A mathematical characterization of the architectural bias of recursive models

Barbara Hammer HAMMER@INFORMATIK.UNI-OSNABRUECK.DE
Department of Mathematics/Computer Science
University of Osnabrück
D-49069 Osnabrück, Germany

Peter Tiňo P.TINO@CS.BHAM.AC.UK
School of Computer Science
University of Birmingham
Edgbaston, Birmingham B15 2TT, UK

Alessio Micheli MICHELI@DI.UNIPI.IT
Dipartimento di Informatica
Università di Pisa
Via Buonarroti 2, 56127 Pisa, Italy

Abstract

It has recently been shown that recurrent neural networks (RNNs) with small weights are equivalent to definite memory machines (36). This equivalence explains the architectural bias of training when initializing RNNs with small weights. It has consequences on the dynamical properties of RNNs found during training and their generalization ability as demonstrated in (36, 68). In recent years a straightforward generalization of RNNs has been established which can deal with more complex data structures: recursive neural networks (RecNNs) process trees and acyclic graphs as they occur, for example, in natural language processing, bioinformatics or chemistry (9, 19, 59, 66). We here extend the results of (36) for RNNs to RecNNs and establish an equivalence of small weight RecNNs to Markovian tree processing models. In addition, we discuss the effects of the architectural bias of training on the performance and we derive bounds on the generalization ability of the models.

Keywords: Architectural Bias, Recursive Networks, Finite Memory Models, Learnability

1. Introduction

Feedforward neural networks (FNNs) and statistical counterparts constitute powerful machine learning tools for classification or regression tasks. Canonical extensions to sequential data which occur in bioinformatic, natural language processing, or time-series prediction, for example, are provided by recurrent neural networks (RNNs) and statistical formulations thereof (7, 50, 67). In recent years, an increasing interest in the possibility to process even more expressive data structures can be observed. Regression models for spatial data such as biological sequences can be improved when further dependencies of the single elements are taken into account as demonstrated in (3). Here, bidirectional RNNs are superior to simple RNNs or FNNs for secondary structure prediction of proteins. Other models like kernel methods focus on adequate kernel design for complex data structures such that they can deal with sequences or tree structures in a natural way (37, 43, 52, 53, 76). We are interested in recursive neural networks (RecNNs) which constitute a canonical extension of recurrent neural networks to tree structures or acyclic directed graphs as proposed in (19, 26, 51, 65).

RecNNs have first been introduced in the context of logic to learn search heuristics for automated deduction (25). Until now, they have been successfully applied in various different areas: chemical formulas can often be represented as directed acyclic graphs. RecNNs have thus been used for quantitative structure properties relationship tasks and quantitative structure activity relationship tasks which occur in the context of drug design (9). Graphics or pictures can be represented by their basic primitives and their relative position, the former constituting the nodes of a tree structure, the latter building the connection structure of the tree or graph. The approach (56) demonstrates the applicability of RecNNs within this context. The work (16) considers the classification of documents based on their global layout which is also represented as a tree structure. In (77), RecNNs are combined with support vector machines (SVMs) for fingerprint classification. RecNNs thereby extract relevant, data adapted features which serve as input to the SVM. Prediction of global protein structure based on the sequence of amino acids includes as a subproblem the prediction of contact maps and specific bindings. The work (59, 75) proposes RecNNs for this task. Thereby, the results compare favorably to FNNs or bicausal RNNs (75) and the accuracy of contact prediction exceeds all previously reported results. The approach (66) presents RecNNs for problems in natural language parsing. Interestingly, RecNNs here compare favorably to SVMs with convolutional kernels in the sense that the achieved accuracy is comparable, but the training complexity is much smaller for RecNNs compared to SVMs due to the large data base which is used (17).

It is well known that training recurrent networks faces severe problems. Though they are very powerful in theory (20, 45, 61, 62) training algorithms might fail in practice due to vanishing gradients and the problem of long-term dependencies on the one side (8, 39) and restrictions due to noise (54, 55) on the other side. Thus RNNs are implicitly biased during training towards specific dynamical models which can be trained easily whereas other dynamics require additional regularization terms of training (10, 14, 23, 39). RNNs are usually initialized with small weights. It has been observed that the activation of input driven RNNs initialized with small weights shows meaningful clusters even before training (13, 48, 49). In (36) this architectural bias due to initialization with small weights has been equivalently characterized as Markovian. Markovian models constitute simple though useful models for which several efficient alternative training algorithms have been designed (11, 27, 60, 70). Hence standard recurrent neural network training first explores regions of the search space which correspond to simple but potential useful dynamics before testing more complex dynamics such as finite state machines or alternatives (73). This architectural bias has been verified experimentally (13, 48, 49, 68, 69). It can even be quantified by measuring the fractal dimension of the activations (71, 72). It corresponds to the way in which humans recognize language as pointed out in (13). It can be expected that RNNs are particularly suited for Markovian task and the more the dynamics to be learned differs from this bias, the more complex becomes standard RNN training. Interestingly, when training RNNs for Markovian tasks, adaptation of the interior recursive part might not be necessary but initialization of the recursive part with small weights already provides appropriate embedding of data in a real-vector space based on which standard classifiers can be applied as demonstrated in (68, 69).

Recursive neural networks constitute powerful models for tree structures: in theory, they posses the universal approximation ability and can implement tree automata or even more complex dynamics for arbitrarily high trees (32, 26). In practice, problems of training due to noise or long term dependencies for RNNs transfer to RecNNs (32). Thereby the problem of long term dependencies is here less significant since representation of data by tree structures might avoid the necessity to latch information over a long time interval – a balanced tree with n nodes has paths of length only $O(\log(n))$ (17). The problem of noise, however, remains. In addition, the generalization ability of RecNNs might be in theory not hold in the sense that they are not PAC-learnable (31, 32). It is hence interesting to investigate the architectural bias of RecNNs due to initialization. The induced dynamical behavior gives hints on which tasks are particularly suited for RecNN models and which regularization is put towards training of RecNNs. Similar to RNNs, recursive neural networks are initialized with small weights and a similar architectural bias can thus be expected. We are interested in an exact mathematical characterization of this architectural bias for RecNNs. An exact characterization allows us to assess which problems can be solved easily by these models when initialized with small weights, possibly even without training of the recursive part. It gives hints which tasks are potentially difficult to solve with RecNNs and thus require good training algorithms or alternative models such as specifically designed kernels. We will characterize the bias of RecNNs as Markovian. In addition, we derive explicit bounds on the generalization error of small weight RecNNs thus demonstrating the regularization due to this bias in terms of statistical learning theory: unlike general RecNNs with arbitrary precision, small weight RecNNs are PAC learnable even for arbitrary (possibly real-valued) input trees of potentially arbitrary height (31, 32).

We will now first introduce tree processing models and recast the situation for the case of sequences. We will then extend the results of (36) to recursive tree processing models and show an equivalence of small weight recursive networks to Markovian models. Consequences on the generalization ability are discussed from a theoretical point of view by establishing the uniform convergence of empirical distances property of RecNNs. Finally, the generalization ability of RecNNs is monitored in two experiments.

2. Tree-processing models

We are interested in models which can directly process labeled trees. Thereby, X is a fixed set of the labels. A *tree* t with fan-out $k \in \mathbb{N}$ consists of either the empty tree ξ or it is recursively defined as a root node labeled by $l(t) \in X$ and k child trees $c_1(t), \ldots, c_k(t)$ some of which might be empty. The root node is called parent of its children. In the following we refer to the set of trees over X with fan-out k by the symbol X_k^*. Nodes of which all children are empty are called leaves. We denote a tree with root label l and children $c_1, \ldots, c_k$ in prefix notation by $l(c_1, \ldots, c_k)$. We thereby omit empty children of leaves. A tree can be considered as a graph with labeled nodes and edges between parents and their children. The depth of a node of the tree denotes the length of the shortest path from the root of the tree to the node in this graph. The *height* of a tree t is the maximum depths of the nonempty nodes in the tree. A tree consisting of only a root node and empty children has height 0.

Note that choosing $k = 1$ gives sequences. A larger fan-out k is convenient for adequate representation of data in several application areas such as logic, chemistry, natural language parsing, or bioinformatics. Since terms and logical formulas can be interpreted as tree structures, RecNNs can be used for every domain which can be described by terms or formulas (25, 51, 66). In addition, acyclic directed graphs can be interpreted as tree structures with shared nodes such that RecNNs also constitute valuable tools for chemistry, processing of graphical objects, or bioinformatics (9, 16, 56, 59).

We focus on prediction models on trees. Assume Y is some output space. Then a prediction model implements a function $f : X_k^* \to Y$. Examples might be such simple functions as mapping trees to their height or complex connections such as relating the activity of drugs to their molecular structure (represented by trees). Several specific forms of mappings are of interest in the following. For convenience, we first introduce prediction models we are interested in for the specific situation of sequences, i.e. we consider the case $k = 1$, and we formally state the results of (36) for the case of sequences. Afterwards, we extend the models to tree structures. The remaining part of this paper will then be devoted to a generalization of the results from (36) to tree processing models.

For fan-out 1, we can recast the prefix notation of a tree $l_1(l_2(l_3(\ldots)))$ as sequence $[l_1, l_2, l_3, \ldots]$ whereby l_1 denotes the most recent entry of the sequence. $[\,]$ denotes the empty sequence. We refer to sequences over X by X^*. Markovian models constitute simple, yet powerful models which only depend on a finite memory length L, i.e. the last L items of the sequence and drop all other entries from more distant past. Denote by $T_L : X^* \to X^*$ the L-truncation of sequences, i.e.

$$T_L([x_1, \dots, x_l]) = \begin{cases} [x_1, \dots, x_l] & \text{if } l \leq L \\ [x_1, \dots, x_L] & \text{otherwise} \end{cases}$$

Functions $f : X^* \to Y$ are *Markovian* if some length L exists such that

$$f(s) = f(T_L(s)) \text{ for all } s \in X^*$$

holds. In the case of symbolic sequences, i.e. $|X| < \infty$, Markovian models are also known as definite memory machines or DeBruijn automata (46). Markovian models are restricted due to the finite memory length. They cannot compute the parity function over $\{0, 1\}^*$, for example.

More general models such as finite automata rely on a recursive processing of sequences. Assume a set U, an element $y \in U$, and some function $f : X \times U \to U$ are given. Then f *induces* a recursive function $\tilde{f}_y : X^* \to U$,

$$\tilde{f}_y(s) = \begin{cases} y & \text{if } s = [\,] \\ f(x_1, \tilde{f}_y([x_2, \dots, x_l])) & \text{if } s = [x_1, \dots, x_l] \end{cases}$$

y is called the *initial context*. The *induced function with finite memory length* L is defined by $\tilde{f}_y^L : X^* \to U$,

$$\tilde{f}_y^L(s) = \tilde{f}_y(T_L(s)) \,.$$

A *recursive model* computes a function of the form $g \circ \tilde{f}_y : X^* \to Y$ whereby $f : X \times U \to U$ and $g : U \to Y$. Thus recursive models recursively process sequences starting from the last entry to the most recent one applying the transition function f. If the alphabets X, U, and Y are finite, recursive models are known as Mealy machines. If in addition g is the identity, finite automata arise (40). If the recursive functions f and g can be computed by a standard feedforward network we get recurrent neural networks (RNNs) (35, 50).

Since RNNs and finite automata can be described by the same dynamical equation, it is not surprising that RNNs can simulate finite automata for inputs of arbitrary length as demonstrated in mathematical proofs and experiments (1, 18, 24, 57, 58, 73). Since RNNs can also use real-vector spaces for X, U, and Y, they constitute more powerful models capable of simulating also Turing machines and non-uniform Boolean circuits in theory and are capable of learning some non-regular languages such as the context sensitive language $\{a^n b^n c^n \mid n\}$ in practice (10, 33, 45, 61, 62). However, learning tasks for RNNs with complex dynamical properties often face severe problems and training succeeds only for comparably short input sequences. One problem within this context is that long-term dependencies cannot reliably be propagated by means of gradient algorithms (8). Another problem is related to the fact that the generalization ability of RNNs might be substantially worse compared to standard feedforward networks due to the more complex input structures (34, 47). Furthermore, simulations of Turing machines or alternative models with RNNs rely on the assumption that computation is not subject to computation noise or errors (33, 45, 61, 62). In the presence of noise the capability of RNNs boils down to finite automata or even less depending on the type of noise (54, 55). Hence the design of fast and efficient training algorithms for RNNs still constitutes a challenging problem of ongoing research (35, 39).

