \sqsubseteq \pi\} \\ F_\pi & = & \{\pi\} \\ R_\pi & = & \{\delta_{\pi,a} \mid a \in \Sigma_\pi\} \text{ where } \delta_{\pi,a} = \\ & & \{(\square, \ldots, \square, \square) \in Q_\pi^{n+1} \mid r(a) = n\} \cup \\ & & \{(t_1, \ldots, t_n, t) \mid t \in Q_\pi \text{ and } t = a(t_1, \ldots, t_n)\} \cup \\ & & \{(q_1, \ldots, q_n, \pi) \mid r(a) = n, q_i \in Q_\pi \text{ and there is a } i \text{ with } q_i = \pi\}. \end{array}$$

   Obviously, for every tree t there are transitions in $\mathcal{A}_\pi$ and there is a sequence of transitions leading to the final state π if t matches π. Finally, the subset construction technique is applied to transform the automaton $\mathcal{A}_\pi$ to a deterministic one.

2. There are languages which are recognizable by DTA but that are not in the class OL. Consider the "even-parity tree recognizer" from Example 1. Suppose that π is a pattern that is capable of inducing the even-parity tree language $\mathcal{L}$. Let σ be an arbitrary substitution. Then we get $\sigma(\pi) \in \mathcal{L}$ which leads to the contradiction $1(0, \sigma(\pi)) \notin \mathcal{L}$. ■

The pattern language is beyond the representational capability of the class of DTA if the linearity condition is dropped.

Lemma 11 *The class OL cannot be recognized by DTA.*

Proof Let π be a non-linear pattern with exactly one variable X occurring exactly twice. Suppose there exists a DTA $\mathcal{A}$ which maps a term u to a final state iff $u \in OL[\pi]$.

Due to the finite-state behavior of a DTA there must be at least two different terms $s, t \in \mathcal{T}(\Sigma)$ such that $\tilde{\delta}(s) = \tilde{\delta}(t)$. If we substitute s for one occurrence and t for the other occurrence of the variable X into π then a term π' is generated that does not belong to the pattern language $OL[\pi]$. But $\mathcal{A}$ would map π' to a final state (because of the compositionality of the mapping $\tilde{\delta}$) which leads to an obvious contradiction. ∎

Example 2 Let $\mathcal{V} = \{X\}$ be a set of variables and $\Sigma = \{a, b, f\}$ be a ranked alphabet with $r(a) = r(b) = 0$ and $r(f) = 2$. $\mathcal{L}_1$, $\mathcal{L}_2$ are defined as pattern matching languages:

$$\begin{aligned} \mathcal{L}_1 &= OL[f(a,b)] \\ \mathcal{L}_2 &= OL[f(X,X)] \end{aligned}$$

For example, $f(a,b)$, $f(a,f(a,b))$, $f(f(a,b),f(a,b)) \in \mathcal{L}_1$ while $f(b,a)$, $f(f(a,a),f(b,a)) \notin \mathcal{L}_1$ and $f(f(a,b),f(a,b))$, $f(f(a,a),f(b,a)) \in \mathcal{L}_2$ while $f(a,b)$, $f(a,f(a,b))$, $f(b,a) \notin \mathcal{L}_2$. Clearly, $\mathcal{L}_1$ belongs to LOL while $\mathcal{L}_2$ is in not.

The pattern language can be enriched by allowing arbitrary Boolean combinations of patterns as occurrence condition.

Definition 8 (Boolean Occurrence Combination) Let P be a finite set of variable-disjoint patterns over $\mathcal{T}(\Sigma, \mathcal{V})$ (with $\{\wedge, \neg\} \cap (\mathcal{V} \cup \Sigma) = \emptyset$) and F be a Boolean combination of patterns in P. The language $BOL[F]$ induced by F is recursively defined as

$$BOL[F] = \begin{cases} OL[\pi] & \text{if } F = \pi \text{ and } \pi \in P \\ \mathcal{T}(\Sigma) \setminus BOL[G] & \text{if } F = \neg G, \\ BOL[F_1] \cap BOL[F_2] & \text{if } F = F_1 \wedge F_2. \end{cases}$$

and nothing else is in $BOL[F]$. Let BOL denote the class of languages induced by arbitrary finite variable-disjoint pattern sets and Boolean combinations over it. The *linear* fragment $LBOL$ is defined as a proper subclass of BOL by allowing only pattern sets where each pattern is linear.

Example 3 (BOL) Let Σ be a ranked alphabet, $\mathcal{V}$ a set of variables and F a Boolean pattern combination with $\Sigma = \{f, g, a, b\}$, $r(f) = r(g) = 2$, $r(a) = r(b) = 0$, $\mathcal{V} = \{X, Y\}$ and $F = f(X,X) \wedge \neg g(a,Y)$. For example, $f(a,a), f(f(a,a),g(b,a)), f(g(b,b),g(b,b)) \in BOL[F]$ while $f(a,b), g(f(a,a),g(a,a)), g(a,b) \notin BOL[F]$.

Theorem 14 *The class LBOL is a proper subset of the class of languages recognizable by the FA.*

Proof Lemma 10 together with Theorem 4 (or alternatively Theorem 5 or Theorem 8) shows that LOL is a proper subset of the class of languages recognizable (RTL) by DTA which in turn is a subset of the class of languages recognizable by FA. Further, RTL is known to be closed under complement, union and intersection (see Gécseg and Steinby [20]).

And again (following the line of argumentation given by the proof of Lemma 10, 2), the "even-parity tree language" can be recognized by a DTA, but cannot be captured by a Boolean occurrence condition over a finite set of patterns. ∎

Note that the results presented above are still valid when the pattern language is defined by *instantiation* instead of occurrence conditions (as given by Definition 7). In this case a pattern π has to match the given term t at its root position, i.e. t is an *instance* of π iff there is a substition σ with $\sigma(\pi) = t$. Analogously to LOL, OL, $LBOL$ and BOL one can define the language classes LIL, IL, $LBIL$ and BIL. Clearly, instantiation is a special case of occurrence (at the root position) which implies a subset relation between the languages defined by the corresponding matching conditions, i.e. $LIL[\pi] \subset LOL[\pi]$, $IL[\pi] \subset OL[\pi]$, $BIL[F] \subset BOL[F]$ and $LBIL[F] \subset LBOL[F]$ for any pattern π and Boolean combination F.

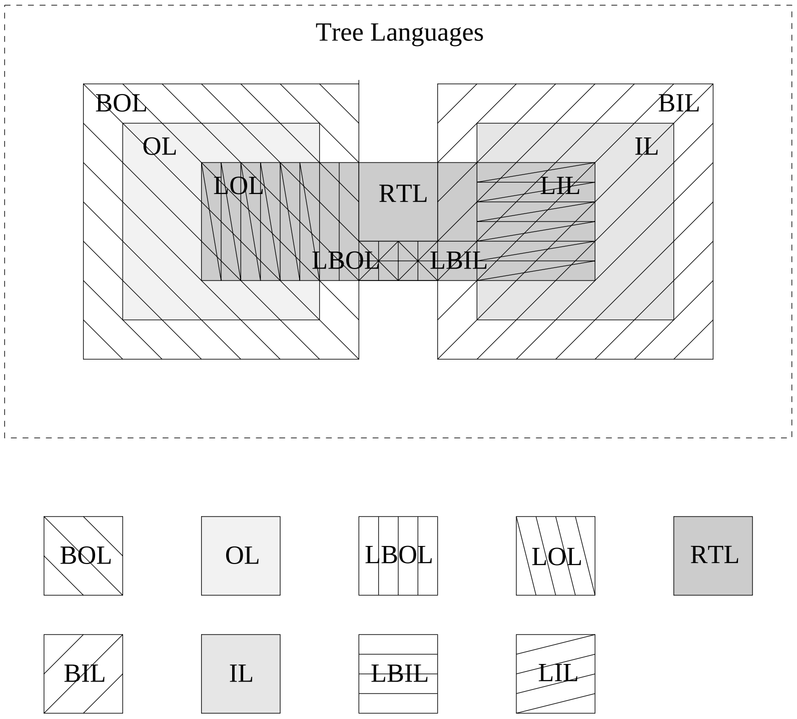

Figure 6: Venn diagram illustrating the relations between RTL and the pattern languages. The size of the area does not reflect the cardinality of the sets. The first subjective visual impression is that of "sunglasses", OL and IL constitute the lenses, framed by $BOL \cap \overline{OL}$ and $BIL \cap \overline{IL}$, connected by RTL.

Lemma 12 *Let Σ be a non-trivial ranked alphabet consisting of at least one symbol with rank 0 and another one with rank greater 0. Further, let TI denote the trivial set intersections $TI = \{\emptyset, \mathcal{T}(\Sigma)\}$. The relations between pattern language classes and to the language RTL (which is exactly the class of languages that corresponds to the machine DTA) can be summarized as follows (see Figure 6):*

1. *$LIL \cap LOL = \emptyset$ and $IL \cap OL = \emptyset$.*
2. *$LBIL \cap LBOL = BIL \cap BOL = TI$.*
3. *$RTL \cap \overline{BOL} \neq \emptyset$ and $\overline{RTL} \cap BOL \neq \emptyset$, $RTL \cap \overline{BIL} \neq \emptyset$ and $\overline{RTL} \cap BIL \neq \emptyset$.*
4. *$LOL \subset LBOL \subset RTL$ and $LOL \subset OL \subset BOL$ and $LBOL \subset BOL$,*

$LIL \subset LBIL \subset RTL$ and $LIL \subset IL \subset BIL$ and $LBIL \subset BIL$.

