On the Approximation Capability of Recurrent Neural Networks

Barbara Hammer
Department of Mathematics/Computer Science
University of Osnabrück, Albrechtstraße 28, D-49069 Osnabrück,
e-mail: hammer@informatik.uni-osnabrueck.de

Abstract: The capability of recurrent neural networks to approximate functions from lists of real vectors to a real vector space is considered.
We show the following results: From approximation results in the feedforward case follows that any measurable function can be approximated with a recurrent network arbitrarily well in probability. For lists with entries from a finite alphabet only one neuron is needed in the recurrent part of the network.
On the other hand there exist computable functions that cannot be approximated in the maximum norm. This is valid even for functions on unary inputs if they have unlimited range. For limited range this is valid for inputs on an alphabet with at least two elements. But if only the length of a sequence is relevant i.e. sequences with entries from a unary alphabet are considered, any function with limited range can be approximated in the maximum norm. As a special case a sigmoidal recurrent network with one irrational weight can compute any function on unary inputs in linear time.

1 Introduction

Recurrent networks are discrete time dynamic systems dealing with sequences of input vectors. They can be used in different ways:

They can serve as a computation function which computes on words, i.e. lists with elements of a finite alphabet. These are accepted or rejected after some time if a specified output unit returns a value corresponding to 'yes' resp. 'no'. The fact that a computation does not terminate can be specified in this formalism as well. As a computation model recurrent neural networks are Turing universal [12], for special activation functions they can compute any (even non recursive) function [11]. These results demonstrate the power of recurrent networks from a theoretical point of view. However, this is purely theoretical because neural computers does not exist up to now.

A second possibility is to use recurrent networks for the approximation of dynamic systems e.g. in control theory. Here a nonlinear plant which maybe specified by its input-output behavior is modeled by a neural network. In [13] it is shown that any dynamic system with continuous transition function can be approximated on a compact input set and time interval because the transition function can be approximated by a standard network arbitrarily well.

A third view is to consider recurrent networks simply as functions that map an input sequence to a real value. The data may be produced by a dynamic system but it can be in principal the output of any function on sequences. In practical applications this scenario takes place if structural data, time series or lists, has to be considered and the advantage of recurrent networks to deal with inputs of arbitrary length is used [8, 9]. In fact, recurrent networks can be generalized in a natural way to so called folding networks which take not only sequences but more complex structured objects, labeled trees, as inputs. This is a very promising approach which has applications in classical symbolic areas: term classification and theorem proving [2]. In this scenario recurrent networks and folding networks offer the possibility to use neural and subsymbolic methods in the domain of structured and symbolic data.

We will deal with this third setting and consider the principal capability of recurrent networks to approximate an unknown function. This capability together with the principle learnability of recurrent networks established e.g. in [3] and the existence of learning algorithm [14, 5] will justify the use of recurrent networks in practical applications.

Now we will proceed as follows: After defining recurrent networks formally we will proof their approximation capability in the L_1-norm, i.e. in probability. It will be shown that sequences with entries from a finite alphabet can be approximated with only one neuron in the recursive part. Additionally any mea-

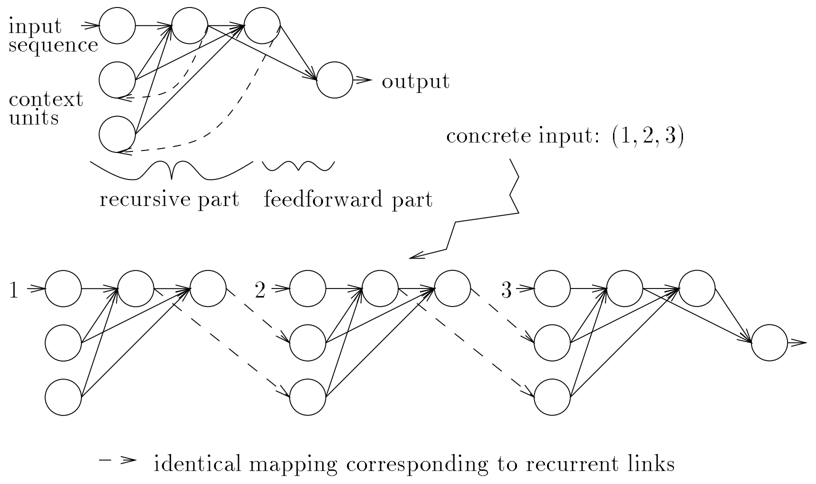
input sequence
context units
output
recursive part
feedforward part
concrete input: (1, 2, 3)
1
2
3
identical mapping corresponding to recurrent links