Specific learning tasks for RNNs can be solved very easily and training shows good results for these settings (14, 23). In several cases, this happens even if solely the output function g is trained and the transition function f is initialized with small weights, but no further adaptation of f takes

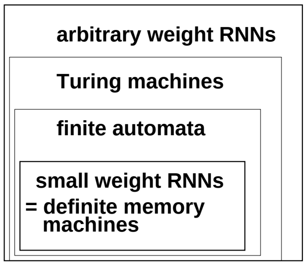


Figure 1: Hierarchy of computation models which can be implemented by RNNs with different size weights.

place, as demonstrated in the experiments in (68, 69). This fact can be explained by the architectural bias of RNNs when initialized with small weights. Even before training, clusters of the hidden neuron activations can be observed as pointed out in (48, 49) and effective prediction models can be based on this interior representation of sequences (68, 69). Functions within the architectural bias of RNNs can be reached easily during training, possibly even without further adaptation of the transition function. Thus RNNs are particularly suited for training tasks close to their architectural bias. It has been shown in (36) that RNNs with small weights implement Markovian models with finite memory length L. Conversely, all definite memory machines can be implemented by RNNs with small weights. Hence the global picture as depicted in Fig. 1 arises. RNNs are super Turing universal if arbitrary weights and precision of the computation might be used. Restriction of the weights reduces the capacity to only Markovian models. Consequences of this fact on generalization bounds have been discussed in (36) and the bias of randomly initialized RNNs has quantitatively been investigated by means of fractal analysis in (71, 72).

We are here interested in related results for more general structure processing models, i.e. recursive networks which can deal with tree structures. We thus extend the above definitions of sequence processing models to trees. Fix some fan-out k of the trees which might be larger than 1. Denote by $T_L : X_k^* \to X_k^*$ for $L \geq 0$ the L truncation of trees which substitutes all children of the nodes of depth L of a given tree by the empty tree. Formally, we can define T_L recursively by

$$T_L(t) = \begin{cases} \xi & \text{if } L = 0 \text{ or } t = \xi \\ l(T_{L-1}(c_1), \ldots, T_{L-1}(c_k)) & \text{if } L > 0 \text{ and } t = l(c_1, \ldots, c_k) \end{cases}$$

A *Markovian* model for trees implements some function which only depends on the topmost part of the trees, but not on the nodes of high depth.

Definition 1 *A Markovian model for trees implements a function* $: X_k^* \to Y$ *such that some* $L \in \mathbb{N}$ *can be found with*

$$f(t) = f(T_L(t)) \text{ for all } t \in X_k^*$$

If X and Y are finite sets, we refer to Markovian models as definite memory tree models in analogy to definite memory machines. Naturally, Markovian models constitute only a subclass of tree models since they cannot distinguish nodes of depth larger than L. Markovian models with large L, however, can approximate general models. In analogy to recursive sequence models we can define recursive tree-processing models as follows:

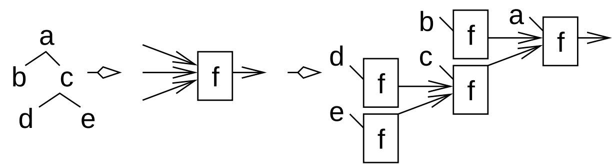


Figure 2: Example of a computation with a recursive function for a given binary tree.

Definition 2 *Assume a set U, an element $y \in U$, and a function $f : X \times U^k \to U$ are given, whereby U^k denotes the product of U. Then the* induced *recursive function on trees is recursively defined as $\tilde{f}_y : X_k^* \to U$,*

$$\tilde{f}_y(t) = \begin{cases} y & \text{if } t = \xi \\ f(l, \tilde{f}_y(c_1), \ldots, \tilde{f}_y(c_k)) & \text{if } t = l(c_1, \ldots, c_k) \end{cases}$$

The induced *recursive function* with finite memory length L *is defined as $\tilde{f}_y^L : X_k^* \to U$,*

$$\tilde{f}_y^L(t) = \tilde{f}_y(T_L(t))$$

A recursive model *computes a function of the form $g \circ \tilde{f}_y : X_k^* \to Y$ whereby $f : X \times U^k \to U$ and $g : U \to Y$.*

Hence a recursive model recursively processes tree structures starting from the leaves up to the nodes. Fig. 2 shows an example of the dynamics. If X, U, and Y are finite alphabets, recursive models are referred to as bottom-up state transducers. If in addition g is the identity, bottom-up tree automata arise (21, 22). Recursive networks are obtained if f and g can be computed by a standard feedforward neural networks (19, 32, 65). Note that RecNNs can be considered a generalization of RNNs to tree structures with fan-out k different from 1. They can be used for application areas involving bioinformatics, chemistry, logic, or natural language parsing where data can canonically be represented by tree structures (9, 19, 59, 66). Tree structures often yield a compact representation of data with comparably small height of the tree, such that the problem of long-term dependencies for training is widely reduced compared to RNNs. In analogy to RNNs, it can be proved that tree automata can be simulated by RecNNs for arbitrary height trees (12). Since RecNNs can also use infinite alphabets and since they include RNNs as a special case, their in principle computational ability is strictly larger. As for RNNs, these results rely on the assumption of arbitrary precision of the computation and the fact that arbitrarily large weights can be used. In the presence of noise, the capability of RecNNs can be restricted in analogy to RNNs (32).

RecNNs are usually initialized with small weights such that an architectural bias is put towards training. We are interested in a characterization of this architectural bias to see which tasks can easily be tackled by RecNNs when initialized with small weights. We will show in the following that RecNNs with small weights can be simulated by Markovian models and, conversely, every definite memory tree model can be implemented by a RecNN with small weights. Thus we will characterize the bias of RecNN training initialized with small weights as Markovian.

3. RecNNs with small weights implement Markovian models

We first consider general recursive functions of the form

$$g \circ \tilde{f}_y : X_k^* \to Y$$

whereby the transition function $f : X \times U^k \to U$ is a contraction with respect to U. We will establish the Markovian property in this more general case and specify the situation for RecNNs later. Assume U and Y are equipped with metrics both of which we denote by $|\cdot|$.

Definition 3 *A function $f : X \times U^k \to U$ is a* contraction *with respect to U if some parameter $C \in (0,1)$ exists such that*

$$|f(x, u_1, \ldots, u_k) - f(x, u'_1, \ldots, u'_k)| \le C \cdot \max\{|u_1 - u'_1|, \ldots, |u_k - u'_k|\}$$

holds for all $x \in X$ and $u_1, \ldots, u_k, u'_1, \ldots, u'_k \in U$.

One can establish for every contractive transition function and every bounded set U a Markovian property:

Lemma 4 *Assume $f : X \times U^k \to U$ is a contraction with respect to U with parameter C. Assume U has finite diameter, i.e. for all $u_1, u_2 \in U$ the inequality $|u_1 - u_2| \le C_0$ holds for some $C_0 > 0$. Assume $\epsilon > 0$. Then, for memory length $L_0 = \log_C(\epsilon/C_0)$, we have*

$$|\tilde{f}_y(t) - \tilde{f}_y^{L_0}(t)| \le \epsilon$$

Proof Assume two trees t and $t' \in X_k^*$ are given such that $T_L(t) = T_L(t')$ for some $L \in \mathbb{N}$. We show by induction over L that the inequality $|\tilde{f}_y(t) - \tilde{f}_y(t')| \le C^L \cdot C_0$ holds. For $L = 0$ the statement is obvious since the evaluation of $\tilde{f}_y$ at the trees t and t' gives two points in U which diameter is limited by C_0. Assume $L \ge 1$. The situation is also obvious if t or t' are empty. In this case, $T_L(t) = T_L(t')$ implies $t = t' = \xi$. If $t = l(c_1, \ldots, c_l)$ and $t' = l'(c'_1, \ldots, c'_k)$ we find

$$\begin{aligned}
|\tilde{f}_y(t) - \tilde{f}_y(t')| &= |\tilde{f}_y(l(c_1, \ldots, c_l)) - \tilde{f}_y(l'(c'_1, \ldots, c'_k))| \\
&= |f(l, \tilde{f}_y(c_1), \ldots, \tilde{f}_y(c_k)) - f(l, \tilde{f}_y(c'_1), \ldots, \tilde{f}_y(c'_k))| \\
&\le C \cdot \max\{|\tilde{f}_y(c_1) - \tilde{f}_y(c'_1)|, \ldots, |\tilde{f}_y(c_k) - \tilde{f}_y(c'_k)|\} \\
&\le C \cdot C^{L-1} \cdot C_0
\end{aligned}$$

because $T_L(t) = T_L(t')$ has the consequence that the roots have the same label, i.e. $l = l'$, and the $L-1$ truncations of the subtrees c_i and c'_i coincide for all i such that we can apply the induction hypothesis. Because of $T_{L_0}(t) = T_{L_0}(T_{L_0}(t))$ we find

$$|\tilde{f}_y(t) - \tilde{f}_y^{L_0}(t)| = |\tilde{f}_y(t) - \tilde{f}_y(T_{L_0}(t))| \le C^{L_0} \cdot C_0 = \epsilon$$

for $L_0 = \log_C(\epsilon/C_0)$. □

We obtain as immediate corollary that recursive tree models for which the transition function is contractive and the output function is Lipschitz continuous constitute Markovian models. Recall that a function $g : U \to Y$ is Lipschitz continuous with parameter C if $|f(x) - f(x')| \le C \cdot |x - x'|$ holds for all $x, x' \in Y$.

Theorem 5 *Assume $f : X \times U^k \to U$ is contractive with respect to U with parameter C, U is bounded with diameter C_0, and g is Lipschitz continuous with parameter C_1. Assume $\epsilon > 0$. Then for $L_0 = \log_C(\epsilon/(C_0 C_1))$ the inequality*

$$|g \circ \tilde{f}_y(t) - g \circ \tilde{f}_y^L(t)| \le \epsilon$$

holds for all $t \in X_k^*$.

Proof Using Lemma 4 we find

$$|g \circ \tilde{f}_y(t) - g \circ \tilde{f}_y^L(t)| \leq C_1 \cdot |\tilde{f}_y(t) - \tilde{f}_y^L(t)| \leq C_1 \cdot \epsilon / C_1$$

□

Thus finite memory length is sufficient to approximate recursive tree models with a contractive transition function arbitrarily well. We would like to go a step further and show that such models can be interpreted as definite memory tree models for which the sets X, Y, and U are finite. Note that RecNNs use a priorly unlimited subset of a real-vector space $U \subset \mathbb{R}^n$ to represent internal states whereas tree transducers use a finite set of states only.

Definition 6 *Assume $\mathcal{F}$ is a function class with domain X and codomain Y, such that Y is equipped with a metric $|\cdot|$. For $f, g \in \mathcal{F}$ we denote the maximum distance by $|f-g|_\infty = \sup_{x \in X} |f(x)-g(x)|$. Assume $\epsilon > 0$. An* external cover *of $\mathcal{F}$ with accuracy ϵ consists of a set of functions $C(\mathcal{F})$ from X to Y, which need not be contained in $\mathcal{F}$, such that for all $f \in \mathcal{F}$ a function $g \in C(\mathcal{F})$ can be found with $|f-g|_\infty \leq \epsilon$.*

Note that for every function class an external cover, the class itself, can be found. A *finite ϵ-cover* of a set U consists of a finite number of points $\{u_1, \ldots, u_n\}$ such that for every $u \in U$ some u_i with $|u - u_i| \leq \epsilon$ exists. A set U is *totally bounded* if for every ϵ a finite ϵ-cover exists. Denote by $\tilde{\mathcal{F}}$ the class of all functions of the form $\tilde{f}_y$ with $f \in \mathcal{F}$ and $y \in U$. Denote by $\tilde{\mathcal{F}}^L$ the set of all truncated functions $\tilde{f}_y^L$ with $f \in \mathcal{F}$ and $y \in U$. The contraction property allows us to transfer covers of the space of transition functions to covers of the space of recursive functions.