5. *$RTL \cap OL = LOL$ and $RTL \cap BOL = LBOL \cup TI$,*

   $RTL \cap IL = LIL$ and $RTL \cap BIL = LBIL \cup TI$,

6. *$LBOL \cap OL = LOL$, $LBIL \cap IL = LIL$.*

Proof (Sketch)

1. Obviously, the classes LIL, LOL and IL, OL are incomparable. Given a pattern π_1 the instantiation condition cannot be generated by an occurrence condition and a pattern π_2 and vice versa.

2. The Boolean conditions are constituted over a finite set of pattern. Thus, we have the same situation as in (1), but due to the possibility of generating logical tautologies and antilogies the trivial set TI is contained in both classes.

3. The "even-parity tree language" is in RTL but not in BOL (BIL) (proof of Theorem 14) and the non-linear pattern languages belong to BOL (BIL) but not to RTL (proof of Lemma 11).

4. $LOL \subset LBOL$, $LOL \subset OL \subset BOL$ and $LBOL \subset BOL$ by Definition and due to the fact that one cannot generate a non-linear matching condition by a Boolean combination over a finite set of linear patterns. Theorem 14 implies $LBOL \subset RTL$. The same argumentation holds for instance conditions.

5. Consequences from (3) and (4).

6. By Definition. ■

The language classes LOL, $LBOL$, OL, BOL (and their counterparts obtained by using the instance condition for matching) cover the languages that were formerly used in experiments to explore the learnability capabilities of the FA (see Schmitt [60], Goller [25], Küchler and Goller [46]). The learning tasks have been organized as *inductive inference* tasks (see e.g. Angluin and Smith [3] or Knuutila [40]). Here we use this term in the following sense.

Definition 9 (Inductive Grammatical Inference) *Inductive Grammatical Inference* (IGI) is the process where a learning system attempts to identify a finite representation for a potential infinite set of trees, called a tree language, based on a finite set of examples chosen from the language and its complement.

Theorem 14 shows that the FA with sigmoid transfer functions can at least represent the classes $LOL, LBOL, LIL, LBIL$. But can the FA represent non-linear pattern languages which are proven to be beyond the regular tree languages RTL? Since learnability prerequisites representability one might draw conclusions from empirical results on IGI tasks.

Before proceeding in this direction (Section 8.4) let us take the previously introduced languages $\mathcal{L}_1 \in LOL$, $\mathcal{L}_2 \notin RTL$ to illustrate how the IGI task can be accomplished by the FA.

8.2 The Experimental Setup

Table 5 reports the characteristics of the six term sets that were generated to accomplish the IGI task with the FA for the languages $\mathcal{L}_1$ and $\mathcal{L}_2$. The first column specifies the allowed complexity of the terms, i.e. $\mathcal{T}(\Sigma)$ is constrained to terms up to a maximum depth x and a maximum number of nodes y (denoted by $\mathcal{T}[x, y]$). For each set independently 1000 terms were randomly (under an uniform distribution) drawn from the space $\mathcal{T}[x, y]$ (the cardinality is given in column four) and subsequently labeled (+1/-1 for membership: yes/no) according to the specification for $\mathcal{L}_1$ and $\mathcal{L}_2$. The average depth and average size of a term in each data set is shown in the second and third column. The last two columns give the classification performance a naive guesser should achieve when knowing the class label distribution. The same set $\mathcal{T}[6, 13]$ is used for both tasks, containing 56.1 % positive and 43.9 % negative examples for $\mathcal{L}_1$, while getting 78.1 % positive and 21.9 % negative example terms for $\mathcal{L}_2$.

$\mathcal{T}[x,y]$	Ø depth	Ø size	term space	largest class (%)	
				$\mathcal{L}_1$	$\mathcal{L}_2$
6,13	5.42	12.49	1.6×10^4	56.1	78.1
7,15	6.15	14.55	1.1×10^5	55.1	80.2
8,20	7.04	18.60	4.7×10^6	56.2	86.5
10,21	7.90	20.66	3.9×10^7	53.8	85.1
11,22	8.03	20.72	4.0×10^7	54.7	85.5
12,24	8.45	22.68	2.8×10^8	59.2	87.1

Table 5: Characteristics of the generated term data sets.

The symbols from $\Sigma = \{f, a, b\}$ are coded in a one-of-three style, i.e. the mapping is $c : \Sigma \to \{-1, +1\}^3$. The used architectures can be specified by FA($r = 1, s = 1, m, n = 3, k = 2, q = 1, \nu_1, \sigma_c$) where ν_1 stands for first-order connections, σ_c the classical sigmoid transfer function $\tanh : I\!R \to]-1, +1[$ and the size of the representation layer is varied across $m = 3, 4, 9, 15$.

The training procedure is backpropagation through structure (BPTS) (an algorithmic form of steepest descent, see Küchler and Goller [46, 26]) augmented by an automatic learning rate adaption and weight decay scheme similar to Rprop (Riedmiller [58]). The training process was carried out in batch modus (weight update after the presentation of all terms in the training set, counts as one epoch) and stopped after 2000 epochs. All results reported here have been obtained by a 10-fold stratified cross-validation (e.g. see Kohavi [41]) on $\mathcal{T}[6, 13]$ with 3 random (uniform distribution from $[-1.0, +1.0]$) weight initializations per fold yielding 30 distinct experiments for each language.

Each of the 30 architectures trained on $\mathcal{T}[6, 13]$ has been tested on the six data sets of higher term complexity in order to estimate the generalization performance. A weak recognition criterion was used: A term was counted as correctly classified if the absolute value of the difference between membership value and output of the network was less than 1.

8.3 Results

Table 6 shows the training performance on $\mathcal{T}[6, 13]$ (the row denoted with 'train') and the generalization accuracy on the six test sets (the other rows, for the characteristics see Table 5) obtained for $\mathcal{L}_1$, $\mathcal{L}_2$ with folding architectures of different number of representation units $m = 3, 4, 9, 15$.

$\mathcal{T}$	$m=3$			$m=4$			$m=9$			$m=15$		
$[x,y]$	Ø	±	max	Ø	±	max	Ø	±	max	Ø	±	max
						$\mathcal{L}_1$						
train	96.17	7.19	100.00	100.00	0.00	100.00	100.00	0.00	100.00	100.00	0.00	100.00
6,13	96.09	7.24	100.00	99.99	0.04	100.00	100.00	0.00	100.00	100.00	0.00	100.00
7,15	93.78	8.03	100.00	99.91	0.17	100.00	99.96	0.13	100.00	100.00	0.00	100.00
8,20	87.98	11.14	100.00	98.92	1.73	100.00	99.62	0.90	100.00	99.96	0.11	100.00
10,21	86.36	11.17	99.20	97.94	3.28	100.00	99.19	1.76	100.00	99.96	0.10	100.00
11,22	84.99	11.68	99.60	97.90	3.14	100.00	99.25	1.68	100.00	99.90	0.16	100.00
12,24	81.46	12.71	99.30	96.46	4.98	100.00	98.64	2.75	100.00	99.80	0.36	100.00
						$\mathcal{L}_2$						
train	87.03	7.68	98.78	94.67	3.89	98.56	99.88	0.19	100.00	99.69	0.73	100.00
6,13	86.90	7.70	98.60	94.41	4.05	98.50	99.70	0.32	100.00	99.19	1.19	100.00
7,15	87.06	6.36	97.90	91.79	4.62	97.60	96.99	1.61	99.30	93.05	6.47	98.90
8,20	89.21	4.05	97.70	91.18	3.58	97.30	95.15	2.04	99.50	90.83	6.14	97.60
10,21	87.77	4.62	96.70	90.57	4.33	98.00	94.77	2.29	99.20	89.03	7.17	97.70
11,22	87.91	4.24	97.30	90.14	4.13	97.80	94.49	2.60	99.30	88.66	7.24	98.10
12,24	88.80	4.29	97.00	90.87	3.83	98.40	94.69	2.42	99.00	88.17	7.35	97.70

Table 6: Generalization capabilities of the FA, learning tasks $\mathcal{L}_1$ and $\mathcal{L}_2$.

For each architecture size and language the mean (Ø), the standard deviation (±) and the maximum value (max) of the recognition performance (in % according to the recognition criterion) over the 30 single experiments is listed.

$\mathcal{L}_1$ The mean recognition rates increase with larger architectures and decrease with tests on term sets of growing term complexity. For the representation layer size $m = 3$ we observe a very large standard deviation caused by a strong sensitivity of the network against different weight initializations. For $m = 4, 9, 15$ the FA achieved a fairly good mean recognition rate close to 100 %. Nevertheless there exists at least one experiment (one trained architecture) which gives a perfect recognition of the language with a mean square error very close to zero (these values are not reported here) consistently throughout the six test sets. During the 2000 epochs training and testing on different partitions of $\mathcal{T}[6,13]$ no over-fitting effects were detected (neither in the case of $\mathcal{L}_2$).