Figure 1: Recurrent network: Example for a computation

surable function can be approximated as well. Afterwards the approximation capability in the maximum norm is considered. There exist functions that cannot be approximated in the maximum norm. If the range is unlimited, this will be valid even if we restrict to sequences with inputs from a unary alphabet. For limited outputs this is valid for a binary alphabet, whereas any function on a unary alphabet can be approximated. Especially, sigmoidal recurrent networks can compute any even non computable function on unary inputs in linear time which corresponds to a computation on standard binary inputs in exponential time. We conclude with a discussion.

2 Recurrent networks

A **feedforward neural network** consists of neurons $n_1, \ldots, n_N$ that are connected in an acyclic graph. Each connection $n_i \to n_j$ is assigned a weight $w_{ij} \in \mathbb{R}$ and each neuron n_i is assigned a bias $\theta_i \in \mathbb{R}$. The neurons $n_1, \ldots, n_n$ without predecessors are the input neurons. A single neuron computes the function $n_i : \mathbb{R}^n \to \mathbb{R}$ which is defined recursively as

$$n_i(\mathbf{x}) = \begin{cases} x_i, & \text{if } i \le n\,, \\ \sigma_i(\sum_{n_i \to n_j} w_{ji} n_j(\mathbf{x}) + \theta_j), & \text{otherwise} \end{cases}$$

where $\sigma_i : \mathbb{R} \to \mathbb{R}$ is the activation function of neuron i. The entire network computes the function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $f(\mathbf{x}) = (n_{i_1}(\mathbf{x}), \ldots, n_{i_m}(\mathbf{x}))$. n_{i_j} are the so called output units.

In practical applications the cases $\sigma_i = \tanh$ for all neurons resp. the modification $\sigma_i = \mathrm{id}$ for output units if a function with unlimited range shall be approximated are of special interest. In the following we will assume that all activation functions σ_i equal a certain function σ maybe except for the output units i which may have linear activation.

Definition 1 *$(\mathbb{R}^m)^*$ denotes the set of finite sequences with elements in $\mathbb{R}^m$. A function $f : \mathbb{R}^{m+l} \to \mathbb{R}^l$ and vector $\mathbf{y} \in \mathbb{R}^l$* **induces** *a function $\tilde{f}_{\mathbf{y}} : (\mathbb{R}^m)^* \to \mathbb{R}^l$ as follows:*

$$\tilde{f}_{\mathbf{y}}((\mathbf{x}_1, \ldots, \mathbf{x}_i)) = \begin{cases} \mathbf{y} & \textit{if } i = 0\,, \\ f(\tilde{f}_{\mathbf{y}}(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}), \mathbf{x}_i) & \\ & \textit{otherwise}\,. \end{cases}$$

A function $f : (\mathbb{R}^m)^* \to \mathbb{R}^n$ is computed by a **recurrent neural network** *if there exist feedforward networks $g : \mathbb{R}^{m+l} \to \mathbb{R}^l$ and $h : \mathbb{R}^l \to \mathbb{R}^n$ such that $f = h \circ \tilde{g}_{\mathbf{y}}$ for an initial context $\mathbf{y}$.*

g is called the recursive part of the recurrent network, h is called the feedforward part. Of course, the function h could be included in the recursive part. But this notation has the opportunity that the number of neurons needed for the recursive computation can be made explicit. This improves e.g. bounds for the number of examples necessary to train a network correctly with high reliability [3].