Lemma 7 *Assume $\mathcal{F}$ is a set of functions mapping $X \times U^k$ to U, such that every $f \in \mathcal{F}$ forms a contraction with respect to U with parameter C. Assume $\epsilon > 0$. Assume $C(\mathcal{F})$ is an external cover of $\mathcal{F}$ with parameter ϵ. Assume U is totally bounded and the constants $\{u_1, \ldots, u_n\}$ cover U with parameter ϵ. Then $\{\tilde{f}_{u_j} \,|\, f \in C(\mathcal{F}), j = 1, \ldots, n\}$ is an external cover of $\tilde{\mathcal{F}}$ with parameter $\epsilon/(1-C)$ and $\{\tilde{f}_{u_j}^L \mid f \in C(\mathcal{F}), j = 1, \ldots, n\}$ is an external cover of $\tilde{\mathcal{F}}_L$ with parameter $\epsilon/(1-C)$.*

Proof Assume $f \in \mathcal{F}$ and $y \in U$. Choose a function f' from the cover $C(\mathcal{F})$ such that $|f - f'|_\infty \leq \epsilon$. Choose a value u_j from $\{u_1, \ldots, u_n\}$ such that $|y - u_j| \leq \epsilon$. It follows by induction over the height h of a tree $t \in X_k^*$ that $|\tilde{f}_y(t) - \tilde{g}_{u_j}(t)| \leq \epsilon \cdot (1 - C^{h+1})/(1-C)$ as follows:
For $t = \xi$ we find

$$|\tilde{f}_y(t) - \tilde{f}'_{u_j}(t)| = |y - u_j| \leq \epsilon = \epsilon \cdot \frac{1 - C^1}{1 - C} .$$

For $t = l(c_1, \ldots, c_k)$ we find

$$\begin{aligned}
& |\tilde{f}_y(l(c_1, \ldots, c_k)) - \tilde{f}'_{u_j}(l(c_1, \ldots, c_k))| \\
= \;& |f(l, \tilde{f}_y(c_1), \ldots, \tilde{f}_y(c_k)) - f'(l, \tilde{f}'_{u_j}(c_1), \ldots, \tilde{f}'_{u_j}(c_k))| \\
\leq \;& |f(l, \tilde{f}_y(c_1), \ldots, \tilde{f}_y(c_k)) - f(l, \tilde{f}'_{u_j}(c_1), \ldots, \tilde{f}'_{u_j}(c_k)))| \\
& + |f(l, \tilde{f}'_{u_j}(c_1), \ldots, \tilde{f}'_{u_j}(c_k))) - f'(l, \tilde{f}'_{u_j}(c_1), \ldots, \tilde{f}'_{u_j}(c_k)))| \\
\leq \;& C \cdot \max_i |\tilde{f}_y(c_i) - \tilde{f}'_{u_j}(c_i)| + \epsilon \\
\leq \;& C \cdot \epsilon \cdot (1 - C^h)/(1 - C) + \epsilon = \epsilon \cdot (1 - C^{h+1})/(1 - C)
\end{aligned}$$

by induction. Since $\epsilon \cdot (1 - C^{h+1})/(1 - C) \leq \epsilon/(1 - C)$, the result follows. □

Assume U is totally bounded and $\{u_1, \dots, u_n\}$ is an ϵ-cover of U. Then we can obviously approximate every function $f : X \times U^k \to U$ by a function the codomain of which is contained in $\{u_1, \dots, u_n\}$ only. Hence we can cover every set of functions mapping to U by functions with images in the discrete set $\{u_1, \dots, u_n\}$. Since the initial contexts in the above Lemma can be chosen as elements of the set $\{u_1, \dots, u_n\}$ and the approximations in the cover only yield values in that set, we obtain as an immediate corollary that a finite set U is sufficient for internal processing:

Corollary 8 *Assume $\mathcal{F}$ and U are as above. Assume $\epsilon > 0$; assume $\{u_1, \dots, u_n\}$ is an ϵ-cover of U. Denote by $Q : U \to \{u_1, \dots, u_n\}$ the quantization mapping, which maps a value $u \in U$ to the nearest value u_i (some fixed nearest u_i if this is not unique). Denote by f^Q the composition $f^Q(x, u) = Q(f(x, u))$. Then $\{(\tilde{f}^Q)_{u_j} \mid f \in \mathcal{F}, j = 1, \dots, n\}$ forms an $\epsilon/(1 - C)$-cover of $\tilde{\mathcal{F}}$ and $\{(\tilde{f}^Q)^L_{u_j} \mid f \in \mathcal{F}, j = 1, \dots, n\}$ forms an $\epsilon/(1-C)$-cover of $\tilde{\mathcal{F}}_L$. Note that these functions use values of $\{u_1, \dots, u_n\}$ only.*

Proof $\{f^Q \mid f \in \mathcal{F}\}$ constitutes an external ϵ-cover of $\mathcal{F}$ because the outputs are changed by at most ϵ. Moreover, $\{u_1, \dots, u_n\}$ constitutes an ϵ-cover of U by assumption. Hence we can apply Lemma 7. As a consequence, the recursive classes $\{(\tilde{f}^Q)_{u_j} \mid f \in \mathcal{F}, j = 1, \dots, n\}$ and $\{(\tilde{f}^Q)^L_{u_j} \mid f \in \mathcal{F}, j = 1, \dots, n\}$ form $\epsilon/(1 - C)$-covers of $\tilde{\mathcal{F}}$ and $\tilde{\mathcal{F}}_L$, respectively. $\square$

Hence, we can substitute every recursive function where the transition constitutes a contraction by a function which uses only a *finite* number of different values for U and a *finite* memory length. This result can be used to show that RecNNs can be simulated by definite memory tree machines which only use a finite input alphabet (and hence also only a finite output set).

Theorem 9 *For every $\epsilon > 0$, function $f : X \times U \to U$ with totally bounded domain U such that f is a contraction with parameter C, function $g : U \to Y$ which is Lipschitz continuous with parameter C_1, and initial context $y \in U$, we can find a finite memory length L, a finite set $\Sigma := \{x_1, \dots, x_m\}$ in X, a quantization map $Q : X \to \{x_1, \dots, x_m\}$, and a function $h : \Sigma^*_k \to Y$ which is Markovian with memory length L, such that the following holds for all trees $t \in X^*_k$:*

$$|g \circ \tilde{f}_y(t) - h(\tilde{Q}(t))| \leq \epsilon$$

where $\tilde{Q}(t)$ denotes the mapping which maps each label l in the tree t to the value $Q(l) \in \Sigma$.

Proof As a consequence of Lemma 4 and Corollary 8 we can approximate $\tilde{f}_y$ up to ϵ/C_1 by a function $\tilde{p}^L_{u_i}$ which uses only a finite number of values $\{u_1, \dots, u_n\}$ in U (n is chosen large enough) and a (sufficiently large) finite memory length L. Define equivalence classes on X via the definition $x \sim x'$ for $x, x' \in X$ iff $p(x, y_1, \dots, y_k) = p(x', y_1, \dots, y_k)$ for all $y_1, \dots, y_k \in \{u_1, \dots, u_n\}$. This yields only a finite number of equivalence classes. Choose a fixed value x_j from each equivalence class and denote the union of these representatives by Σ. Define $Q : x \mapsto x_j$ if x lies in the equivalence class of x_j. Then the choice $h := g \circ \tilde{p}^L_{u_i}$ yields the desired approximation since g is Lipschitz continuous with parameter C_1 and $\tilde{p}^L_{u_{i_0}}(\tilde{Q}(t))$ differs from $\tilde{f}_y(t)$ by no more than ϵ/C_1. $\square$

Hence we know that contractive transition functions lead to Markovian models and, in addition, the possibly infinite input set can equivalently be substituted by a finite set such that only definite memory tree models arise. We now consider the specific situation of recursive neural networks. We assume that $X = \mathbb{R}^{n_1}$, $U = \mathbb{R}^{n_2}$, and $Y = \mathbb{R}^{n_3}$ are real-vector spaces equipped with the maximum norm which we denote by $|\cdot|_\infty$.

Definition 10 *A* recursive network *(RecNN) computes a function of the form* $g \circ \tilde{f}_y : (\mathbb{R}^{n_1})^*_k \to \mathbb{R}^{n_3}$, *where* $g : \mathbb{R}^{n_2} \to \mathbb{R}^{n_3}$ *and* $f : \mathbb{R}^{n_1} \times \mathbb{R}^{k \cdot n_2} \to \mathbb{R}^{n_2}$ *can be written as*

$$g(u) = M_1 \cdot \boldsymbol{\sigma}(M_2 \cdot u + d_1) + d_2, \quad f(x, u_1, \ldots, u_k) = \boldsymbol{\sigma}(W_1 \cdot x + W_2^1 \cdot u_1 + \ldots + W_2^k \cdot u_k + b)$$

$M_1 \in \mathbb{R}^{n_3 \times n_4}$, $M_2 \in \mathbb{R}^{n_4 \times n_2}$, $W_1 \in \mathbb{R}^{n_2} \times \mathbb{R}^{n_1}$, *and* $W_2^i \in \mathbb{R}^{n_2}$ $(i = 1, \ldots, k)$ *are matrices,* $d_1 \in \mathbb{R}^{n_4}$, $d_2 \in \mathbb{R}^{n_3}$, $b \in \mathbb{R}^{n_2}$, *and* $\boldsymbol{\sigma}$ *denotes the component-wise application of a transition function* $\sigma : \mathbb{R} \to \mathbb{R}$.

In the above definition, g constitutes a feedforward network with one hidden layer which maps the recursively processed sequences to the desired outputs. f defines the recurrent part of the network. The transition function f is given by a feedforward network without hidden layer.[1] Popular choices for σ are the hyperbolic tangent $\tanh$ or the logistic function $\mathrm{sgd}(x) = (1 + \exp(-x))^{-1}$. We now identify situations in which the transition function f of a RecNN is a contraction and g is Lipschitz continuous such that the above results apply. Note the following simple observation:

Lemma 11 *Assume* $\sigma : \mathbb{R} \to \mathbb{R}$ *is Lipschitz continuous with parameter* C. *Then the function* g *as above is Lipschitz-continuous with parameter* $C \cdot n_3 \cdot n_4 \cdot \max_{ij} |(M_1)_{ij}| \cdot \max_{ij} |(M_2)_{ij}|$ *and* f *as above is Lipschitz continuous with parameter* $C \cdot k \cdot n_2 \cdot \max_{ijl} |(W_2^l)_{ij}|$. *For* $\max_{ijl} |(W_2^l)_{ij}| < 1/(C \cdot k \cdot n_2)$ *the transition function* f *is a contraction with respect to* U.

Proof For affine mappings $u \mapsto A \cdot u + b$ with $A \in \mathbb{R}^{n \cdot m}$ and $b \in \mathbb{R}^n$ we find

$$|(A \cdot u + b) - (A \cdot u' + b)|_\infty = \max_i |\sum_j A_{ij}(u_j - u'_j)| \leq n \cdot \max_{ij} |A_{ij}| \cdot |u - u'|_\infty$$

A composition of Lipschitz continuous functions with parameters C_1 and C_2 yields to a Lipschitz continuous function with parameter $C_1 \cdot C_2$. Hence g is Lipschitz continuous with parameter $C \cdot n_3 \cdot n_4 \cdot \max_{ij} |(M_1)_{ij}| \cdot \max_{ij} |(M_2)_{ij}|$ and f is Lipschitz continuous with parameter $C \cdot k \cdot n_2 \cdot \max_{ijl} |(W_2^l)_{ij}|$. $\square$

We can thus directly apply our results to recursive networks, because common activation functions map to a bounded subset $U \in \mathbb{R}^{n_2}$ and they are differentiable, such that the Lipschitz constant can be estimated by

$$|\sigma(x') - \sigma(x'')| \leq \sup_x |\sigma'(x)| \cdot |x' - x''|$$

Thereby, the supremum is to be taken over all images of $M_2 \cdot u + d$ and $W_1 \cdot x + W_2^1 \cdot u_1 + \ldots + W_2^k \cdot u_k + b$. For the popular choice $\sigma = \mathrm{sgd}$ we find $U = (0,1)^{n_2}$ and $C = 0.25$. For $\sigma = \tanh$ we can chose $U = (-1,1)^{n_2}$ and $C = 0.5$. Thus we immediately obtain the result that RecNNs are Markovian if the weights are restricted and the recursive weights fulfill the inequality

$$\max_{ijl} |(W_2^l)_{ij}| < 4/(k \cdot n_2)$$

for $\sigma = \mathrm{sgd}$ and

$$\max_{ijl} |(W_2^l)_{ij}| < 2/(k \cdot n_2)$$

for $\sigma = \tanh$.

1. One could easily take more hidden layers of f and g into account, which would yield similar results.

4. Definite memory tree models can be implemented by small weight RecNNs

We are now interested in the converse direction, i.e. the simulation of Markovian models by small weight RecNNs. Note that it has been shown in (12, 26) that RecNNs can simulate tree automata and thus also Markovian models. These results rely on the assumption that arbitrary weights are available. In particular, the transition function used for the simulation in (12, 26) is not a contraction.