$\mathcal{L}_2$ With few exceptions the larger the term complexity in the test set the worse is the recognition rate. The performance is consistently worse than in the case of $\mathcal{L}_1$, but nevertheless there is at least one experiment which yields a rate above 95 % (99 % for $m = 9$) throughout the six test sets. In contrast to $\mathcal{L}_1$ the best performance can be observed at $m = 9$, while $m = 15$ gives no better result.

Remarks The last observation might be origined in the fact that there are too few training samples (900, remind the 10-fold cross-validation scheme) compared to the number of *programmable parameters* (weights and thresholds) in our learning system. The FA considered here ($m = 15$) counts $(3 + 2 \times 15) \times 15 + 15 \times 1 + 15 + 1 = 526$ programmable parameters. This conjecture is consistent with theoretical results on the *sample complexity* of the FA (see Section 4.5).

Although there is not an exact match between VC-dimension and sample complexity in the context of neural network learning and although there is always a theoretical-practical gap, Theorem 2 ($O(u^2w^4)$ as an upper bound for the VC-dimension of the FA, where w is the number of weights and u the size of the input trees) gives some evidence and plausibility for the above conjecture for the decrease of performance of the FA from $n = 9$ to $n = 15$ in the case of $\mathcal{L}_2$.

8.4 Implications and Conjectures about the Full Computational Power of the FA

The experiments on the two simple IGI tasks shown in the previous section can only be regarded as illustration of the basic procedure. However, the theoretical considerations presented in this paper together with the empirical results gathered by applying the FA to IGI tasks on pattern languages (reported elsewhere, see Goller [25], Schmitt [60], Küchler and Goller [46], Calmbach [8]) motivates the formulation of the following conjectures on the computational power of the FA which might guide further theoretical research and empirical investigations.

For each conjecture stated we try to collect evidence and arguments for its plausibility.

Conjecture 1 (Computational Power) *The FA equipped with sigmoid transfer functions is strictly more powerful than DTA, i.e. there are tree languages that are not in RTL but can be recognized/represented by the FA (assuming infinite arithmetic precision).*

The empirical results show that the FA performs surprisingly (with a generalization accuracy near to 100 %) well on IGI tasks over non-linear pattern languages from $BOL \cap \overline{LOL}$ ($BIL \cap \overline{LIL}$) and other languages beyond RTL. However, this can only be regarded as support but not as proof. As illustrated (in Section 8.3) by the experiment on the IGI of the language $\mathcal{L}_2$ one can test the accuracy only on a finite number of trees of a finite size.

Recently, the question in the case of word languages, whether RNN with sigmoid transfer functions are strictly more powerful than FSA has been answered positive. Hölldobler et al. [33] showed how a very simple RNN can be implemented to recognize the context-free language a^nb^n for any n if infinite arithmetic precision is assumed. The solution has been analytically derived by an analysis of the non-linear dynamics in the state space of the network. (see also comments on Conjecture 2).

The proposed RNN architecture may be viewed as a "trivial" folding architecture of the type FA$(1, 1, 1, 1, 1, 2, \nu_1, \sigma_c)$. Let $\Sigma = \{a, b\}$ be a ranked alphabet with $r(b) = 1$ and $r(a) = \{0, 1\}$. Obviously, the pathological tree language

$$\mathcal{L} = \{\underbrace{b(b(\ldots b}_{n}\underbrace{(a(a(\ldots a(a}_{n})\ldots)))\ldots)) \mid n \geq 1\}$$

is beyond the class RTL (this can be easily proven by applying the *pumping lemma* for regular tree languages, see Gécseg and Steinby [20]). The FA recognizes $\mathcal{L}$ by choosing the parameters (thresholds, weights, input and output encodings) exactly as given by Hölldobler et al. [33]. The original language is "mirrored" and the initial state of the RNN has to be chosen as encoding for the empty tree *nil* since the processing mode in the FA is directed from the leaves to the root.

Further, it is known that a RNN (equivalent to a FA with $k = 1$) can simulate a real-time n-stack Turing machine (Siegelmann and Sontag [66]). Thus, it seems very likely that one can build a FA that has the capability to recognizes even non-pathological tree languages beyond RTL.

Conjecture 2 (Adequacy of the Automata Framework) *The classical formal language theory (Chomsky hierarchy, automata) is not the adequate tool to analyze and understand the behavior and "full" computational power of the analog device FA exhibiting potentially infinite states.*

For the threshold case there is the exact equivalence between FA and DTA. Empirical experiences applying the FA with sigmoid transfer functions (and gradient-based learning procedures like BPTS) to IGI tasks on tree pattern languages consistently show that languages belonging to the class RTL are "easier" (in terms of convergence speed and absolute generalization accuracy) to learn than those beyond RTL (see also the remarks at the end of this section).

However, there is no guarantee that the result of a learning process yields a "neural" DTA implementation (in the sense of the FA construction schemes presented in Section 6). It is unlikely that a FA instance trained to recognize the pattern language $\mathcal{L}_1$ will behave like a corresponding DTA recognizer. Neither means the representability of one language example beyond the class RTL that we will be able to identify a class out of the classical formal automata framework that characterize exactly the computational power of the FA.

Things become more difficult if we allow the nodes of trees from the domain being labeled by continuous-valued vectors from $I\!R^n$. In that case the formal language framework might at most serve as a discrete approximation to a completely continuous phenomenon.

Nevertheless, in the sigmoid case the automata framework can be used to explore the "lower limits" of the principled capabilities of the FA. The membership of a given tree language to the class RTL seems to be an indicator that the IGI problem can be solved by the FA with high accuracy. The automata view may also be of benefit when it can be guaranteed that a DTA is injected directly into the FA (e.g. according to the proof construction for Theorem 4) and the hypothesis space is explicitly constrained to automata during the training process (see Section 9).

Recently, discrete-time recurrent neural networks (RNN) were viewed from a *non-linear dynamical systems* perspective (e.g. see Kolen [43], Casey [9], Tino et al. [75], Tino [74]). Wiles and Elman [80] demonstrated that a simple recurrent network architecture can be trained to perform an example of a context-free language prediction task. They discovered that this phenomenon can be perfectly explained by different dynamical regimes interacting in the state space of the network. Steijvers and Grünwald [71] were even able to tune (by simulation) few parameters of a recurrent network such it can accomplish a context-sensitive language prediction task. They showed that the dynamics of the network can be controlled in a way that regions of the state space that correspond to the symbols to be predicted are linearly separable. Hölldobler et al. [33] (see the discussion of Conjecture 1 above) were able to analytically derive a RNN that recognizes the context-free language $a^n b^n$. They utilized the network as something like a "continuous counter" in the state space, the dynamics is controlled in a way that the presentation of the symbol a triggers the increment and the symbol b the decrement operation.

For discrete-time recurrent neural networks the term "time" is defined as a linear point structure, i.e. as a set of discrete time points constrained by a strict partial ordering, continuous, left-bounded and linear. If we want to apply the dynamical system point of view to the FA we have to deal with a *non-linear time ontology.* Let us reconsider and adapt the description of the processing dynamics of the FA (see Section 4.3, Equation 2). The state $\tilde{g}(t)$ of the system at time t is computed by

$$\tilde{g}(t) = g\big(I(t) \oplus \tilde{g}(t_1) \oplus \tilde{g}(t_2) \oplus \cdots \oplus \tilde{g}(t_k)\big),$$

where $t > t_i$ $(i = 1, 2, \ldots, k)$, the function $g : I\!R^{n+k\cdot m} \to I\!R^m$ is constituted by the folding part and $I(t)$ is the external input (in our case the coding of a symbol from the alphabet) at time t. The output $\Xi(t)$ of the system at time t is determined by:

$$\Xi(t) = h(\tilde{g}(t)),$$

where the function $h : \mathbb{R}^m \to \mathbb{R}^q$ is given by the transformation part.

Thus, the state of the system depends on a sequence of k prior states. In contrary to *RNN* the concept "time" in the FA dynamics is defined by a *right-linear* ordering and a k-fold *branching past* (see Figure 7). Clearly, these concepts are equivalent for $k = 1$.

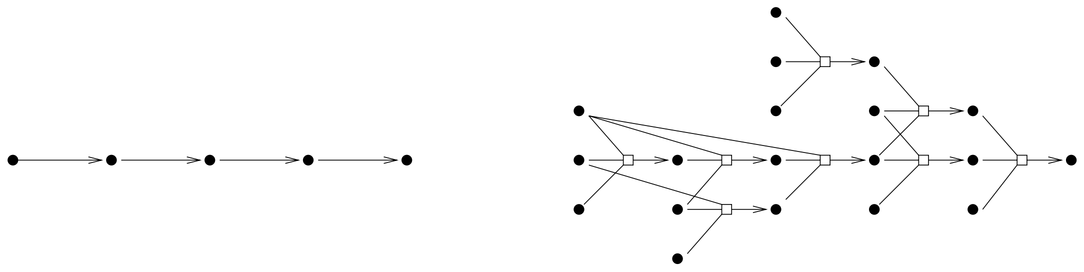

Figure 7: Sketch of two different ontologies in discrete time. On the left: concept of time exhibited by the dynamics of a RNN. On the right: a three-fold branching past induced by the FA ($k = 3$) dynamics. Time points are denoted by black bullets, arrows illustrate the *prior-to* relation between the left and the right time point.