One example of a recurrent network and the computation performed on a special input is depicted in Figure 1. In contrast to feedforward networks a recurrent network can deal with input sequences of an arbitrary length.

On $(\mathbb{R}^m)^* = \bigcup_{i=0}^{\infty} (\mathbb{R}^m)^i$ we will consider the algebra resp. topology induced by the Borel algebra resp. standard topology on $(\mathbb{R}^m)^i$. As a consequence a mapping $f : (\mathbb{R}^m)^* \to \mathbb{R}^n$ will be measurable resp. continuous exactly if each restricted mapping $f|(\mathbb{R}^m)^i$ is measurable resp. continuous.

3 Approximation in probability

Let P be a probability measure on the lists $(\mathbb{R}^m)^*$. For measurable functions f_1 and $f_2 : (\mathbb{R}^m)^* \to \mathbb{R}^n$ we say that f_1 approximates f_2 with accuracy ϵ and confidence δ in probability if

$$P(\mathbf{x} \in (\mathbb{R}^m)^* \mid |f_1(\mathbf{x}) - f_2(\mathbf{x})| > \epsilon) < \delta\,.$$

That means the probability of points where f_1 approximates f_2 only poorly is arbitrarily small. Consequently, the two functions differ at most $\epsilon + \delta$ in the L_1-norm $\int |f_1(\mathbf{x}) - f_2(\mathbf{x})| d_P(\mathbf{x})$.

We are interested in the principle capability of recurrent networks to approximate a measurable function $f : (\mathbb{R}^m)^* \to \mathbb{R}^n$ arbitrarily well in probability. Note that $P(\bigcup_{i>i_0} (\mathbb{R}^m)^i)$ gets arbitrarily small for large i_0. Therefore it is sufficient to approximate functions only up to a certain length of the input sequences. This corresponds to the choice of an index i_0 such that $P(\bigcup_{i>i_0} (\mathbb{R}^m)^i) < \delta$. Consequently, to show the approximation capability we could use the ability of recurrent networks to approximate dynamic systems on a limited time interval [13]. It would remain to show that the values of the function f can be produced by a dynamic system with continuous transition function on compact intervals and up to a fixed time e.g. using appropriate polynomials and a counter for the time steps.

But here, we will construct an approximating network of f directly. This has the advantage that for lists with entries from a finite alphabet the number of neurons in the recursive part is limited by 1 whereas in the approximation of a dynamic system this number would grow arbitrarily. Further, it shows that measurable functions can be approximated as well which are in general not produced by a dynamic system with continuous transition.

First, we consider lists with elements from a finite alphabet $\Sigma = \{1, \ldots, b\}$. The following lemma can be seen as a special case of [4].

Lemma 2 *Any function $f : \Sigma^* \to \mathbb{R}^n$ can be approximated arbitrarily well in probability with a recurrent neural network if the following holds: The activation of the output units in the feedforward network is the identity, any other activation equals σ which is a squashing function and continuously differentiable in a neighborhood of one point x_0 with $\sigma'(x_0) \neq 0$. Only one neuron is needed in the recursive part.*

Proof: Let S denote a finite subset of sequences such that the probability of the complement is smaller than δ. A value in $\{1, \ldots, b\}$ needs at most $b_0 = [\log b] + 1$ digits in a decimal representation. The mapping

$$g_{\mathbf{y}}(\mathbf{x}) = \mathbf{x} \cdot (0.1)^{b_0} + \mathbf{y} \cdot (0.1)^{b_0+1}$$

induces a mapping $\tilde{g}_0((\mathbf{x}_1, \ldots, \mathbf{x}_l))$ which simply concatenates the numbers of the sequence expanded by 0 to exactly b_0 digits in a decimal representation: $\tilde{g}_0((\mathbf{x}_1, \ldots, \mathbf{x}_l)) = 0.x_l \ldots x_1$. $\tilde{g}_0$ is injective and g can be approximated by a recursive network with one computation node using the identity

$$\frac{\sigma(x_0 + \epsilon x)}{\epsilon \sigma'(x_0)} \approx x$$

for small ϵ. We can choose ϵ such that the resulting approximation of $\tilde{g}_0$ remains injective on S. Further, this approximation can be computed by a network with only one hidden neuron, because the terms $\epsilon\sigma'(x_0)$ and ϵ can be considered to be part of the links in the recursive steps, maybe the initial context has to be changed.