We assume that $\Sigma = \{x_1, \ldots, x_n\}$ is a finite alphabet. The symbols x_i are presented to the network in a unary way in $\mathbb{R}^n$, i.e. entry 1 in dimension i indicates x_i, all other dimensions are set to 0. Denote by Q this encoding and by $\tilde{Q}$ the extensions to trees whereby each label of the tree is encoded in this unary way. σ must be a nonlinear function and we assume that this nonlinearity is of sigmoidal type. I.e., we assume that σ is a strictly monotonously increasing and Lipschitz continuous function with finite limits $a_- = \lim_{x\to-\infty} \sigma(x)$ and $a_+ = \lim_{x\to\infty} \sigma(x)$. Then the following holds:

Theorem 12 *Assume $h : \Sigma_k^* \to Y \subset \mathbb{R}^m$ is Markovian with memory length L and $\epsilon > 0$. Then a RecNN with activation function σ and contractive transition function can be found which implements $g \circ \tilde{f}_y$ such that*

$$|h(t) - g \circ \tilde{f}_y(\tilde{Q}(t))| \leq \epsilon$$

for all $t \in \Sigma_k^*$.

Proof We first construct a transition function $f : \mathbb{R}^m \times \mathbb{R}^{k \cdot n_1} \to \mathbb{R}^{n_1}$ which is contractive and which outputs a unique code for all tree structures of height at most L. We chose $n_1 = n \cdot \sum_{i=0}^{L} k^i$. Note that a tree of height L has at most $\sum_{i=0}^{L} k^i$ nonempty nodes, since there exist at most k^i nodes of depth i for all $i \geq 0$. The idea of the construction is that each node in a tree up to depth L is directly encoded in some position of the output vector of the recursive part of the RecNN. Fix a bijective function $\text{index} : (i, j, k') \mapsto l \in \{1, \ldots, n_1\}$ whereby $i \in \{1, \ldots, n\}$, $j \in \{0, \ldots, L\}$, and $k' \in \{1, \ldots, k^j\}$. We construct a transition function which encodes trees t via an encoding $x \in \mathbb{R}^{n_1}$ such that $x_{\text{index}(i,j,k')} > 0$ iff node number k' of depth j of t contains the element x_i.

We first consider the situation that $\sigma(0) = 0$. Chose the initial context $y = 0$ and the bias $b = 0$. Since σ is Lipschitz continuous, we can find some δ such that the choice of matrix entries of W_1 and W_2^i with absolute value smaller than δ gives a contractive map f. We enumerate the matrix entries of $W_1 \in \mathbb{R}^{n_1 \times n}$ by $(\text{index}(i, j, k'), i')$ and the matrix entries of $W_2^l \in \mathbb{R}^{n_1 \times n_1}$ ($l = 1, \ldots, k$) by $(\text{index}(i, j, k'), \text{index}(i', j', k''))$. We set $(W_1)_{(\text{index}(i,0,1),i)} = \delta/2$ for all i, and we set $(W_2^l)_{(\text{index}(i,j,k'),\text{index}(i,j+1,(l-1)\cdot k^j+k'))} = \delta/2$ for all i, j, k', l. All other matrix entries are 0. Note the following: given a tree t with t_l being its l'th subtree. Then node number k' of depth j of t_l is node number $(l-1) \cdot k^j + k'$ of depth $j+1$ of t. Hence this choice has the effect that the actual label is stored in the places $(i, 0, 1)$, and all other nodes are recursively stored at their respective places up to depth L.

Assume $\sigma(0) \neq 0$. Then we can use the translated function $\sigma_t(x) = \sigma(x) - \sigma(0)$ in the above construction and find some $\tilde{f}_y$ with $f(x, u_1, \ldots, u_k) = \boldsymbol{\sigma}_t(W_1 \cdot x + W_2^1 \cdot u_1 + \ldots + W_2^k \cdot u_k + b)$ and the desired properties. Note that $\tilde{f}_y(t) = \tilde{f}'_z(t) - \boldsymbol{\sigma}(0)$ holds for all trees t whereby $f'(x, u_1, \ldots, u_k) = \boldsymbol{\sigma}(W_1 \cdot x + W_2^1 \cdot u_1 + \ldots + W_2^k \cdot u_k + b - W_2^1 \cdot \boldsymbol{\sigma}(0) - \ldots - W_2^k \cdot \boldsymbol{\sigma}(0))$, $\boldsymbol{\sigma}(0)$ is the vector with components $\sigma(0)$, and $z = \boldsymbol{\sigma}(0)$. Obviously, $\tilde{f}_y$ encodes trees of height L uniquely iff $\tilde{f}'_y$ does, hence this latter function can be chosen in this case.

We thus obtain a unique encoding of the L truncation of trees. It follows immediately from well-known approximation or interpolation results, respectively, for feedforward networks that some h can be found which maps the finite number of outputs of $\tilde{f}_y$ to the desired values (41, 42, 64). h can be chosen as a feedforward network with one hidden layer. $\square$

We can chose arbitrarily small weights within the recursive part instead of $\delta/2$ in the previous proof. Hence arbitrarily small weights of the transition function allow to simulate Markovian models with RecNNs. We can also restrict the weights in the feedforward part if the activation function σ is analytic because of the approximation results as proved in (41). The number of neurons in h might increase in this case.

5. Learnability

Initialization of RecNNs with small weights thus pushes training towards regions with particularly simple dynamics. The resulting networks can be identified with Markovian models with finite memory length. Apart from the fact that this simple dynamics might also be sufficient for interesting learning tasks, one can prove theoretical worst case generalization bounds for RecNNs with small weights. We here first introduce several definitions which allow to formalize the notion of generalization within the context of statistical learning theory.

Assume $\mathcal{F}$ is a function class with domain X and codomain Y with metric $|\cdot|$. [2] A learning algorithm outputs a function $g \in \mathcal{F}$ given a finite set of examples $(x_1, f(x_1)), \ldots, (x_m, f(x_m))$ for an unknown function $f \in \mathcal{F}$. Denote by $\mathcal{P}$ the set of probability measures on X and by P its elements. P^m is the product measure induced by P on X^m. The distance between functions f and g with respect to P is denoted by

$$d_P(f,g) = \int_X |f(x) - g(x)| d_P(x) \, .$$

The aim of training is to find some g such that $d_P(f,g)$ is small. This distance, however, is not available since f and P are not known. Therefore standard learning algorithms often minimize the empirical distance instead. The empirical distance between f and g given $\vec{x} = (x_1, \ldots, x_m) \in X^m$ refers to the quantity

$$\hat{d}_m(f,g,\vec{x}) = \sum_i |f(x_i) - g(x_i)|/m$$

which is obtained if the distance of f and g is evaluated at m given data points. A justification of this principle can be established if the empirical distance is representative of the real distance. I.e., the term 'generalization' means that $\hat{d}(f,g,\vec{x})$ and $d_P(f,g)$ nearly coincide for large enough m uniformly for every f and $g \in \mathcal{F}$ whereby $\vec{x}$ is chosen according to P^m, i.e., the components x_i are independent and identically distributed with respect to P. In mathematical terms, this is the UCED property of $\mathcal{F}$[3]:

Definition 13 *$\mathcal{F}$ fulfills the* distribution independent uniform convergence of empirical distances property (UCED-property)*, if for all $\epsilon > 0$*

$$\lim_{m\to\infty} \sup_{P\in\mathcal{P}} P^m(\vec{x} \mid \exists f, g \in \mathcal{F} : \; |d_P(f,g) - \hat{d}_m(f,g,\vec{x})| > \epsilon) = 0 \, .$$

Since one can think of f as the function to be learned and of g as the output of the learning algorithm, this property characterizes the fact that we can find prior bounds (independent of the underlying probability) on the necessary size of the training set, such that every algorithm with small training error yields good generalization with high probability. [4] The following is a well known fact:

2. We assume in the following that every function or set which occurs is measurable.
3. UCED stands for 'uniform convergence of empirical distances'.
4. Note that the framework tackled by statistical learning theory usually deals with a more general scenario, the so-called agnostic setting (38). There, the function class $\mathcal{F}$ used for learning need not contain the unknown function which is to be learned, and the error is measured by a general loss function. Valid generalization then refers to the property of

Lemma 14 *Finite function classes fulfill the UCED-property.*

Assume Σ is a finite alphabet and $\mathcal{F}$ is the class of functions from Σ_k^* to Σ which can be computed by a definite memory tree model with fixed finite memory length L. Then $\mathcal{F}$ fulfills obviously the UCED-property because the function class is finite. Hence these models can generalize, when provided with enough training data.

Assume $\mathcal{F}$ is the function class which is given by the functions computed by all recursive neural networks as defined in Definition 10 where the dimensionalities n_i and σ are fixed, but the entries of the matrices can be chosen arbitrarily. Then $\mathcal{F}$ does not possess the UCED-property as shown in (31, 34), for example. Hence general recursive networks with no further restrictions do not yield valid generalization in the above sense, unlike fixed length definite memory tree models. Training of general RecNNs may thus need in theory an exhaustive number of patterns for valid generalization and certain underlying input distributions. One particularly 'bad' situation is explicitly constructed in (31) where the number of examples necessary for valid generalization in the worst case increases more than polynomially in the required accuracy. In practical applications, however, such situations might not be found because of computation noise, finite computation accuracy, and further implicit regularization mechanism imposed on learning which are not yet fully understood. Here we offer a theory for initial phases of RecNN training by linking RecNNs with small weights to definite memory machines.

Note that RecNNs with small weights and a finite input set approximately coincide with definite memory tree models with fixed length, where the length depends on the size of the weights. Hence we can conclude that RecNNs with a priori limited small weights and a finite input alphabet possess the UCED property contrarious to general RecNNs with arbitrary weights and finite input alphabet. That means, the architectural bias through the initialization emphasizes a region of the parameter search space where the UCED property can be formally established. We will show in the remaining part of this section that an analogous result can be derived for recursive networks with small weights and arbitrary real-valued inputs. This shows that function classes given by RecNNs with a priori limited small weights possess the UCED property in contrast to general RecNNs with arbitrary weights and infinite precision.

We consider function classes $\mathcal{F}$ with domain X and codomain equal to $[0, 1]$ equipped with the maximum norm $|f - g|_\infty = \sup_{x \in X} |f(x) - g(x)|$. Moreover, we assume that the constant function 0 is contained in $\mathcal{F}$. Then alternative characterization of the UCED property can be found in the literature which relate the generalization ability to the capacity of the function class. Appropriate formalizations of the term 'capacity' are as follows:

Definition 15 *Assume $\mathcal{F}$ is a function class. Let $\epsilon > 0$. The* external covering number $L(\epsilon, \mathcal{F}, |\cdot|_\infty)$ *denotes the size of the smallest external ϵ-cover of $\mathcal{F}$ with respect to the metric $|\cdot|_\infty$. $L(\epsilon, \mathcal{F}, |\cdot|_\infty)$ is infinite if no finite external cover of $\mathcal{F}$ exists.*

The ϵ-fat shattering dimension $\mathcal{VC}_\epsilon(\mathcal{F})$ *of $\mathcal{F}$ is the largest size (possibly infinite) of a set of points $\{x_1, \ldots, x_n\}$ in X which can be shattered with parameter ϵ. Shattering with parameter ϵ means that real values $r_1, \ldots, r_n$ exist, such that for each function $d : \{x_1, \ldots, x_n\} \to \{-1, 1\}$ some function $f \in \mathcal{F}$ exists with $|f(x_i) - r_i| \geq \epsilon$ and $(f(x_i) - r_i) \cdot d(x_i) > 0$.*

Both, the covering number and the fat-shattering dimension measure the richness of $\mathcal{F}$: the number of essentially different functions up to ϵ, or the number of points where a rich behavior can be observed

uniform convergence of empirical means (UCEM) of a class associated to $\mathcal{F}$ via the loss function. However, under several conditions on $\mathcal{F}$ and the loss function, learnability of this associated class can be related to learnability of $\mathcal{F}$ (2, 74). For simplicity, we will only investigate the UCED property of recursive networks with small weights.

within the function class, respectively. Assume $\vec{x} = (x_1, \ldots, x_m) \in X^m$ is a vector. Denote the restriction of $\mathcal{F}$ to $\vec{x}$ by $\mathcal{F}|_{\vec{x}} = \{g : \{x_1, \ldots, x_m\} \to Y \mid \exists f \in \mathcal{F} : \; g(x_i) = f(x_i)\}$. Proofs for the following alternative characterizations of the UCED property can be found in (2, 5, 74):

Lemma 16 *The following characterizations are equivalent for a function class $\mathcal{F}$ with codomain $[0,1]$ which contains the constant function 0:*

- *$\mathcal{F}$ fulfills the UCED-property.*
- $\lim_{m\to\infty} \sup_{P\in\mathcal{P}} \dfrac{E_{P^m}(\log_2 L(\epsilon, \mathcal{F}|_{\vec{x}}, |\cdot|_\infty))}{m} = 0 \quad \forall \epsilon > 0\,.$
- *$\mathcal{VC}_\epsilon(\mathcal{F})$ is finite for every $\epsilon > 0$.*

E_{P^m} denotes expectation with respect to $P^m(\vec{x})$, $\vec{x} = \{x_1, \ldots, x_m\}$. Furthermore, the estimation

$$L(\epsilon, \mathcal{F}|_{\vec{x}}, |\cdot|_\infty) \leq 2 \left(\frac{4\,m}{\epsilon^2}\right)^{d \log_2(12\,m/(d\epsilon))}$$

holds for every $\vec{x} = \{x_1, \ldots, x_m\}$, where $d = \mathcal{VC}_{\epsilon/4}(\mathcal{F})$.