The dynamical system point of view might be fruitful (and more adequate than the formal automata theory) to analyze the behavior of architectures like the FA. However, in order apply tools and concepts from the non-linear dynamical systems theory one has first to ask the question whether and how these can be lifted from "linear" to the "branching past" dynamics.

Remarks Although not explicitly addressed in this paper, it may be clear that questions on the *learnability* of tree languages are much harder to answer than questions about the *representation*. Even the formulation of an adequate definition of "learnability" is a non-trivial problem. The classical concepts known from the computational learning theory like *learning in the limit* from positive and negative examples (Gold [24]) or the *probably approximately correct identification (PAC)* model (Valiant [78]) do not fit very well to the continuous optimization process carried out during the training of the FA, to the convergence criterion and the quality of the final solution.

Further, the gradient-based optimization algorithms are known to get stuck into local minima; the absence from local minima and thereby convergence to the global minimum can only be guaranteed in trivial cases (Frasconi et al. [19]).

The *model selection* problem, i.e. the question about the right scaling of the folding architecture (number of layers, number of neurons in each layer, dimension m of the representation layer) can be only partially answered. The constraints on the FA (derived in Section 6) to achieve the computational power of DTA give only very rough decision factors and assume a priori knowledge about the class of languages to be presented. If one knows the language to be learned in advance one might use some good heuristics to estimate the number of neurons required (see Siegelmann and Giles [65] in the case of regular languages, FSA and RNN). Another alternative to get rid with the model selection problem might be in the use of constructive learning algorithms, i.e. algorithms which lead to an incremental growth of the network if the current one cannot solve the problem accurately. Sperduti [70] shows that techniques like *cascade-correlation* can be extended to network models like FA. However, special care has to be taken (see Giles et al. [22] to conserve the computational power.

The empirical results sketch a much more friendlier picture (see Goller [25], Schmitt [60], Küchler and Goller [46], Calmbach [8]). IGI tasks on languages beyond RTL were solved by the FA (using gradient-based learning algorithms) with fairly high generalization accuracy. We have the conjecture that even if the concept to be learned is beyond the representational capability of the FA (or beyond that of a given finite-size instance of the FA) it is likely that the concept can be approximated well (for trees of bounded size) by the restricted means.

9 Remarks on Combining Symbolic and Neural Learning

The previous section indicate that the FA can be used as a "neural learning device" to inductively acquire automata behavior given positive and negative examples for of the corresponding language. On the other side there are several "symbolic approaches" to solve the inductive grammatical inference task for tree languages resp. tree automata (see Knuutila [40], a collection of further references can be found in Miclet [50], Schalkhoff [59] and Steinby [72]). The metaphor applied in the later case is the *hypothesis space* spread out by all automata instances of a given language class. IGI means a clever navigation through this space in order to find the best hypothesis consistent with the given training data. Each of both approaches is characterized by its own merits and disadvantages.

9.1 The Injection-Refinement-Extraction Framework

Recently, a general framework has been identified in order to combine the strength of symbolic with those of neural learning (Shavlik [63]). It can be roughly outlined as constituted by the following three phases:

$$(1)\text{ Injection} \longrightarrow (2)\text{ Refinement} \longrightarrow (3)\text{ Extraction}$$

(1) Symbolic knowledge prior available about the application domain is given by a human expert or/and acquired by a "symbolic learning device". The ideal notion is that this knowledge does not necessarily has to be a complete, minimal and perfect solution of the problem, but only a rough approximation thereof. The next step is to transform (inject) it into an equivalent neural network.

(2) Then possibly some additional degrees of freedom are added to the network architecture. Further, the network has to be prepared to become trainable by an available neural training procedure. Now, known (and additional) training data is used to refine/improve the first approximative solution.

(3) Ideally the trained network represents a better solution of the initial problem. But one does not want to accept the network as a "black-box" result, the objective rather is to extract it into a "comprehensible" symbolic way (e.g. in the same language the prior knowledge was given).

In spite of some principled problems within this framework (not discussed here, see Shavlik [63]) several researchers were able to successfully apply this idea and empirically demonstrated that this combination of symbolic and neural approaches yields better accuracy than each individual. Towell and Shavlik [77] instantiated this framework into a method to transform symbolic knowledge (in form of propositional Horn logic) given by human experts into a special feedforward network architecture, which was subsequently trained. Knowledge extraction is done by an algorithm that produces so called "N-of-M" rules. Ivanova and Kubat [37] fruitfully combined the well-known symbolic decision tree learning approach with refinement of the acquired knowledge by a multilayer network and a standard neural training procedure.

Similar strategies were also applied in the context of FSA (IGI of regular languages) and RNN (e.g. see Omlin and Giles [55], Frasconi et al. [16]). However, special measures have to be taken to ensure that not only the automaton behavior but also the structure is preserved during refinement by neural training. This was shown as a necessary condition to have a chance to extract useful "automata knowledge" from the state space information of the network (Kolen [43], Casey [9]).

9.2 Robust Injection of a DTA into the FA

The general combination framework might in principle be also applicable to the context of FA and DTA. Prior knowledge given by experts in form of a DTA (or implicitly in form of a regular tree grammar that has been transformed to a minimal equivalent DTA) or induced by symbolic machine learning approaches (see Knuutila [40]) can directly be injected into an equivalent FA with sigmoid transfer functions.

Assuming an infinite arithmetic precision one can choose among the different possible instances of the FA (first-order vs. higher-order connections, two-layered vs. one-layered folding part) and apply the corresponding construction schemes established in Section 6 (see proof constructions to Theorem 4, Theorem 5 or Theorem 8).

Before refinement (Phase 2 of the framework sketched in Section 9.1) usually some preprocessing steps are carried out (Towell and Shavlik [77], Ivanova and Kubat [37]). The connection scheme between successive layers is extended to a fully-connected one, the weights of *regular* connections (that are required for the construction scheme) and of *irregular* connections (new connections with weights that were formerly set to zero) are slightly perturbed by small random values to provide a suitable initialization for subsequent refinement by a numerical training (optimization) algorithm.

Further, for practical implementations (specially on digital parallel hardware) one can only rely on a finite precision of the number representation and the arithmetic operations.

Therefore we will investigate whether and how the FA construction schemes given in Section 6 can be adapted to cope with irregular connections based on the following noisy neuron model.

Definition 10 (Noisy Neuron Model) Let o_i be the output of the neuron indexed by the number i with first-order activation function ν_1 and the classical sigmoid function σ_c, θ_i be the bias and $o_j, 1 \leq j \leq n+m$ be the outputs from neurons connected to by n regular and m irregular connections with weight w_{ij}. Further let $\Delta_\tau \in [-\tau, +\tau], \Delta_\rho \in [-\rho, +\rho]$ and $\Delta_\pi \in [-\pi, +\pi]$ be random variables (under arbitrary distributions) with $\tau, \rho, \pi \in \mathbb{R}$. The output o_i is computed according to

$$o_i \quad = \quad \sigma_c \left(\sum_{j=1}^{n+m} (w_{ij} + \Delta_\tau)\, o_j + (\theta_i + \Delta_\tau) + \Delta_\rho \right) + \Delta_\pi$$

where Δ_τ represents noise on the weights, Δ_π noise on the output and the activation is corrupted by Δ_ρ.

Again, we start with the neuron as the basic building block of the FA. The next Lemma shows that the simulation of Boolean gates (see also Section 5) by a single sigmoid neuron in a noisy environment (as specified by Definition 10) can only be accomplished by restricting the amount of noise Δ_π on the output. The weights can be tuned to tolerate any given (but a priori known) amount of noise Δ_τ on the weights and noise Δ_ρ on the activation.