On the image $\tilde{g}_0(S)$ of the recursive computation we can define a function h such that $h(\tilde{g}_0(s)) = f(s)$ because $\tilde{g}_0$ is injective on S. h can be completed to a continuous function and therefore be approximated arbitrarily well by a feedforward network [6]. The composition of these networks approximates f. $\square$

Here, one can explicitely bound the number of neurons necessary for exact interpolation of a finite training set in $\Sigma^* \times \mathbb{R}^n$. As already mentioned, one neuron in the recurrent part is sufficient. Since the encoded images of the recursive part lie in general position it follows directly from [1] that at most $2\frac{pn}{1+n}$ hidden neurons are necessary in the feedforward part. Here, p is the number of patterns.

Further, the construction can be expanded to approximate a measurable function $f : (\mathbb{R}^m)^* \to \mathbb{R}^n$, too: It is sufficient to approximate a discretization of f which can be seen as a function on lists with entries in a finite alphabet as before.

Theorem 3 *Let the activation be as in Lemma 2. Any measurable function $f : (\mathbb{R}^m)^* \to \mathbb{R}^n$ can be approximated arbitrarily well in probability by a recurrent network.*

Proof: It follows e.g. from [6] that any measurable function $f|(\mathbb{R}^m)^i$ can be approximated arbitrarily well with probability $1 - (0.5)^{i+1}\delta/6$ by a continuous function and therefore f with probability $1 - \delta/6$ by a continuous function. Now we consider this continuous approximation instead of f and denote it with the same symbol. First, we discretize f. Choose

$B > 0$ such that

$$P(x \mid f(x) \notin]-B, B[^n) < \delta/6\,.$$

Decompose $]-B,B[^n$ in disjoint open intervals $I_1, \ldots, I_k$ in $\mathbb{R}^n$ with diameter at most $\epsilon/2$ such that for the boundary b between the intervals I_i the inequality $P(f^{-1}(b)) < \delta/6$ holds. For any j the set $f^{-1}(I_j)$ is open and can be written as a countable union of intervals $J_1^j, \ldots$ where $J_i^j \in (\mathbb{R}^m)^{l_{ij}}$ for some l_{ij}. Choose a finite number of these intervals such that the complement in $(\mathbb{R}^m)^*$ has probability smaller than $\delta/6$. Again, these finite number of intervals is decomposed such that each interval is the product of some standard intervals $]b_i, b_{i+1}[= J_i$ where $b_1 < b_2 < \ldots < b_{l+1} \in \mathbb{R}$ and the symmetric difference of the original intervals and the decomposition into products of standard intervals has probability at most $\delta/6$.

Now we have got a discretization: We can map a sequence $(\mathbf{x}_1, \mathbf{x}_2, \ldots)$ which is characterized by the intervals $J_{i_1}, J_{i_2}, \ldots$ the components of $\mathbf{x}_i$ belong to, to an arbitrary value in the interval I_i where $J_{i_1} \times J_{i_2} \times \ldots$ is mapped to. The value will differ at most $\epsilon/2$ from $f((\mathbf{x}_1, \mathbf{x}_2, \ldots))$ except for a set of probability smaller than $5\delta/6$ in $(\mathbb{R}^m)^*$.