These alternative characterizations will now be used to prove that recursive networks with small weights and arbitrary inputs fulfill the UCED property. Denote by $\mathcal{G} \circ \mathcal{F}$ the class of compositions $\{g \circ f \mid g \in \mathcal{G}, f \in \mathcal{F}\}$ for function classes $\mathcal{F}$ and $\mathcal{G}$ with common domain of $\mathcal{G}$ and codomain of $\mathcal{F}$, respectively. We can relate the UCED property of recursive models to the UCED property of *truncated* recursive models:

Theorem 17 *Assume $C_1 \in (0,1)$ and $C_2 > 0$ are fixed. Assume U is totally bounded. Assume $\mathcal{F}$ is a function class with domain $X \times U^k$ and codomain U such that every function in $\mathcal{F}$ is a contraction with parameter C_1 with respect to U. Assume $\mathcal{G}$ is a function class with domain U and codomain $[0,1]$ such that every function in $\mathcal{G}$ is Lipschitz continuous with parameter C_2. Then the function class $\mathcal{G} \circ \tilde{\mathcal{F}}$ fulfills the UCED property if the function class $\mathcal{G} \circ \tilde{\mathcal{F}}_L$ fulfills the UCED property for every $L \in \mathbb{N}$.*

Assume $\epsilon > 0$. Assume $\vec{t} = (t_1, \ldots, t_m)$ is a vector of trees with labels in U. Because of Theorem 5 we can find some L such that every $g \circ \tilde{f}_y$ in $G \circ \tilde{\mathcal{F}}$ deviates from $g \circ \tilde{f}_y^L$ in $\mathcal{G} \circ \tilde{\mathcal{F}}_L$ by at most $\epsilon/2$ for all inputs. Hence

$$L(\epsilon, \mathcal{G} \circ \tilde{\mathcal{F}}|_{\vec{t}}, |\cdot|_\infty) \leq L(\epsilon/2, \mathcal{G} \circ \tilde{\mathcal{F}}_L|_{\vec{t}}, |\cdot|_\infty) = L(\epsilon/2, \mathcal{G} \circ \tilde{\mathcal{F}}_L|_{T_L(\vec{t})}, |\cdot|_\infty)$$

where $T_L(\vec{t})$ denotes the application of the truncation T_L to every t_i in $\vec{t}$. Hence we can bound the term $L(\epsilon, \mathcal{G} \circ \tilde{\mathcal{F}}|_{\vec{t}}, |\cdot|_\infty)$ for every $\vec{t}$ by $2\left(16m/\epsilon^2\right)^{d\,\log_2(24\,m/(d\epsilon))}$, where $d = \mathcal{VC}_{\epsilon/8}(\mathcal{G} \circ \tilde{F}_L)$ is finite because $\mathcal{G} \circ \tilde{F}_L$ fulfills the UCED property. Hence the quotient $E_{P^m}(\log_2\, L(\epsilon, \mathcal{G} \circ \tilde{\mathcal{F}}|_{\vec{t}}, |\cdot|_\infty)/m)$ becomes arbitrarily small for large m and every $\epsilon > 0$. This is uniform with respect to $P \in \mathcal{P}$. $\square$

As a consequence, standard recursive networks with small weights in the recursive part such that the transition function constitutes a contraction and with limited weights in the feedforward part such that Lipschitz continuity is guaranteed fulfill the UCED property: the function classes $\mathcal{G} \circ \tilde{\mathcal{F}}_L$ from the above proof correspond in this case to simple feedforward networks which are obtained via formally unfolding all recurrent connections according to the given structure. These classes have a finite fat-shattering dimension and therefore fulfill the UCED property for standard activation functions like the hyperbolic tangent (6, 44). Explicit bounds on the fat-shattering dimension can be found in (32).

We would like to go a step further and relate the UCED property or RecNNs with small weights to the UCED property of the class $\mathcal{G} \circ \mathcal{F}$ which does not contain any recurrence. For this purpose, we first extend Lemma 7 which relates the external cover of the class of transition functions to the external cover of the class of induced recursive functions to general recursive models.

Corollary 18 *Assume the same situation as in Lemma 7 is given and $\mathcal{G}$ are functions from U to Y which are Lipschitz continuous with parameter C_1. Assume $C(\mathcal{G})$ is an external cover of $\mathcal{G}$ with parameter $\delta > 0$. Then $\{g \circ \tilde{f}_{u_j} \mid g \in C(\mathcal{G}), f \in C(\mathcal{F}), j = 1, \dots, n\}$ is an external cover of $\mathcal{G} \circ \tilde{\mathcal{F}}$ with parameter $\delta + \epsilon \cdot C_1/(1-C)$ and $\{g \circ \tilde{f}^L_{u_j} \mid g \in C(\mathcal{G}), f \in C(\mathcal{F}), j = 1, \dots, n\}$ is an external cover of $\mathcal{G} \circ \tilde{\mathcal{F}}_L$ with parameter $\delta + \epsilon \cdot C_1/(1-C)$.*

Proof Assume $g \in \mathcal{G}$, $f \in \mathcal{F}$, and $y \in U$ are given. Choose some closest $g' \in C(\mathcal{G})$, $f' \in C(\mathcal{F})$, and $u_j \in U$. Then we find for all $t \in X_k^*$

$$|g'(\tilde{f}'_{u_j}(t)) - g(\tilde{f}_y(t))| \leq |g'(\tilde{f}'_{u_j}(t)) - g(\tilde{f}'_{u_j}(t))| + |g(\tilde{f}'_{u_j}(t)) - g(\tilde{f}_y(t))| \leq \delta + C_1 \cdot \epsilon/(1-C)$$

□

This allows us to relate the UCED property of recursive networks to the UCED property of non-recursive models:

Theorem 19 *Assume $C_1 \in (0,1)$ and $C_2 > 0$ are fixed. Assume U and X are totally bounded sets. Assume $\mathcal{F}$ is a function class with domain $X \times U^k$ and codomain U such that every function in $\mathcal{F}$ is a contraction with parameter C_1 with respect to U. Assume that in addition, every function in $\mathcal{F}$ is Lipschitz continuous with parameter C_1'. Assume $\mathcal{G}$ is a function class with domain U and codomain $[0,1]$ such that every function in $\mathcal{G}$ is Lipschitz continuous with parameter C_2. Then $\mathcal{G} \circ \tilde{\mathcal{F}}$ fulfills the UCED property if $\mathcal{G} \circ \mathcal{F}$ does.*

Proof Note that

$$L(\epsilon, \mathcal{G} \circ \tilde{\mathcal{F}}|_{\vec{s}}, |\cdot|_\infty) \leq L(\epsilon, \mathcal{G} \circ \tilde{\mathcal{F}}, |\cdot|_\infty)$$

for all $\vec{s}$. Because of Corollary 18 we find

$$L(\epsilon, \mathcal{G} \circ \tilde{\mathcal{F}}, |\cdot|_\infty) \leq L(\delta, \mathcal{G} \circ \mathcal{F}, |\cdot|_\infty)$$

for some $\delta > 0$ which depends on ϵ, C_1, and C_2. Because X and U are totally bounded, we can find a finite covering $\vec{xu} := \{(x_1, u_1^1, \dots, u_1^k), \dots, (x_c, u_c^1, \dots, u_c^k)\}$ with parameter $\delta/(3\,C_1' C_2)$ of the set $X \times U^k$. Denote by $N(\delta, \mathcal{H}, |\cdot|_\infty)$ the smallest size of a δ-cover of a function class $\mathcal{H}$ with respect to the metric $|\cdot|_\infty$ such that all functions in the cover are contained in $\mathcal{H}$ itself. Because of the triangle inequality, the estimation

$$N(2\delta, \mathcal{H}, |\cdot|_\infty) \leq L(\delta, \mathcal{H}, |\cdot|_\infty) \leq N(\delta, \mathcal{H}, |\cdot|_\infty)$$

follows immediately for every function class $\mathcal{H}$. Now we find

$$L(\delta, \mathcal{G} \circ \mathcal{F}, |\cdot|_\infty) \leq N(\delta/3, \mathcal{G} \circ \mathcal{F}|_{\vec{xu}}, |\cdot|_\infty)$$

because of the following: choose for $(x, u^1, \ldots, u^k) \in X \times U^k$ a closest $(x_i, u_i^1, \ldots, u_i^k)$ in $\vec{xu}$ and for $g \circ f$ in $\mathcal{G} \circ \mathcal{F}$ a function $g_j \circ f_j$ corresponding to a function in $N(\delta/3, \mathcal{G} \circ \mathcal{F}|_{\vec{xu}}, |\cdot|_\infty)$ such that the distance to $g \circ f$ is minimum on $\vec{xu}$. Then

$$\begin{aligned}
& |g(f(x,u^1,\ldots,u^k)) - g_j(f_j(x,u^1,\ldots,u^k))| \\
\leq\ & |g(f(x,u^1,\ldots,u^k)) - g(f(x_i,u_i^1,\ldots,u_i^k))| \\
& +|g(f(x_i,u_i^1,\ldots,u_i^k)) - g_j(f_j(x_i,u_i^1,\ldots,u_i^k))| \\
& +|g_j(f_j(x_i,u_i^1,\ldots,u_i^k)) - g_j(f_j(x,u^1,\ldots,u^k))| \\
\leq\ & C_1' C_2 \delta/(3C_1'C_2) + \delta/3 + C_1'C_2\delta/(3C_1'C_2) = \delta\,.
\end{aligned}$$

Since the UCED property holds for $\mathcal{G} \circ \mathcal{F}$, we can bound the quantity

$$N(\delta/3, \mathcal{G} \circ \mathcal{F}|_{\vec{xu}}, |\cdot|_\infty) \leq L(\delta/6, \mathcal{G} \circ \mathcal{F}|_{\vec{xu}}, |\cdot|_\infty) \leq 2\left(\frac{4\cdot 36\cdot c}{\delta^2}\right)^{d\,\log_2(12\cdot 6\cdot c/(d\delta))}$$

where c only depends on δ, C_1', and C_2, and $d = \mathcal{VC}_{\delta/24}(\mathcal{G} \circ \mathcal{F})$ is finite because of the UCED property of $\mathcal{G} \circ \mathcal{F}$. Hence the quantity $L(\epsilon, \mathcal{G} \circ \tilde{\mathcal{F}}|_{\vec{s}}, |\cdot|_\infty)$ can be limited by a finite number for fixed ϵ. Therefore, the UCED property of $\mathcal{G} \circ \tilde{\mathcal{F}}$ follows. □

Thus the UCED property of recursive tree models with contractive transition function and totally bounded input sets directly follows from the UCED property of the corresponding non-recursive function class. These results show that small weights constitute one sufficient sufficient condition for the generalization ability of RecNNs in the PAC sense. Alternative regularization schemes could be applied such as finite precision of the computation or explicit discretization of interior states, restriction to stable dynamical behavior, or data dependent properties (35).

6. Experiments

We are interested in the effect of the small weight restriction on concrete training scenarios. We have seen that only Markovian tasks can be solved by small weight RecNNs. The question now occurs whether the restriction of RecNNs to small weights leads to better solutions or better generalization ability of the networks if used in practice for Markovian tasks. To test this hypothesis we consider two simple learning problems. To state these problems, we first extend recursive models as introduced in Definition 2 to the slightly more general data structure of so-called labeled directed positional acyclic graphs (DPAGs) with fan-out k and one supersource.

DPAGs constitute directed acyclic graphs with labeled vertices with limited fan-out k. In addition, a positioning of all successors of each vertex is fixed, i.e. each successor of a given node is assigned a value in $\{1, \ldots, k\}$ which is different for any two successors of the same node. Moreover, a supersource exists, i.e., each node can be reached by a path starting in one common node. DPAGs can be interpreted as tree structures with fan-out k and possibly shared nodes: the supersource constitutes the root of the tree and the successors of the supersource together with their positioning point to the (at most k) subtrees of the root. Thereby, nodes of the DPAG with more than one predecessor can be considered 'shared nodes' of the tree, i.e., they occur at more than one position in the tree. Turning a DPAG into this associated tree allows us to also process DPAGs by recursive models. The recursive function defined in Def. 2 cannot distinguish between a DPAG (including shared nodes) and the correspondent tree structure (with different nodes). However, the internal representation of trees as DPAGs has the advantage that the recursion is to be evaluated only once for every shared node. This allows us to also deal with deep structures with high connectivity efficiently if they contain shared nodes.