Lemma 13 *A single neuron provided with first-order connections and the classical sigmoid transfer function σ_c can simulate*

a. *The n-ary Boolean $\wedge$,*

b. *the n-ary Boolean $\vee$,*

obeying the following additional constraints:

1. *There are n regular (for the n Boolean arguments) and m irregular incoming connections with weights w and the bias θ which are allowed to be perturbed by given noise $\Delta_\tau \in [-\tau, +\tau]$.*
2. *The activation of the neuron is corrupted by given noise $\Delta_\rho \in [-\rho, +\rho]$.*
3. *The noise $\Delta_\pi \in [-\pi, +\pi]$ tolerated on the output (input) is restricted to $0 < \pi < \varepsilon < \frac{1}{2n}$ where ε is the discretization parameter applied to map the continuous output to a Boolean value.*

Proof We apply the same discretization scheme as given by the proof of Lemma 1. Since σ_c has a continuous range of $]0,1[$ we choose an $\varepsilon \in [0, \frac{1}{2}[$ and interpret the output of a neuron in the following way. We say that (the output of) a neuron is *high* if $\sigma(x + \Delta_\rho) + \Delta_\pi \geq 1 - \varepsilon$, *low* if $\sigma(x + \Delta_\rho) + \Delta_\pi \leq \varepsilon$ and undefined, else. Thus, the Boolean values $\{0, 1\}$ correspond to $\{low, high\}$. Let $\psi = \rho + \ln \frac{1-\varepsilon+\pi}{\varepsilon-\pi}$ with $\pi < \varepsilon$. Due to the monotonicity and symmetry of σ_c the output will be high if the activation of the neuron is $x \geq \psi$, low if $x \leq -\psi$ and undefined, else.

a. A neuron receiving input from n neurons connected by identical weights $w \in I\!R, w > 0$ (possibly perturbed by noise Δ_τ) and from m irregular connections (with weights arbitrarily distributed in $[-\tau, +\tau]$) behaves as a n-ary Boolean $\wedge$ iff the following constraints can be satisfied:

$$\begin{aligned} (1-\varepsilon) \cdot n \cdot (w - \tau) - m \cdot (1+\pi) \cdot \tau + \theta - \tau &\geq \psi \\ (1+\pi) \cdot (n-1) \cdot (w+\tau) + \varepsilon \cdot 1 \cdot (w + \tau) + m \cdot (1+\pi)\tau + \theta + \tau &\leq -\psi \end{aligned}$$

By turning these constraints into equations we get at least one solution

$$\begin{aligned} w &= \frac{-2\,\psi + [-1 - 2\,m + \varepsilon\,(-1+n) - 2\,n + (1 - 2\,m - n)\,\pi]\,\tau}{-1 + \varepsilon\,(1+n) + (-1+n)\,\pi} \\ \theta &= \frac{[\varepsilon + (1-\varepsilon)\,n + (-1+n)\,(1+\pi)]\,\psi + \theta_\wedge(n, m, \varepsilon, \tau, \pi)}{-1 + \varepsilon\,(1+n) + (-1+n)\,\pi} \end{aligned}$$

where $\theta_\wedge(n, m, \varepsilon, \tau, \pi)$ is a polynomial of order 2 and ε, π are constrained by

$$0 < \pi < \varepsilon < \frac{1}{2n}$$

b. For simulating a n-ary Boolean $\vee$ we get the following conditions ($w > 0$):

$$\begin{aligned}(1-\varepsilon)\cdot 1\cdot(w-\tau)-\pi\cdot(n-1)\cdot(w-\tau)-m\cdot(1+\pi)\cdot\tau+\theta-\tau &\geq \psi \\ \varepsilon\cdot n\cdot(w+\tau)+m\cdot(1+\pi)\cdot\tau+\theta+\tau &\leq -\psi\end{aligned}$$

resulting in at least one solution

$$\begin{aligned} w &= \frac{-2\,\psi+[-3-2\,m+\varepsilon\,(1-n)+(-1-2\,m+n)\,\pi]\,\tau}{-1+\varepsilon+\varepsilon\,n-\pi+n\,\pi} \\ \theta &= \frac{[1-\varepsilon+\varepsilon\,n-(-1+n)\,\pi]\,\psi+\theta_\vee(n,m,\varepsilon,\tau,\pi)}{-1+\varepsilon+\varepsilon\,n-\pi+n\,\pi}\end{aligned}$$

where $\theta_\vee(n,m,\varepsilon,\tau,\pi)$ is a polynomial of order 2 and ε,π are constrained by

$$0<\pi<\varepsilon<\frac{1}{2n}$$

■

For example a Boolean $\vee$ with $n = 3$ regular and $m = 2$ irregular connections and the noise levels $\tau = \rho = \frac{1}{2^8}$ can be implemented in a robust way choosing $\varepsilon = 0.1$, $\pi = \frac{1}{2^8}$ and calculating $w = 7.63064$ and $\theta = -4.54742$ according to the proof of Lemma 13. Again, the single neuron can be used as basic building block to implement any given DTA by the folding architecture.

Theorem 15 (Robust Injection) *The FA with first-order connections and the classical sigmoid transfer function σ_c can simulate any DTA using two layers in the folding part and only one unit in the transformation part (layers are fully-connected). This implementation is robust (w.r.t. the noise model specified in Definition 10) if the noise level on the output of each neuron is bounded by the discretization parameter.*

Proof (Robust Injection) Take the building plan from the proof of Theorem 4. Let n' be the maximum fan-in among all neurons, $\varepsilon = \frac{1}{2n'+1}$ be the discretization parameter and $[-\tau,+\tau],[-\rho,+\rho],[-\pi,+\pi] \subset I\!R$ three given intervals with $\pi < \varepsilon$. Insert new (*irregular*) connections so that all layers are fully connected and initialize them with zero. Regular links are weighted and the thresholds are set according to the proof of Lemma 13.

Lemma 13 guarantees the injection of the DTA to be robust against arbitrarily distributed noise $\Delta_\tau \in [-\tau,+\tau]$ on the weights, noise $\Delta_\rho \in [-\rho,+\rho]$ on the activation and noise $\Delta_\pi \in [-\pi,+\pi]$ on the outputs of the neurons. ■

Remarks Maass and Sontag [49] showed (using a noise model slightly different to that specified by Definition 10) that discrete-time recurrent neural networks with noise (under very mild assumptions) on the output of neurons can only recognize a subclass of regular languages. As demonstrated by Theorem 15 our construction for a DTA injection can be tuned to tolerate a very small noise level on the outputs of neurons.

The proof construction (belonging to Theorem 15) gives us a scheme for the injection of an arbitrary DTA into an equivalent FA. The addition of irregular links and a random perturbation of the weights within the τ-interval prepares the neural network for the refinement phase. Thus, standard training algorithms like BPTS (Küchler and Goller [46]) can be applied.

For the refinement and extraction phase concepts and methods might be borrowed from existing knowledge about the instantiation of the framework to FSA and RNN (e.g. see Omlin and Giles [55], Alquézar and Sanfeliu [2] Tino and Sajda [76], Frasconi et al. [16]). If one wants to also extract knowledge in "automata form" special measures – e.g. by regularization techniques (Frasconi et al. [17]) or by explicitly constraining the weight space to feasible regions (Frasconi et al. [16]) – have to be taken to prevent the dynamical system from leaving the automata hypothesis space during retraining (compare also Section 8.4, Conjecture 2).

The extraction of FSA from successfully trained discrete-time recurrent networks is usually accomplished by a discretization and exploration of the state space (e.g. see Omlin and Giles [56], Tino and Sajda [76], Frasconi et al. [17]).

To avoid the combinatorial explosion of the exploration process (imposed by the tree domain $\mathcal{T}(\Sigma)$ in the case of DTA and FA) it might be a good idea to navigate through the state space using the known training resp. test data or only a small fraction of the tree domain.

10 Related Work

The correspondence between special recurrent neural network architectures and tree automata was first shown by Sperduti [70, 69]. One layer equipped with so-called *first-order generalized recursive* neurons and threshold transfer functions has been proven to have the power to simulate any deterministic bottom-up tree automaton. Without explicitly referencing the *state-splitting* approach of Goudreau et al. [28] and without explicitly identifying neurons as Boolean gates a construction scheme was developed for the case of threshold functions. For the case of sigmoid transfer functions the approximation (of threshold functions by sigmoid ones) argument is supplied.

Generalized recursive neurons implanted into Elman-style networks are a special case of the folding architecture, i.e. are equivalent to FA with one layer as the folding and one as the transformation part. Obviously, the mentioned result of Sperduti [70] is a special case of Theorem 8 and its proof construction (for verification one can substitute the sigmoid neurons by the threshold gate neurons given in Lemma 4). The discretization scheme used here (see Lemma 1, proof construction) allows (contrary to Sperduti [70]) the exact simulation of any DTA by a FA with sigmoid transfer functions. As shown in Section 9 our constructions can be extended to a DTA injection scheme which is robust against a small amount of noise (on the weights and the output of neurons) and that is ready for subsequent refinement.

The recipe for simulating Boolean functions by single neurons using various transfer functions was partially taken from Ivanova and Kubat [37], extended and adapted for our purposes (see Section 5). This abstraction technique – using neurons which simulate Boolean functions as primitive building block for complex network architectures – may be found in several similar contexts (e.g. see Frasconi et al. [16] or Towell and Shavlik [77]).

Our investigations are driven by the observation that folding architectures limited to the arity $k = 1$ are equivalent to the class of multilayer discrete-time recursive networks used for sequence processing (Küchler and Goller [46], Giles et al. [23]). Thus, most of the argumentations, theorems and proof techniques presented in Section 6 are directly lifted from the excellent framework developed by Kremer [45, 44] (for the correspondence of different discrete-time recurrent network architectures to the formal language theory) to different instances of the FA and their correspondence to tree automata.

New insights for discrete-time recurrent networks are obtained by specializing the FA to the fixed arity $k = 1$. The proof construction to Theorem 4 gives a (to our knowledge) novel scheme for injecting FSA into first-order discrete-time recurrent networks. In contrary to the state-splitting technique (Kremer [44], Goudreau et al. [28]) to implant a FSA into one-layer Elman-style networks (Elman [13]) our construction requires only a moderate resource (in terms of the number of neurons and weights needed, see Section 6) consumption which makes it attractive for practical applications.