The discretization is implemented with a network as follows: The function $f_{\mathbf{y}}(\mathbf{x})$ in the proof of Lemma 2 is combined with the sum of indicator functions $\sum_{j=1}^m (l^j \cdot \sum_{i=1}^l i \cdot 1_{J_i}(x_j))$. This maps $\mathbf{x}$ to a value $\sum_j (l^j \cdot i_j)$ if the component x_j of $\mathbf{x}$ is in J_{i_j}. The feedforward part just maps the finite set of numbers characterizing the intervals $J_{i_1} \times J_{i_2} \times \ldots$ to an appropriate image of $J_{i_1} \times J_{i_2} \times \ldots$ under f. In the recursive part the indicator 1_{J_i} is approximated by

$$0.5\,\sigma(\frac{x - b_i}{\epsilon_0}) + 0.5\,\sigma(\frac{b_{i+1} - x}{\epsilon_0})\,.$$

Because $\lim_{\epsilon_0 \to 0} \sigma(x/\epsilon_0) = \pm 1$ except for $x = 0$ this approximation can be performed uniformly on the intervals J_i except for the values near the boundary. For large ϵ_0 this has measure at most $\delta/6$. $\square$

The same technique as in Theorem 3 can be used to expand the proof in [4] to a similar result concerning the approximation of mappings on trees with real valued labels. As a consequence of this theorem recurrent networks are in principle well suited for approximation purpose as long as we do not want to approximate a function on any input up to a certain degree. Additionally if we use inputs from a finite alphabet e.g. for modeling terms we need only one neuron in the recursive part.

4 Approximation in the maximum norm

In the feedforward case it is well known that any continuous mapping can be approximated on a compact set arbitrarily well in the maximum norm [6]. We can ask the same question in the recurrent case, too. Actually, this reaches the topic of computability. Recurrent networks where the activation function is locally Lipschitz can be simulated by a family of non-uniform Boolean circuits with resources limited by the time the network uses for computation [11]. Therefore, any function that is not computable by a non uniform circuit with resources growing at most polynomially cannot be computed by a recurrent network. It remains to show that such a function exists.

Here, we do not use this simulation but construct directly a computable function which cannot be approximated by a recurrent network in the maximum norm. The first construction holds even for inputs from a unary alphabet and for any continuous activation function.

Construction: [1] Assume the activation function is continuous. On the sequences $x_1 = (1)$, $x_2 = (1,1), \ldots$ a function value $y_i \in \mathbb{R}$ is recursively constructed such that it lies outside the range of any network with at most i neurons and weights absolutely bounded by i. This is possible because any network architecture defines a continuous mapping on the weights and input sequence. The values (x_i, y_i) can be completed to a continuous mapping on $\mathbb{R}^*$ which is computable on the restriction to $\mathbb{N}^*$. This function cannot be approximated by any recurrent network because it is growing too fast. $\square$

Even if the function that shall be approximated has a limited range, it cannot necessarily be approximated with a network. But here, we need inputs from a binary alphabet. In fact, the following construction can be used to find a function that is not computable with non uniform circuits in polynomial time, too. The Lipschitz condition for neural networks is substituted by the following conditions: The identity can be approximated, e.g because σ is continuously differentiable at x_0 with $\sigma'(x_0) \neq 0$. Any network architecture with inputs restricted to a certain length l can only implement a polynomial number of all possible mappings $\{0,1\}^l \to \{-1,1\}$. That is, the architecture has a VC-dimension growing polynomially in the input length. This is fulfilled e.g. if the activation function is the tanh or a piecewise polynomial function.

[1] This idea is due to T.Elsken.

Construction: As already mentioned, we assume that the activation is a function σ which can approximate the identity. Further we assume, that the number of input sequences of length at most l which can be mapped to values with arbitrary sign by an appropriate choice of the weights is limited by a function p depending on l and the number of the neurons such that p is polynomially in l.

Note that for any finite set of sequences and recurrent networks with at most N neurons we can find a network with N^2 neurons which approximates each single network by an appropriate choice of the weights on the input sequences in a uniform manner. This is due to the fact that any graph of N nodes can be simulated within N^2 nodes such that the input and output units can be taken identical for each simulation. Further, the identity can be approximated.

Now we consider input sequences in $\{0,1\}^*$ and recursively construct outputs such that the number of neurons of a minimal network which approximates the values is growing in each step. As a consequence the entire mapping which can be completed to a continuous mapping on $\mathbb{R}^*$ with bounded range cannot be approximated in the maximum norm.