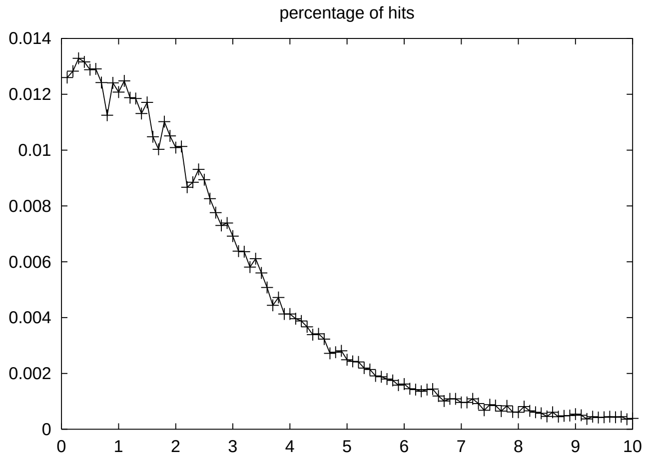


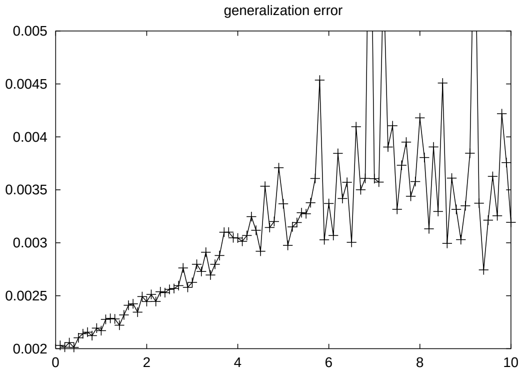
Figure 3: Percentage of randomly generated networks such that the error on the training set is smaller than 0.13 for S1 in dependence on the size of the interval in which the recurrent weights are initialized.

weight range	0.1-1.0	1.1-3.0	3.1-5.0	5.1-10.0
training set	0.1077	0.1066	0.1091	0.1119
test set	0.1080	0.1071	0.1102	0.1136

Table 1: Mean training and test set error for S1. The mean value over the indicated intervals is given.

The first task, referred to as S1 in the following, is a simple artificial Markovian problem. We consider DPAGs with fan-out 2 and real-valued labels. For the training set and the test set we generate independently two DPAGs with $5 \cdot 200$ nodes each with real-valued labels. The labels are chosen randomly as $x + \eta$ whereby $x \in \{0.1, 0.9\}$ and η is uniform noise with support $[-0.1, 0.1]$. The two DPAGs have depth 200 and their connectivity is determined as follows: In both DPAGs and for each depth $i \leq 200$, five nodes of depth i exist and each node of depth $i < 200$ is connected to two randomly chosen successor nodes of depth $i + 1$. Since all nodes are internally shared, efficient evaluation of a RecNN is possible. The task is to map each node of the given DPAG with label $x_1 + \eta_1$ and its successors with labels $x_2 + \eta_2$ and $x_3 + \eta_3$, respectively, to the output $o = 0.2 \cdot x_1 \cdot x_2 + 0.2 \cdot x_1 \cdot x_3 + 0.4 \cdot x_1 + \eta$, whereby η is uniform noise with support $[-0.1, 0.1]$ and the value o is in addition clipped to the interval $[0.05, 0.95]$ such that it lies in the codomain of the logistic activation function. Since the output depends only on the current node and its two children, the task is clearly Markovian. The overall number of different training (test) patterns (counting shared nodes only once) is thus 1000. A default classification of the trees to the mean value on the training set leads to the (linear) error 0.246 on the training set and 0.251 on the test set.

We consider the possibility to learn the task S1 with a small sigmoidal recursive network with different range of recursive weights. The network implements a function $g \circ \tilde{f}_y$ whereby both, $f : \mathbb{R}^{1+2\cdot n} \to \mathbb{R}^n$, and $g : \mathbb{R}^n \to \mathbb{R}$ are chosen as sigmoidal network without hidden layer, i.e. they are given by the combination of an affine mapping and the sigmoidal transfer function sgd. We refer to the parameters of g as output weights in the following. The parameters of f decompose



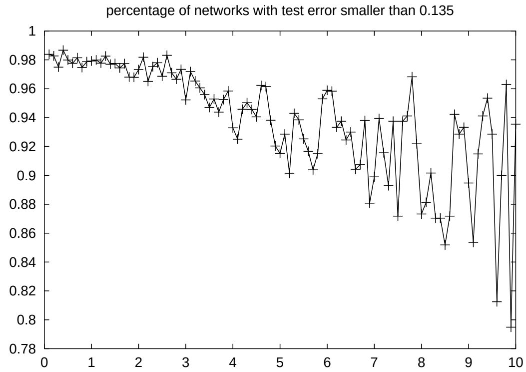


Figure 4: Mean absolute distance of the training error and test error for S1 in dependence of the radius B of the interval used for initializing the weights (left). Percentage of networks with training set error smaller than 0.13 and test set error smaller than 0.135 in dependence of B (right).

into the input weights, i.e. the parameters weighting the current input label given by the DPAG, and the recursive weights, i.e. the parameters weighting the contribution of recursively computed values from the previous step. The size of the recursive weights determines whether a function f which is contractive with respect to U results. The dimensionality n is chosen as 2 in our experiments. To separate effects of RecNN training such as the problem of long-term dependencies and potential ill-conditioning of the Jacobi matrix from the effect of small weights, we did not use an optimization algorithm. Rather, we sample the space of potential solutions. Thus, 'training' is performed solely by generating networks randomly with different range of recursive weights, and rejecting the networks with too large an error on the training set. We then evaluate the error on the test set and compute the generalization error, i.e. the absolute distance of the training error and the test error. To separate positive effects of small weights for standard feedforward networks (4) from the effect of small recursive weights, we initialize the output weights (the parameters of g) randomly in the interval $[-15, 15]$ for all settings. The input weights of f are initialized in the interval $[-2, 2]$. The recursive weights of f are randomly initialized in the interval $[-B, B]$, whereby B is varied from 0.1 to 10. Note that for $B < 1$ the transition function f necessarily forms a contraction (with respect to the recursive argument). For larger B, the chosen weights might still lead to contractive mappings, but there is no guarantee for this. We randomly generated 100000 networks in this way.

As mentioned earlier, for S1, the default error by constant mapping to the mean output of the training set is 0.246 for the training set and 0.251 for the test set. In our experiments, we reject all randomly generated networks for which the error on the training set is larger than 0.13 which is about half of the default error. The following behavior could be observed: the number of randomly generated networks which have a small training error becomes smaller for increasing B; the number decreases from 1260 hits for $B = 0.1$ to 39 hits for $B = 10$. The percentage of randomly generated networks which have a small training error is depicted in Fig. 3. Thereby, the range around $B = 0.5$ yields the maximum number where the weights are large enough to allow the representation of the task but still small enough to ensure a contraction. Due to a comparably small number of hits for $B > 2$, a large variance of the observed quantities such as the training error and test error is observed in this range. In Tab. 1 we report the mean training and test error *averaged* for different weight ranges. The reported values are averages of the mean training and test errors for the corresponding weight range. The training and test error become larger for large weights, whereby the differences are small (but significant) due to the task. Thereby, also small weights allow to solve the task. Slightly larger weights around 2, which already allow non-contractive transitions but where still most randomly initialized networks will be contractive, give the best results. The generalization ability of RecNNs refers to the distance of the mean training error and test error. The generalization error is depicted in Fig. 4. It becomes larger if the recurrent weights are initialized in a larger range. Moreover, the percentage of RecNNs with small training error which have a comparably large test error increases for large weights. Fig. 4 depicts the number of networks with training error smaller than 0.13 and test error not larger than 0.135. This number decreases if B is large. However, as already observed in the case of RNNs (36), the generalization error increases smoothly with increasing size of the weights and the test error is still much smaller than the default error obtained by constant prediction. Thus regularization also takes place for large weights in this task such that the generalization ability is still acceptable for RecNNs initialized with large weights. Nevertheless, in this Markovian task, when sampling randomly, small weights provide more solutions, on average better solutions, and a better mean generalization error.

As a second example we consider the policemen data set as introduced in (28, 29, 30).[5] Thereby, images which are composed of graphical objects are represented as directed acyclic graphs. The basic primitives (circles, squares, triangle, etc.) are represented by their center of gravity. One could, of course, use more detailed representation at the labels; but this crude representation will already be sufficient to demonstrate the Markovian bias. Thus labels are elements in $\mathbb{R}^2$. Two nodes in the graph are connected if their respective primitives are connected or overlap in the image. The connection structure of the graphs is sparse, only few nodes having more than one or two children. The orientation of edges is thereby determined by the order in which the primitives are reached when using a scan-line algorithm from the lower left to the upper right corner of the image. The positioning of successors of nodes is also determined in this way. This procedure results in DPAGs with up to 6 children per node and single supersources for each object for the policemen data set. Fig. 5 shows an example picture and its graph representation. Since DPAGs can be transferred to trees as beforehand, recursive networks are applicable.

The learning task is to classify objects as members of three classes: ships, houses, and policemen. We use 500 training examples and 2000 test examples, whereby the classes are uniformly distributed on the test set such that a constant classification gives the error 0.66 on the test set. On the training set, ships have a slightly higher probability such that the default classification to this class leads to the error 0.62. Graphs which represent policemen tend to be high but shallow with maximum fan-out 3 and depth 4 or 5. The number of nodes is between 9 and 11. Houses and ships have a higher fan-out in particular for the root node. The maximum fan-out for houses is 5, for ships it is 6. Ships and houses are represented by flat graphs, the depth being between 1 and 3 for ships and 2 or 3 for houses. The number of nodes ranges from 3 to 13 for ships and 4 to 7 for houses. In addition, the root label of the objects contains valuable information since the center of gravity of the supersource of these groups tends to be located at different positions. It can be expected that the task can be solved based on a Markovian model – the fan-out of root nodes is a good indicator of the class and the root label gives additional valuable information – as well as models which look at the whole structure – the depth of the tree is an alternative relevant feature.

We consider a similar network architecture as in the first experiment: the network implements a function $g \circ \tilde{f}_y$ whereby g and f are sigmoidal networks without hidden layer. The codomain of

5. We would like to thank Markus Hagenbuchner for providing the data set.

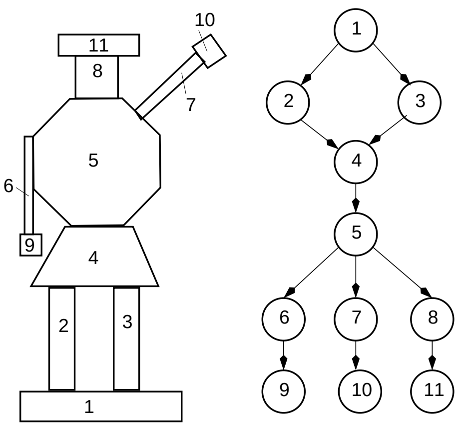
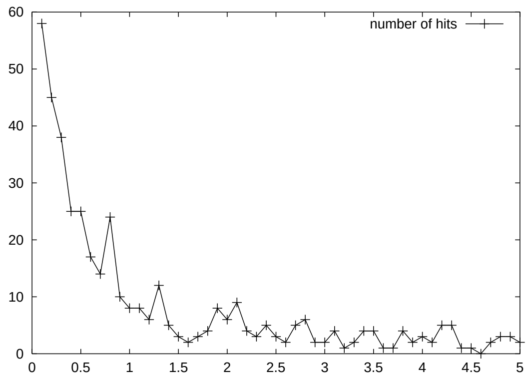


Figure 5: An example policemen and its graph representation obtained by a scan-line algorithm as described in (29). Since the graph is acyclic, directed, and it has one supersource, it can be interpreted as a tree with shared nodes.

f has dimensionality 2, as beforehand. The codomain of g is here also chosen as 2 and the three classes which are to be predicted are encoded as $(0,0)$ for ships (class1), $(0,1)$ for houses (class2), and $(1,0)$ for policemen (class3). A point is assumed to be classified correctly, if the output deviates for both dimensions by at most 0.4 from the desired value. The error consists in the sum of all misclassifications divided by the pattern set size. As beforehand, we evaluate the generalization ability of recursive networks with different size weights based on sampling. Thus we generate 100 random recursive transition functions whereby the recursive weights are chosen in the interval $[-B, B]$. The node labels are scaled to $[0, 0.5]^2$ and the input weights are initialized in $[-5, 5]$. The input biases are fixed to 0. The output weights and biases are chosen in the interval $[-6, 6]$. Since random sampling with only 100 networks does not find networks with acceptable classification accuracy and the training set is too large, to sample large parts of the weight space, we change the procedure from the previous experiment to obtain the output function g as follows: we scale the output activation of the input driven recursive function to the interval $[0, 1]^2$ based on the training set. I.e., the input for g is $\text{norm}(\tilde{f}_y(t))$, norm being a fixed linear scaling function determined on the training set. Thus inputs for g are within a reasonable range. Without scaling, the difference of inputs produced by randomly initialized recursive functions $\tilde{f}_y$ for different graph structures is very small such that no solution could be found.[6] In addition, we randomly generate up to 10000 guesses g for each recursive function $\tilde{f}_y$ until the first solution with training error at most 0.32 is found.