The theorems and proofs concerning the node complexity of FA implementations of DTA are directly transferred from the work of Horne and Hush [36, 35]. Setting the maximum arity k to $k = 1$ and the cardinality n of the automaton alphabet to $n = 2$ yields exactly their upper bound results for discrete-time recurrent networks with transfer functions. The proof technique for the lower bound is due to Alon et al. [1]. However, we were not able to lift their construction for a lower bound on the number of divergent mealy machines to the DTA case.

Various research on the computational power of discrete-time recurrent neural network models can be found in literature. Siegelmann and Sontag [66] demonstrated the Super-Turing power (real-time processing mode, with a four-step slowdown) for first-order single-layer recurrent neural networks with saturated linear transfer functions by a constructive proof using an a priori fixed number of neurons. Later, Kilian and Siegelmann [39] provided a proof for the Turing universality in the case of sigmoid transfer functions. Assuming real-time processing the Turing universality also holds for the FA.

There are several attempts to represent and to learn examples of context-free languages with discrete-time recurrent neurons network models. Das et al. [11] and Zeng et al. [82] study recurrent architectures which are augmented by an external (continuous) stack. Internal signals to control the stack operations are implicitly included into the given learning task and the input is still represented as sequence. The FA directly operates on structured objects (trees), structured expressions as usually described by CFL can be represented in a “natural” way. However, external memory (for the storage of at least k prior state representations) is also required for the operation of a trained FA (see Section 4).

Wiles and Elman [80] were able to train a simple discrete-time recurrent neural network to accomplish a beyond-regular-language (CFL) prediction task. The underlying mechanisms could be explained by taking the dynamical systems point of view. Steijvers and Grünwald [71] followed this idea idea and showed that even examples of languages beyond the context-free border can be represented by simple discrete-time recurrent architectures. The same phenomena can be observed in the context of the FA. The arguments and empirical results presented in Section 8 (and other experimental results reported elsewhere [46, 62]) give strong evidence that there are languages beyond the regular tree language class which can be represented by a FA (and for which the FA can be effectively and efficiently trained to behave as recognizer). This conjecture is also supported by two computation models related to discrete-time recurrent neural networks that recently emerged. Moore [52] presented classes of *iterated functions systems* that are (in the case of piecewise linear maps and beyond) strictly more powerful than deterministic real-time pushdown automata. Tabor [73] specified classes of so-called *pushdown dynamical automata* and metrics on its members that can be constrained to be equivalent to classical pushdown automata.

The application of the known framework (see Shavlik [63]) for the combination of symbolic and neural learning to DTA and FA is essentially inspired by the work of Ivanova and Kubat [37] on combining decision tree induction with neural learning. Decision trees (induced by classical machine learning approaches) are interpreted as Boolean expressions, transformed into disjunctive normal form and directly transferred into an equivalent three-layered feedforward network architecture. Each Boolean gate is simulated by a sigmoid neuron. Before refinement new degrees of freedom are added to the architecture, weights are slightly perturbed. In this paper we followed the same strategy to develop a robust scheme for injecting given DTA into FA (see Section 9 and Theorem 15).

From the research methodology point of view we have seen that many concepts and results known in the field of sequence processing with recurrent neural network models can be fruitfully transferred to structure (tree) processing by the folding architecture. Sperduti [70] proved this way as successful by lifting the well-known *cascaded* recurrent networks and *neural trees* to structure processing. Recently, Frasconi et al. [18] gave a probabilistic framework for the adaptive processing of structures and show how the well-known *hidden markov models* for temporal sequence processing can be lifted to fit into that framework.

11 Conclusions

In this work we explored some obvious correspondences between neural folding architectures and tree automata. Our investigations were guided by two observations. First, both models operate in the same way, i.e. both compute a new state by combining a sequence of prior states with symbols read from the input. Secondly, the known class of discrete-time recurrent neural network models may be viewed as a special case of the FA (by constraining the maximum rank to $k = 1$).

Figure 8 gives an overview over the most important machine models and formal language classes and their mutual relations considered in this article. It is well-known that FSA correspond to regular word languages (REG) which in turn is a strict subclass of the class of context-free languages (CFL) (see Relations 2,14 and Hopcroft and Ullman [34]). Further, tree automata and regular tree languages (RTL) are known to be a straight forward generalization of FSA and REG (see Relations 4,5,7 and Doner [12]). Deterministic bottom-up tree automata (DTA) have the same computational power as the non-deterministic variant and as their non-deterministic top-down counterpart (Gécseg and Steinby [20]). RNN are known to possess at least the computational power of FSA (see Relation 1, and e.g. Kremer [45, 44], Giles [55], Goudreau et al. [28], Minsky [51]).

Here we closed the triangle of Relations 3,1,4 by proving that the FA has at least the computational power of DTA (Relation 6). Several instances of the generic architecture were investigated (see also Table 8). For first-order activation functions at least two layers are required – either two in the folding part or one there and one in the transformation part. By using activation functions of order $k + 1$ (where k is the maximum rank of the given tree automata) a given DTA can be simulated by exactly one layer as folding part. In the case of threshold transfer functions the Relations 1 and 6 are reduced to equivalence.

We investigated the node complexity of DTA implementations for folding architectures with threshold transfer functions. Upper bounds depending on the number of layers in the folding part (see also Table 7) and a "weak" lower bound were derived.

Terms sets extracted from the application field of learning search control functions for symbolic deduction systems and formerly used in experiments to test the representational capabilities of the FA (see Goller [25], Schmitt [60], Küchler and Goller [46]) are identified to belong to the class of tree (term) pattern languages. Pattern occurrence languages (OL) are specified by a term pattern.

$$\begin{tikzcd}
\text{RNN} \arrow[r, dash, "\sqsupset", "1)"'] \arrow[d, dash, "3)"', "\sqcap"] & \text{FSA} \arrow[r, dash, "\overset{\mathcal{L}}{\equiv}", "2)"'] \arrow[d, dash, "4)"', "\sqcap"] & \text{REG} \arrow[r, dash, "\subset", "14)"'] \arrow[d, dash, "5)"', "\cap"] & \text{CFL} \\
\text{FA} \arrow[r, dash, "\sqsupset\ ?", "6)"'] \arrow[dr, dash, "?", "8)"'] & \text{DTA} \arrow[r, dash, "\overset{\mathcal{L}}{\equiv}", "7)"'] & \text{RTL} \arrow[d, dash, "9)"', "\cup"] & \\
 & \text{BOL} \arrow[r, dash, "\supset", "10)"'] \arrow[d, dash, "11)"', "\cup"] & \text{LBOL} \arrow[d, dash, "12)"', "\cup"] & \\
 & \text{OL} \arrow[r, dash, "\supset", "13)"'] & \text{LOL} &
\end{tikzcd}$$

Figure 8: Visualization of the basic relations between different machine models and language classes (denoted by solid lines and labeled by numbers to facilitate referencing from the text). The result that one machine is strictly more powerful than the other is expressed by "$\sqsupset$". "$\overset{\mathcal{L}}{\equiv}$" denotes the correspondence between a machine and its recognizable language class, "$\subset$" is the usual strict subset relation. "?" indicate interesting open questions. Implicitly, the usage of the sigmoid transfer function is assumed for RNN and FA.

A given term is in a given pattern language iff the specified pattern occurs as subterm. Linear pattern languages (LOL) are defined by patterns where each variable occurs not more than once (see Relation 13,10). The expressiveness of pattern languages can be enhanced by allowing a finite Boolean combination occurrence constraints (BOL and LBOL, Relations 11,12). Linear pattern languages (like LBOL and LOL) were shown to be in RTL while the non-linear pattern languages (like BOL and OL) are beyond RTL (Relation 9). Thus, by the chain of Relations 6,7,9 the FA has at least the representational capability of linear pattern languages.

Recently, Relation 1 has been proven to be strict (see Hölldobler et al. [33]) by constructing a RNN that is capable to recognize the context-free language $a^n b^n$ (assuming infinite precision representation of the arithmetic operations and the weights). This language can be taken as a pathological example of a language beyond RTL that is recognizable by a trivial FA. We conject that Relation 6 is strict also for non-trivial tree languages. Empirical results show that inductive inference tasks on non-linear pattern languages (which are beyond RTL) can be solved by the FA with a high generalization accuracy. This may be counted as support for this conjecture. A further interesting open question is in the relationship between the FA and the non-linear pattern languages BOL and OL (Relation 8). What is and how to describe the "full" computational power of the generic architecture that allows a discrete-time analog computation in a potentially infinite state space? The dynamical systems point of view (see e.g. Blair and Pollack [7], Casey [9], Kolen [43], Moore [52], Tabor [73]) might be helpful in finding answers.

The theoretical results presented here together with former empirical results on artificial data

FA(r,2,$\lceil\log m\rceil$, $\lceil\log n\rceil$,k,1,ν_1,σ_t)			
$r=2$	$r=3$	$r=4$	r arbitrary
$O(nm^k2^k)$	$O(\log m\sqrt{nm^k2^k})$	$O\left(\sqrt{\frac{nm^k2^k\log m}{\log n+k\log m}}\right)$	$\Omega(\sqrt{\frac{nm^k}{\log n+k\log m}})$

Table 7: Bounds on the node complexity of DTA in FA (first-order activation function ν_1, threshold transfer unction σ_t) implementations using r layers in the folding part. The complexity is measured by the number of states m, the size of the alphabet n with maximum rank k of the given DTA.