Assume we have constructed the mapping up to sequences of length l_0 and let N_0 be the minimal number of neurons of a network mapping the sequences correctly. There exist 2^{2^l} mappings $\{0,1\}^l \to \{-1,1\}$. Any network with N_0 neurons can be simulated in a uniform manner by a network with N_0^2 neurons which can implement any mapping on at most $p(l, N_0^2)$ input sequences of length l. Take l such that $l > l_0$ and $2^{2^l} > p(l, N_0^2)$. Then, there exist at least one mapping of the sequences of length l that cannot be implemented by a network with N_0 neurons. □

Note that this is constructive if the loading problem for feedforward networks is solvable. As a consequence there exist computable mappings that cannot be approximated in the maximum norm. But on the other hand there exist mappings that are not computable but can be implemented by a neural network [10]. In fact, we can approximate any mapping on unary inputs:

Theorem 4 *Assume the activation σ is continuously differentiable in a neighborhood of x_0 with $\sigma(x_0) = 0$ and $\sigma'(x_0) \neq 0$. Further, σ is a continuous squashing function with range $[-1,1]$. Any function $f : \{1\}^* \to [-1,1]$ on sequences of a unary alphabet can be approximated arbitrarily well in the maximum norm with a recurrent network.*

Proof: We decompose the range of f in k intervals

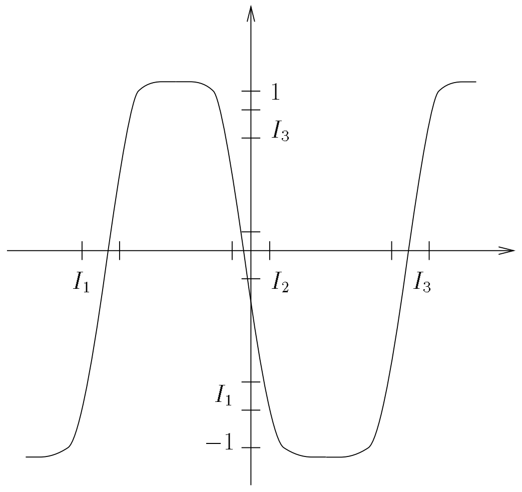

Figure 2: Function g with $g(I_i) \supset \bigcup_j I_j$

J_i, k odd, with diameter at most ϵ. We can find a function $g : \mathbb{R} \to \mathbb{R}$ which is an affine combination of σ and in each J_i a closed interval I_i with $g(I_i) \supset \bigcup_j I_j$. This can be seen as follows: Choose a point x_i inside J_i and constants K_1 and K_2 such that

$$K_2 \frac{\sigma(x_0 + K_1(x - x_i))}{\sigma'(x_0)} = g_i(x)$$

has the properties:

In a small interval I_i containing x_i the function is nearly linear and covers the union of the J_i with a small extra distance δ. For $K_1 \geq 1$ this is possible by an appropriate choice of K_2 since the quotient equals approximately $K_2 K_1 (x - x_i)$ for values in a small interval containing x_i, the diameter of this interval I_i still depends on K_1 which is chosen in the next step.

Outside an interval containing I_i, but itself contained in J_i the function equals $\pm K_2/\sigma'(x_0)$ up to an accuracy δ/k. This is possible by an appropriate choice of K_1 since σ is a squashing function.

Now the sum of the functions $g(x) = \sum_i (-1)^i g_i(x)$ looks like depicted in Figure 2. It has the property $g(I_i) \supset \bigcup_j I_j$: Inside I_i only the part $\pm g_i$ which is nearly linear is relevant, the other g_j are approximately $\pm K_2/\sigma'(x_0)$ which sums to 0.