The limit 0.32 is chosen such that a solution has to classify about two classes correctly. Since policemen differ from ships and houses with respect to the fan out of the root as well as the height of trees, it can be expected that arbitrary size networks can find solutions which classify at least two classes correctly. The number of hits (maximum 100) is shown in Fig. 6 in dependence of the weight range. For small weights, half of the randomly initialized recursive parts lead to a classification with accuracy better than 0.32. This number decreases rapidly with increasing size B, yielding only a few or even no solution for large weights. Correspondingly, the resulting mean error on the training and test set is subject to large variance and it is only depicted for $B < 2$ in Fig. 6. Obviously, the achieved

6. The scaling can be considered part of the output function g. Note that g is still Lipschitz continuous.



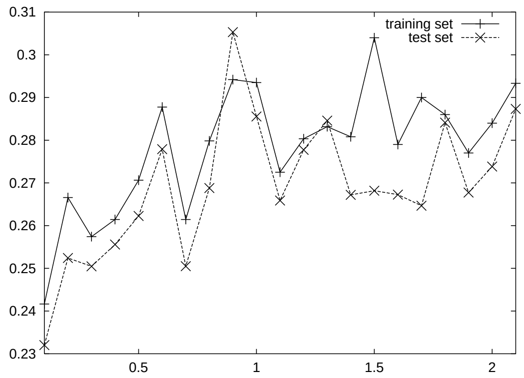


Figure 6: Hits for the policemen task when initializing with different size weights (left). Mean training and test error for networks on the policemen dataset whereby the training error has to be smaller than 0.32 (right).

results are best if $B = 0.1$. The mean training error averaged over all results for $B = 2, \ldots, 5$ is 0.2852, the mean test error averaged over this weights range is 0.2656. Note that the error is slightly smaller on the test set compared to the training set since the three classes are balanced in the test set whereas ships form the majority in the training set. The mean generalization error for values $B = 2, \ldots, 5$ is 0.0224. The mean generalization error for values $B \leq 2$ is 0.0175, whereby $B = 0.1$ leads to a generalization error of 0.01672. Thus the generalization ability decreases for larger weights as in the previous experiment. More importantly, the number of recursive functions which allow a successful implementation of the task drops rapidly for $B \geq 1$.

Thus small weight RecNNs can solve this Markovian task better than large weight RecNNs. This is manifested in the geometry of the activations of randomly initialized transition functions $\tilde{f}_y$ with different size weights. Fig. 7 shows the output (normalized to $[0, 1]^2$) of the input driven recursive transition function initialized with different size weights for all 500 training points and the first 500 test points. For $B \geq 0.33$, the transition function need no longer be contractive. However, since we deal with sparse and relatively shallow trees, non-contractive transition functions can here be observed only for larger B. For small weights, clusters within the space can be found as they have already been observed for standard RNNs with small weights (48, 49). The clusters are meaningful with respect to our learning task: class 3 (policemen) is separated from the other two classes for all plots with $B < 1$. Note that policemen can be distinguished from ships and houses because of a smaller fan-out of the root. Ships and houses also form clusters, but they partially overlap also for small weights since the fan-out of the root and the root label are similar. Class 2 (houses) is distributed among several (often three) quite compact clusters, whereas class 1 (ships) occupies a larger range. As B becomes larger, the clusters tend more and more to overlap. Finally, for large values of B the system 'collapses': activations can merely be found at saturated regions close to the borders and the points arrange according to their global structure. Thus the points similar in structure are mapped to almost the same points and the space is only sparsely used. Thereby, the representation of policemen still tends to form a cluster which is separated from the other points – policemen have shallow and high trees in contrast to the other classes – such that also large weights allow a comparably good classification of the data.

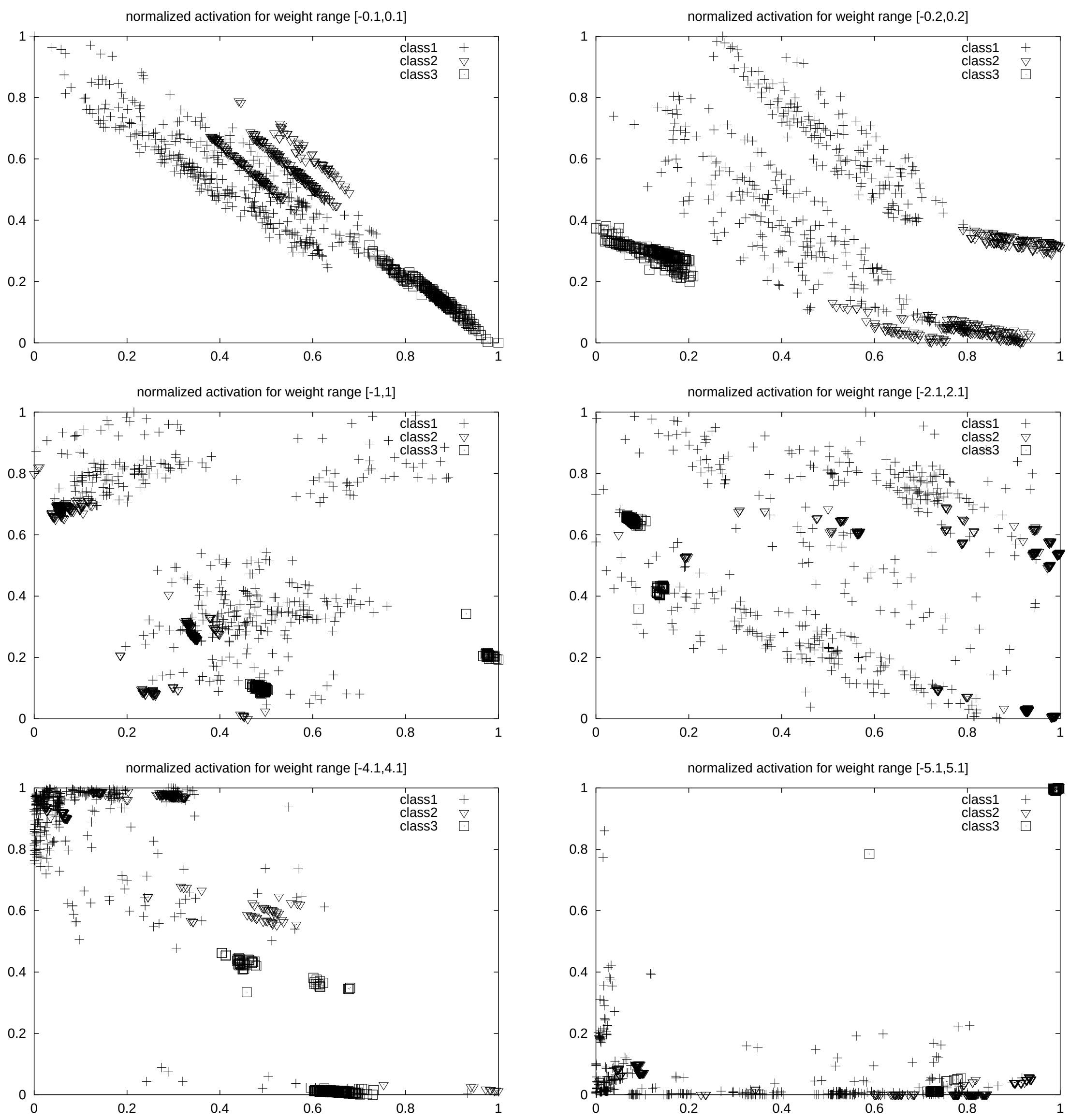


Figure 7: Output of the recursive part of a recursive network for the policemen dataset when initializing recursive weights in different ranges. For small weights, meaningful clusters can be observed which allow a solution of the problem based on this representation. For large weights, the clusters start to overlap and finally collapse.

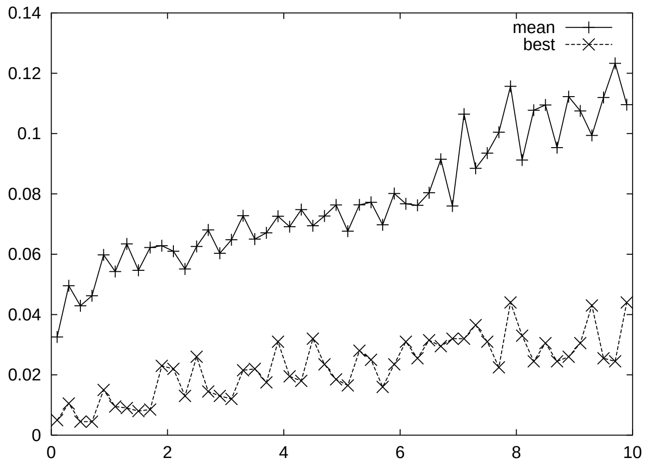


Figure 8: Result of a 3-nearest neighbor classification based on the activations computed by randomly initialzied recursive networks on the test set.

We conclude with a hint on the possibility to build a classifier based on randomly generated recursive transition functions: a simple 3-nearest neighbor classifier constructed on top of the outputs of a randomly initialized recursive transition function $\tilde{f}_y$. The outputs are thereby normalized to $[0,1]^2$ and the recursive weights are initialized in the weight range $[-B, B]$, as beforehand. The prototypes of the classifier are provided by the activations on the training set. The test set error is depicted in Fig 8. Thereby, the mean value over 50 random initialization of the recursive network with different recursive weights in $[-B, B]$ and the respective best classification are shown. Small weights achieve a mean misclassification rate smaller than 4%. The best solution for small weights yields only 10 errors on the test set (0.5%). In the mean, we find 3.3% wrong classification for small weights and 11% wrong classification for large weights. The dataset is thus an easy Markovian task and can be solved with very small recursive networks. For comparison, a prototype based learning algorithm which directly works on graph structure and which has been applied for a (smaller) problem generated by the same source yields the accuracy 0.25% (15). Small *random* weight RecNNs already allow to get comparably good results. *Trained* RecNNs allow to solve the task with 100% accuracy. For a profound evaluation whether randomly initialized recursive networks can be used as appropriate feature extractors for classification, however, experiments on more complex tasks are clearly necessary.

7. Conclusions

We have shown an equivalence of recursive neural networks initialized with small weights and definite memory tree models. Thus a restriction to small weights reduces the capacity of recursive networks, which in general can simulate tree automata (26, 32), to only finite memory models. Thereby also small weight RecNNs with input trees with *real-valued* labels can be simulated by Markovian models with only a *finite* alphabet. This theoretical investigation forms a counterpart for general structure processing networks to the results reported in (36). We can thus characterize the dynamical behavior of RecNNs when initialized with small weights and during the initial phase of training as Markovian.

The theoretical investigation has been accompanied by experimental results for two specific Markovian tasks. We sampled networks with different sizes of the recursive weights and found that Markovian tasks can be solved with small weights more easily in the sense that the percentage of random networks which can solve these tasks up to a priorly fixed accuracy is higher for small weights. This refers to two aspects: the number of networks with a small error on the training set is larger; among these networks, the number of networks which also generalize to unseen data is larger. This experimental observation is supported by a theoretical counterpart on the PAC learnability of RecNNs with small weights. Unlike general RecNNs (with arbitrary weights and infinite precision), small weight RecNNs have the uniform convergence of empirical distances property. Thus a theoretical guarantee that valid generalization can be found for RecNNs also for arbitrarily large input trees and real-valued labels (with arbitrary precision) can be established. Naturally, generalization can also be observed for trained RecNNs with large weights in practice and it can be explained e.g. by the effects of noise on the computation or the specific shape of the data which limits the effective fat-shattering-dimension of recursive networks (32, 54, 55). The results of this article provide one sufficient condition for the generalization ability for the initial phase of training which holds regardless of the accuracy of the computation and the shape of the data.