FA			neurons	weights	weights
r	s	ν		(full connectivity)	(non-zero)
2	0	ν_1	$O(m^kn)$	$O(km^{k+1}n+m^kn^2)$	$O(km^kn)$
1	1	ν_1	$O(m^kn)$	$O(km^{2k}n^2)$	$O(km^{2k-1}n^2)$
1	0	ν_h	$O(m)$	$O(n(m+1)^{(k+1)})$	$O(m^kn)$

Table 8: Comparison of different tree automata injection schemes in terms of their space complexity. The three folding architectures are differentiated from the type of activation functions (ν_1, first-order vs. ν_h, higher-order) and the number of layers r and s required in the folding and transformation part. The resource consumption for the injection of a given DTA $\mathcal{A}=(\Sigma,Q,F,R)$ is characterized by the number of automata states $m=|Q|$, the size of the alphabet $n=|\Sigma|$ and the maximum rank k of Σ and is measured by the number of neurons and number of weights (connections) required in the folding architecture. Here, only the case of a complete specification of nm^k automata transitions is shown. The sixth column counts the number of connections with non-zero weights while a full connectivity is assumed in column five.

give some practical hints for the application of the FA in inductive inference tasks. The FA together with appropriate training algorithms is a promising candidate in a scenario where objects can be adequately represented by rooted labeled ordered trees, the size of the representations cannot be fixed a priori, there is enough training data available and structural concepts have to be captured in order to solve the task (but unknown in advance). Concepts belonging to regular tree languages can be represented by the FA. One has several design alternatives concerning the layout of the architecture, we give some minimum (in terms of layers used in the folding respectively in the transformation part) instances that guarantee the representational capapability of RTL.

The constructive proofs give effective knowledge injection schemes (see also Table 8) that can be made robust against weight perturbations and tolerates a small noise level on the output of neurons. Especially the Boolean combination of pattern occurrence languages seems to be an useful formalism to describe knowledge partially available in a pattern recognition domain. However, the embedding of the FA into a practical and advantageous injection-refinment-extraction framework has to be evaluated by solid experiments.

The correspondences established between folding architectures and tree automata further show that there are indeed connectionist models that are capable to perform structure-sensitive operations. This property is postulated by some schools of cognitive science for any adequate model describing the higher-level cognitive tasks carried out by the human brain (e.g. see Fodor and Pylyshyn [14], van Gelder [79], Niklasson and Sharkey [54]). However, the FA is far away from any biological reality.

Acknowledgements This research was partially supported by the German Research Community (DFG) under grant no. Pa 268/10-2. I thank Paolo Frasconi, Christoph Goller, Marco Gori, Barbara Hammer, Yvonne Kalinke, Stefan Kremer, Mirek Kubat, Marco Maggini, Günther Palm, Alessandro Sperduti and Ah Chung Tsoi for exciting and fruitful discussions on the topics adressed in this report.

References

[1] Noga Alon, A.K. Dewdney, and Teunis J. Ott. Efficient Simulation of Finite Automata by Neural Nets. *Journal of the ACM*, 38(2):495–514, April 1991.

[2] R. Alquézar and A. Sanfeliu. An Algebraic Framework to Represent Finite State Machines in Single-Layer Recurrent Neural Networks. *Neural Computation*, 7(5):931–949, 1996.

[3] Dana Angluin and Carl H. Smith. Inductive Inference: Theory and Methods. *ACM Computing Surveys*, 15(3):237–269, September 1983.

[4] Martin Anthony. Probabilistic Analysis of Learning in Artificial Neural Networks: The PAC Model and its Variants. *Neural Computing Surveys*, 1:1–47, 1997.

[5] Setsuo Arikawa, Satoru Miyano, Ayumi Shinohara, Saturo Kuhara, Yasuhito Mukouchi, and Takeshi Shinohara. A Machine Discovery from Amino Acid Sequences by Decision Trees over Regular Patterns. *New Generation Computing*, 11(3):361–375, 1993.

[6] Anna Maria Bianucci, Alessio Micheli, Alssandro Sperduti, and Antonina Starita. Quantitative Structure-Activity Relationships of Benzodiazepines by Recursive Cascade Correlation. In *Proceedings of the 1998 IEEE International Joint Conference on Neural Networks*, 1998. to appear.

[7] Alan D. Blair and Jordan B. Pollack. Analysis of Dynamical Recognizers. *Neural Computation*, 9(5):911–926, 1997.

[8] Stefan Calmbach. Die Neuronale Faltungsarchitektur – Erweiterung, Implementierung und empirische Evaluierung am Beispiel der Termersetzung. Masters Thesis, Fakultät für Informatik, Universität Ulm, 1996. in German.

[9] Mike 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.

[10] Hubert Comon, Max Dauchet, Rémi Gilleron, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree Automata Techniques and Applications. ongoing book project, Laboratoire d'Informatique Fondamentale de Lille, 1997.

[11] S. Das, C.L. Giles, and G.Z. Sun. Using Prior Knowledge in a NNPDA to Learn Context-Free Languages. In Stephen José Hanson, Jack D. Cowan Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 65–72. Morgan Kaufmann, San Mateo, 1993.

[12] John Doner. Tree Acceptors and some of their Applications. *Science of Computer Programming*, 4(5):406–451, October 1970.

[13] J.L. Elman. Finding Structure in Time. *Cognitive Science*, 14:179–211, 1990.

[14] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition*, 28:3–71, 1988.

[15] E. Francesconi, P. Frasconi, M. Gori, S. Marinai, J.Q. Sheng, G. Soda, and A. Sperduti. Logo Recognition by Recursive Neural Networks. In *Proceedings of the 2nd IAPR Workshop on Graphics Recognition (GREC'97)*, pages 144–151. Springer, 1997.

[16] P. Frasconi, M Gori, and G. Soda. Recurrent Neural Networks and Prior Knowledge for Sequence Processing: A Constrained Nondeterministic Approach. *Knowledge-Based Systems*, 8(6):313–332, 1995.

[17] Paolo Frasconi, Marco Gori, Marco Maggini, and Giovanni Soda. Representation of Finite State Automata in Recurrent Radial Basis Function Networks. *Machine Learning*, 23(1):5–32, 1996.

[18] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A General Framework for Adaptive Processing of Data Structures. Technical Report DSI-RT-15/97, Università degli Studi di Firenze, Dipartimento di Sistemi e Informatica, April 1997.

[19] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. On the Efficient Classification of Data Structures by Neural Networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, 1997.

[20] Ferenc Gécseg and Magnus Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[21] Ferenc Gésceg and Magnus Steinby. Tree Languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer, 1996.

[22] C.L. Giles, D. Chen, G.Z. Sun, H.H. Chen, Y.C. Lee, and M.W. Goudreau. Constructive Learning of Recurrent Neural Networks: Limitations of Recurrent Cascade Correlation and a Simple Solution. *IEEE Transactions on Neural Networks*, 6(4):829–836, 1995.

[23] C.L. Giles, G.M. Kuhn, and R.J. Williams, editors. *Special Issue on Dynamic Recurrent Neural Networks*, volume 5(2) of *IEEE Transactions on Neural Networks.* 1994.

[24] E. M. Gold. Language Identification in the Limit. *Information and Control*, 10:447–474, 1967.

[25] Christoph Goller. *A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems.* Phd thesis, Technical University Munich, Faculty of Computer Science, 1997. submitted.

[26] Christoph Goller and Andreas Küchler. Learning Task-Dependent Distributed Representations by Backpropagation Through Structure. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'96)*, pages 347–352, 1996.

[27] M. Gori, M. Mozer, A.C. Tsoi, and R.L. Watrous, editors. *Special Issue on Recurrent Neural Networks for Sequence Processing*, volume 15 (3-4) of *Neurocomputing.* June 1997.

[28] M.W. Goudreau, C.L. Giles, S.T. Chakradhar, and D. Chen. First-Order vs. Second-Order Single Layer Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, 5(3):511–515, 1994.

[29] Barbara Hammer. Universal Approximation of Mappings on Structured Objects using the Folding Architecture. Technical Report no. 183, 11/1996, Universität Osnabrück, Fachbereich Informatik, 1996.

[30] Barbara Hammer. Learning Recursive Data is Intractable. Technical Report no. 194, 7/1997, Universität Osnabrück, Fachbereich Informatik, 1997.

[31] Barbara Hammer. A Note on the Number of Divergent Tree Automata, April 1998. personal communication.

[32] Barbara Hammer and Volker Sperschneider. Neural Networks can approximate Mappings on Structured Objects. In *Proceedings of the 2nd International Conference on Computational Intelligence and Neuroscience (ICCIN'97)*, Research Triangle Park, USA, March 1997.

[33] Steffen Hölldobler, Yvonne Kalinke, and Helko Lehmann. Designing a Counter: Another Case Study of Dynamics and Activation Landscapes in Recurrent Networks. In Gerhard Brewka, editor, *KI-97: Advances in Artificial Intelligence*, Lecture Notes in Computer Science (LNCS 1303), pages 313–324, Berlin, 1997. Springer.