The Markovian bias of RecNNs manifests itself in the activation of randomly initialized input driven recursive networks. It is well known that the activation of input driven RNNs initialized with small weights leads to clusters even before training (48, 49). Small weight RNNs can be interpreted as iterated function systems similar to recurrent encoding mechanisms used e.g. in fractal prediction machines (70). We demonstrated the existence of meaningful clusters for input driven recursive networks on the policemen dataset. When the weights become larger, the clusters start to overlap. Finally an arrangement which uses only small parts of the activation space is found which corresponds to an operation of the network in the saturated areas of the logistic function. A comparison of Markovian models and RNNs can be found in (68, 69). Here RNNs are compared with randomly initialized RNNs enlarged by a statistical classifier based on the RNN's activation. In particular, the interior representation obtained by the RNNs is not trained during the experiments. As pointed out in (68, 69), solutions of Markovian training tasks can already be found without further adaptation of the interior activations. Our experiments give a hint that a similar situation can be observed also for RecNNs. For non-Markovian tasks, of course, trained RNNs and RecNNs are superior, though the effective gain depends on the task.

References

[1] Alon, N., Dewdney, A.K., and Ott, T.J. (1991). Efficient simulation of finite automata by neural nets. *Journal of the ACM* 38(2):495-514, 1991.

[2] Anthony, M., and Bartlett, P.L. (1999). *Neural Network Learning: Theoretical Foundations.* Cambridge University Press.

[3] Baldi, P., Brunak, S., Frasconi, P., Pollastri, G., and Soda, G. (2001). Bidirectional dynamics for protein secondary structure prediction. R. Sun, C.L. Giles (eds.), *Sequence Learning: Paradigms, Algorithms, and Applications*, pp. 80-104, Springer.

[4] Bartlett, P.L. (1997). For valid generalization, the size of the weights is more important than the size of the network. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, Volume 9. The MIT Press, pp. 134-141.

[5] Bartlett, P.L., Long P., and Williamson, R. (1994). Fat-shattering and the learnability of real valued functions. In *Proceedings of the 7th ACM Conference on Computational Learning Theory*, pp. 299-310.

[6] Baum, E.B., and Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1(1):151-165.

[7] Bengio, Y. and Frasconi, P. (1996). Input/output HMMs for sequence processing. *IEEE Transactions on Neural Networks*, 7(5):1231-1249.

[8] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157-166.

[9] Bianucci, A.M., Micheli, A., Sperduti, A., and Starita, A. (2000). Application of cascade correlation networks for structures to chemistry. *Journal of Applied Intelligence*, 12:117-146.

[10] Bodén, M. and Wiles, J. (2002). On learning context free and context sensitive languages. *IEEE Transactions on Neural Networks*, 13(2):491-493.

[11] Bühlmann, P., and Wyner, A.J. (1999). Variable length Markov chains. *Annals of Statistics*, 27:480-513.

[12] Carrasco, R.C., and Forcada, M.L. (2001). Simple strategies to encode tree automata in sigmoid recursive neural networks. *IEEE Transactions on Knowledge and Data engineering*, 13(2):148-156.

[13] Christiansen, M.H., and Chater,N. (1999). Towards a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23:157-205.

[14] Clouse, D.S., Giles, C.L., Horne, B.G., and Cottrell, G.W. (1997). Time-Delay Neural Networks: Representation and Induction of Finite State Machines. *IEEE Transactions on Neural Networks*, 8(5):1065.

[15] Cordella, L.P., Foggia, P., Sansone, C., and Vento, M. (2002). Learning structural shape descriptions from examples. *Pattern Recognition Letters* 23:1427-1437.

[16] Diligenti, M., Frasconi, P., and Gori, M. (2003). Hidden tree Markov models for document image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4):519-523.

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

[18] Frasconi, P., Gori, M., Maggini, M., and Soda, G. (1995). Unified integration of explicit rules and learning by example in recurrent networks. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):313-332.

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

[20] Funahashi, K., and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 12:831-864.

[21] Gésceg, F. and Steinby, M. (1984). *Tree Automata*, Akadémiai Kiadó, Budapest.

[22] Gésceg, F. and Steinby, M. (1996). Tree languages. Rozenberg, G., Salomaa, A.(eds.), *Handbook of Formal Languages*, vol.3, pages 1-68, Springer.

[23] Giles, C.L., Lawrence, S., and Lin, T. (1995). Learning a class of large finite state machines with a recurrent neural network. *Neural Networks*, 8(0):1359-1365.

[24] Giles, C.L., Lawrence, S., and Tsoi, A.C. (1997). Rule inference for financial prediction using recurrent neural networks. *Proceedings of the Conference on Computational Intelligence for Financial Engineering*, pp.253-259, New York City, NY.

[25] Goller, C. (1997). *A connectionist approach for learning search control heuristics for automated deduction systems*. PhD thesis, Technische Universität München.

[26] Gori, M., Küchler, A., and Sperduti, A. (1998). On the implementation of frontier-to-root tree automata in recursive neural networks. *IEEE Transactions on Neural Networks*, 10, 1999.

[27] Guyon, I., and Pereira, F. (1995). Design of a linguistic postprocessor using variable memory length Markov models. *Proceedings of International Conference on Document Analysis and Recognition*, pp.454-457, Montreal, Canada, IEEE Computer Society Press.

[28] Hagenbuchner, M. *The Traffic Policemen Benchmark Software Package*, http://artificial-neural.net

[29] Hagenbuchner, M., Gori, M., Bunke, H., Tsoi, A.C., and Irniger, C. (2003). Using attributed plex grammars for the generation of image and graph databases. *Pattern Recognition Letters* 24(8): 1081-1087.

[30] Hagenbuchner, M., Sperduti, A., and Tsoi, A.C. (2003). A Self-Organizing Map for Adaptive Processing of Structured Data *IEEE Transactions on Neural Networks* 14(3):491-505.

[31] Hammer, B. (1999). On the learnability of recursive data. *Mathematics of Control, Signals, and Systems*, 12:62-79.

[32] Hammer, B. (2000). *Learning with recurrent neural networks*. Springer Lecture Notes in Control and Information Sciences. Springer.

[33] Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1-4): 107-124.

[34] Hammer, B. (1997). On the generalization of Elman networks. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicaud, editors, *Artificial Neural Networks – ICANN'97*. Springer, pp. 409-414.

[35] Hammer, B. and Steil, J.J. (2002). Perspectives on learning with recurrent neural networks. In M. Verleysen, editor, *ESANN'02*, pages 357-368. D-side publications.

[36] Hammer, B. and Tiňo, P. (2003). Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, 15(8):1897-1929.

[37] Haussler, D. (1999). *Convolution kernels on discrete structure*. Technical report, UC Santa Cruz.

[38] Haussler, D. (1992). Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100:78-150.

[39] Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735-1780.

[40] Hopcroft, J.E. and Ullmann, J.D. (1979). *Introduction to Automata Theory, Languages, and Computation*. AddisonWesley Publishing Co., Reading Massachusetts.

[41] Hornik, K. (1993). Some new results on neural network approximation. *Neural Networks*, 6:1069-1072.

[42] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359-366.

[43] Jaakkola, T., Diekhans, M., and Haussler, D. (2000). A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7:95–114.

[44] Karpinski, M., and Macintyre, A. (1995). Polynomial bounds for the VC dimension of sigmoidal neural networks. In *Proceedings of the 27th annual ACM Symposium on the Theory of Computing*, pp. 200-208.

[45] Kilian, J. and Siegelmann, H.T. (1996). The dynamic universality of sigmoidal neural networks. *Information and Computation*, 128:48-56.

[46] Kohavi, Z. (1978). *Switching and finite automata*. McGraw-Hill.

[47] Koiran, P., and Sontag, E.D. (1997). Vapnik-Chervonenkis dimension of recurrent neural networks. In *Proceedings of the 3rd European Conference on Computational Learning Theory*, pp. 223-237.

[48] Kolen, J.F. (1994). Recurrent networks: state machines or iterated function systems? *Proceedings of the 1993 Connectionist Models Summer School,* pp.203-210, Lawrence Erlbaum Associates, Hilsdale, NJ.

[49] Kolen, J.F. (1994). The origin of clusters in recurrent neural state space. *Proceedings of the 1993 Connectionist Models Summer School,* pp.508-513, Lawrence Erlbaum Associates, Hilsdale, NJ.

[50] Kremer, S.C. (2001). Spatio-temporal connectionist networks: A taxonomy and review. *Neural Computation*, 13(2):249-306.

[51] Küchler, A. and Goller, C. (1996). Inductive learning symbolic domains using structure-driven neural networks. In G. Görz and S. Hölldobler, editors, *KI-96: Advances in Artificial Intelligence*. Springer, Berlin.

[52] Leslie, C., Eskin E., and Noble, W. (2002). The spectrum kernel: A string kernel for SVM protein classification. *Proc. Pacific Symposium on Biocomputing*, pages 564-575.

[53] Lodhi, H., Shawe-Taylor, J., Cristianini, N., and Watkins, C.J.C.H. (2000). Text classification using string kernels. *NIPS*, pages 563-569.

[54] Maass, W., and Orponen, P. (1998). On the effect of analog noise in discrete-time analog computation. *Neural Computation*, 10(5):1071-1095.

[55] Maass, W., and Sontag, E.D. (1999). Analog neural nets with Gaussian or other common noise distributions cannot recognize arbitrary regular languages. *Neural Computation*, 11:771-782.

[56] De Mauro, C., Diligenti, M., Gori, M., and Maggini, M. (2003). Similarity learning for graph-based image representations. *Pattern Recognition Letters* 24(8):1115-1122.

[57] Omlin, C.W., and Giles, C.L. (1996). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 43(6):937-972.

[58] Omlin, C.W., and Giles, C.L. (1996). Stable encoding of large finite-state automata in recurrent networks with sigmoid discriminants. *Neural Computation*, 8:675-696.

[59] Pollastri, G., Baldi, P., Vullo, A., and Frasconi, P. (2002). Prediction of protein topologies using GIOHMMs and GRNNs. In: *Advances of Neural Information Processing Systems 2002*.

[60] Ron, D., Singer, Y., and Tishby, N. (1996). The power of amnesia. *Machine Learning*, 25:117-150.

[61] Siegelmann, H.T., and Sontag, E.D. (1994). Analog computation, neural networks, and circuits. *Theoretical Computer Science*, 131:331-360.

[62] Siegelmann, H.T., and Sontag, E.D. (1995). On the computational power of neural networks. *Journal of Computer and System Sciences*, 50:132-150.

[63] Sontag, E.D. (1998). VC dimension of neural networks. In C. Bishop, editor, *Neural Networks and Machine Learning*. Springer, pp. 69-95.

[64] Sontag, E.D. (1992). Feedforward nets for interpolation and classification. *Journal of Computer and System Sciences*, 45:20-48.

[65] Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714-735.

[66] Sturt, P., Costa, F., Lombardo, V., and Frasconi, P. (2003), Learning first-pass structural attachment preferences with dynamic grammars and recursive neural networks. *Cognition*, 88(2):133-169, 2003.

[67] Sun, R. (2001), Introduction to sequence learning. R. Sun, C.L. Giles (eds.), *Sequence Learning: Paradigms, Algorithms, and Applications*, pp. 1-10, Springer.

[68] Tiňo, P., Čerňanský, M., and Beňušková, L. (2002). Markovian architectural bias of recurrent neural networks. P. Sinčák, J. Vaščák, V. Kvasnička and J. Pospichal (eds.), *Intelligent Technologies - Theory and Applications. Frontiers in AI and Applications* – 2nd Euro-International Symposium on Computational Intelligence, pp. 17-23, IOS Press, Amsterdam.

[69] Tiňo, P., Čerňanský, M., and Beňušková, L. (2003). Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, accepted.

[70] Tiňo, P., and Dorffner, G. (2001). Predicting the future of discrete sequences from fractal representations of the past. *Machine Learning*, 45(2):187-218.

[71] Tiňo, P., and Hammer, B. (2002). Architectural bias of recurrent neural networks - fractal analysis. J. R. Dorronsoro (ed.), *Int. Conf. on Artificial Neural Networks (ICANN 2002)*, pp. 1359-1364, Springer.

[72] Tiňo, P., and Hammer, B. (2003). Architectural bias of recurrent neural networks - fractal analysis. *Neural Computation*, 15(8):1931-1957.

[73] Tiňo, P., and Šajda, J. (1995). Learning and extracting initial Mealy machines with a modular neural network model. *Neural Computation*, 4:822-844.

[74] Vidyasagar, M. (1997). *A Theory of Learning and Generalization*. Springer.

[75] Vullo, A. and Frasconi, P. (2003). A recursive connectionist approach for predicting disulfide connectivity in proteins. *Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing (SAC 2003)*, Melbourne, FL.

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

[77] Yao, Y., Marcialis, G.L., Pontil, M., Frasconi, P., and Roli, F. (2003). Combining flat and structured representations for fingerprint classification with recursive neural networks and support vector machines. *Pattern Recognition*, 36(2): 397-406.