[34] J. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Language, and Computation.* Addison–Wesley, Reading, Massachusetts, 1979.

[35] Bill G. Horne and Don R. Hush. On the Node Complexity of Neural Networks. *Neural Networks*, 7(9):1413–1426, 1994.

[36] Bill G. Horne and Don R. Hush. Bounds on the Complexity of Recurrent Neural Network Implementations of Finite State Machines. *Neural Networks*, 9(2):243–252, 1996.

[37] Irena Ivanova and Miroslav Kubat. Initialization of Neural Networks by Means of Decision Trees. *Knowledge-Based Systems*, 8(6):333–344, 1995.

[38] Marek Karpinski and Angus Macintyre. Polynomial Bounds for VC Dimension of Sigmoidal and General Pfaffian Neural Networks. *Journal of Computer and Systems Sciences*, 54(1):169–176, February 1997.

[39] Joe Kilian and Hava T. Siegelmann. The Dynamic Universality of Sigmoidal Neural Networks. *Information and Computation*, (128):48–56, 1996.

[40] Timo Knuutila. *On the Inductive Inference of Regular String and Tree Languages.* PhD thesis, Computer Science, University of Turku, Finland, 1994. also appeared as Research Report R-94-14.

[41] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the forteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*, pages 1137–1145, 1995.

[42] Pascal Koiran and Eduardo D. Sontag. Vapnik-Chervonenkis Dimension of Recurrent Neural Networks. In *Proceedings of the Third European Conference on Computational Learning Theory, 1997*, volume 1208 of *Lecture Notes in Computer Science (LNCS)*, chapter 223–237. 1997.

[43] John F. Kolen. *Exploring the Computational Capabilities of Recurrent Neural Networks.* PhD thesis, Ohio State University, 1994.

[44] Stefan Charles Kremer. On the Computational Power of Elman-Style Recurrent Networks. *IEEE Transactions on Neural Networks*, 6(4):1000–1004, 1995.

[45] Stefan Charles Kremer. *A Theory of Grammatical Induction in the Connectionist Paradigm.* PhD thesis, University of Alberta, Department of Computing Science, 1996.

[46] Andreas Küchler and Christoph Goller. Inductive Learning in Symbolic Domains Using Structure-Driven Recurrent Neural Networks. In Günther Görz and Steffen Hölldobler, editors, *KI-96: Advances in Artificial Intelligence*, Lecture Notes in Computer Science (LNCS 1137), pages 183–197, Berlin, 1996. Springer.

[47] O. Lupanov. Circuits using Threshold elements. *Soviet Physics – Doklady*, 17(2):91–93, 1972.

[48] Wolfgang Maass. Vapnik-Chervonenkis Dimension of Neural Nets. NeuroCOLT Technical Report Series NC-TR-96-015, Institute for Theoretical Computer Science, Technische Universität Graz, 1996.

[49] Wolfgang Maass and Eduardo Sontag. Analog Neural Nets with Gaussian or other Common Noise Distributions cannot Recognize Arbitrary Regular Languages. 1998. submitted.

[50] Laurent Miclet. *Structural Methods in Pattern Recognition.* North Oxford Academic, 1986.

[51] Marvin Minsky. *Computation: Finite and Infinite Machines*, chapter 3. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1967.

[52] Christopher Moore. Dynamical Recognizers: Real-time Language Recognition by Analog Computers. *Theoretical Computer Science*, 1998. to appear.

[53] S. Muroga. *Threshold Logic and its Applications.* Wiley, New York, 1971.

[54] L. Niklasson and N. E. Sharkey. Systematicity and Generalization in Compositional Connectionist Representations. In G. Dorffner, editor, *Neural Networks and a New Artificial Intelligence*, pages 217–232. Thomson Computer Press, London, UK, 1997.

[55] C.W. Omlin and C.L. Giles. Constructing Deterministic Finite-State Automata in Recurrent Neural Networks. *Journal of the ACM*, 43(2):937–972, 1996.

[56] C.W. Omlin and C.L. Giles. Extraction of Rules from Discrete-time Recurrent Neural Networks. *Neural Networks*, 9(1):41–52, 1996.

[57] Jordan B. Pollack. The Induction of Dynamical Recognizers. *Machine Learning*, (7):227–252, 1991.

[58] Martin Riedmiller. Supervised Learning in Multilayer Perceptrons – from Backpropagation to Adaptive Learning Techniques. *Intl. Journal of Computer Standards and Interfaces*, 16(3), 1994. Special Issue on Artificial Neural Networks.

[59] Robert Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches.* John Wiley & Sons, Inc., 1992.

[60] Thorsten Schmitt. Evaluation of the Neural Folding Architecture for Learning Tasks concerning Logical Terms and Chemical Structures. Masters Thesis, Technical University Munich, Faculty of Computer Science, 1997.

[61] Thorsten Schmitt and Christoph Goller. Relating Chemical Structure to Activity: An Application of the Neural Folding Architecture. In *Proceedings of the fifth International GI-Workshop on Fuzzy-Neuro Systems'98*, 1998. to appear.

[62] Stephan Schulz, Andreas Küchler, and Christoph Goller. Some Experiments on the Applicability of Folding Architecture Networks to Guide Theorem Proving. In D.D. Dankel II, editor, *Proc. of the 10th FLAIRS Conference, Daytona Beach*, pages 377–381. Florida AI Research Society, 1997.

[63] Jude W. Shavlik. Combining Symbolic and Neural Learning. *Machine Learning*, 14:321–331, 1994.

[64] Takeshi Shinohara and Setsuo Arikawa. Pattern Inference. In Klaus P. Jantke and Steffen Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 259–291. Springer-Verlag, 1995.

[65] Hava T. Siegelmann and C. Lee Giles. The Complexity of Language Recognition by Neural Networks. *Neurocomputing*, 15:327–345, 1997.

[66] Hava T. Siegelmann and Eduardo D. Sontag. On the Computational Power of Neural Nets. *Journal of Computer and System Sciences*, 50:132–150, 1995.

[67] Kai-Yeung Siu, Vwani Roychowdhury, and Thomas Kailath. Depth-Size Tradeoffs for Neural Computation. *IEEE Transactions on Computers*, 40(12):1402–1412, December 1991.

[68] Kai-Yeung Siu, Vwani Roychowdhury, and Thomas Kailath. *Discrete Neural Computation – A Theoretical Foundation.* Information and System Sciences Series. Prentice Hall, 1995.

[69] Alessandro Sperduti. Neural Networks for the Classification of Structures. In *Workshop Notes of the ECAI'96 Workshop on Neural Networks and Structured Knowledge*, pages 68–75, 1996.

[70] Alessandro Sperduti. On the Computational Power of Recurrent Neural Networks for Structures. *Neural Networks*, 10(3):395–400, April 1997.

[71] Mark Steijvers and Peter Grünwald. A Recurrent Network that Performs a Context-sensitive Prediction Task. NeuroCOLT Technical Report Series NC-TR-96-035, February 1996.

[72] Magnus Steinby. Tree Language Problems in Pattern Recognition Theory. In J. Csirik, J. Demetrovics, and F. Gécseg, editors, *Proceedings of the International Conference on Fundamentals of Computation Theory*, volume 380 of *Lecture Notes in Computer Science (LNCS)*, pages 444–450. Springer, 1989.

[73] Whitney Tabor. Metric Relations among Analog Computers, 1997. submitted.

[74] Peter Tino. Fixed Points in Two–Neuron Discrete Time Recurrent Networks: Stability and Bifurcation Considerations. Technical Report UMIACS-TR-95-51, University of Maryland, 1995.

[75] Peter Tino, Bill G. Horne, C. Lee Giles, and Pete C. Collingwood. Finite State Machines and Recurrent Neural Networks – Automata and Dynamical Systems Approaches. In J.E. Dayhoff and O. Omidvar, editors, *Temporal Dynamics and Time-Varying Pattern Recognition*, special volume: Progress in Neural Networks. Ablex Publishing, 1997. to be published.

[76] Peter Tino and Jozef Sajda. Learning and Extracting Initial Mealy Automata with a Modular Neural Network Model. *Neural Computation*, 7:822–844, 1995.

[77] G. Towell and J. Shavlik. The Extraction of Refined Rules From Knowledge Based Neural Networks. *Machine Learning*, 13(1):71–101, 1993.

[78] Leslie G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[79] Tim van Gelder. Compositionality: A Connectionist Variation on a Classical Theme. *Cognitive Science*, 14:355–384, 1990.

[80] Janet Wiles and Jeff Elman. Learning to Count without a Counter: A Case Study of Dynamics and Activation Landscapes in Recurrent Networks. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, Cambridge, MA, 1995.

[81] Reinhard Wilhelm and Dieter Maurer. *Compiler Design*, chapter 11, "Trees: Pattern Matching and Parsing", pages 513–546. Addison-Wesley, 1995.

[82] Zheng Zeng, Rodney M. Goodman, and Padhraic Smyth. Discrete Recurrent Neural Networks for Grammatical Inference. *IEEE Transactions on Neural Networks*, 5(2), March 1994.