The function g can be implemented by a recurrent network. We want to approximate $f : \{1\}^* \to \mathbb{R}$. It is sufficient to map $\underbrace{(1,\ldots,1)}_{i}$ to an arbitrary value inside I_{k_i} if $f((1,\ldots,1)) \in J_{k_i}$. As a consequence it is sufficient to choose any value of $\bigcap_{i\in\mathbb{N}} (g^i)^{-1}(I_{k_i})$ as initial context. Note that this intersection is not empty because I_i is compact, g is continuous, and any finite intersection $\bigcap_{i\leq n} (g^i)^{-1}(I_{k_i})$ is not empty because of the property $g(I_i) \supset \bigcup_j I_j$. □

Theorem 4 demonstrates the super Turing capability of recurrent networks. In fact it is well known that recurrent networks with semilinear activation can implement any mapping if the computation may take exponential time. Here, the result is extended to the sigmoidal activation since computation in exponential time corresponds to computation on unary inputs in linear time.

5 Conclusion

The principal capability of recurrent networks to approximate functions arbitrarily well in probability has been established. Consequently, they are in principal well suited for learning tasks where the input consists of lists or sequences, i.e. the input dimension can vary from example to example. In the maximum norm approximation is not possible in general due to the connection of recurrent networks with inputs of arbitrary length to computation models similar to Turing machines.

This result is in some way complementary to the situation concerning the learnability of recurrent neural networks: Consider a fixed recurrent architecture. For inputs up to a finite length learnability is guaranteed. Generally, this holds for a fixed probability: In this case it can be said up to which length input sequences have to be considered. The probability of longer sequences nearly vanishes. On the contrary neither learnability without refereeing to the concrete distribution nor distribution independent bounds for the number of examples required for valid generalization can be established. That means, there exist too many different recurrent networks if the difference on inputs up to arbitrary length is relevant. But for finite inputs a fixed network can be identified by a finite number of examples, up to small distances there exist only a finite number of networks with a fixed architecture [3].

However, this situation demonstrates that the approximation results are sufficient for practical applications. We do not need approximation in the maximum norm because due to information theoretic limitations we cannot train a network such that it corresponds to a special function on any input in general.

The universal approximation capability on sequences from a unary alphabet has an interesting theoretical consequence: In [7] the Turing capability of sigmoidal recurrent networks with batch inputs and exponential time limit is established. Although the super Turing capability of sigmoidal networks is well known [10] the possibility to compute any mapping in exponential time has only been established for functions with a semilinear activation [11]. The technique in Theorem 4 can be used to show the capability to compute any mapping in exponential time with batch input for the sigmoidal activation, too: Assume a sequence $(x_1, x_2, \ldots, x_n)$ is encoded as $3^{-\sum_{i=1}^{n} x_i 2^i}$ in the activation of one unit. After an exponential number of multiplications with 3 we get the number 1, before, the value is always smaller than 0.5. But as in [7] the multiplication with 3 can be approximated by a sigmoidal network such that the relative error compared to the output after an exponential number of steps can be limited. The error is independent of the number of steps n because it is mainly composed by a small constant and an exponentially decaying sum. Consequently, the simultaneous computation of this multiplication and the computation in the proof of Theorem 4 will output after $\sum x_i 2^i$ steps the value ≈ 1 and another value which can be chosen as $\approx \pm 1$ arbitrarily. Any ± 1-valued function can be computed in this way. This is a nice generalization of the surprising super Turing results of recurrent networks [11] to the standard sigmoidal activation.

References

[1] A. Eliseeff and H. Paugam-Moisy. Size of multilayer networks for exact learning. In *Neural Information Processing Systems*, 1996.

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

[3] B. Hammer. Learning recursive data may be intractable. Technical report, Universität Osnabrück, 1997.

[4] B. Hammer and V. Sperschneider. Neural networks can approximate mappings on structured objects. In *2nd International Conference on Computational Intelligence and Neuroscience*, 1997.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997.

[6] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 1989.

[7] J. Kilian and H. T. Siegelmann. The dynamic universality of sigmoidal neural netwroks. *Information and Computation*, 128, 1996.

[8] M. Mozer. Neural net architectures for temporal sequence processing. In A. Weigend and N. Gershenfeld, editors, *Predicting the future and understanding the past*. Addison-Wesley, 1994.

[9] M. Reczko. Protein secondary structure prediction with partially recurrent neural networks. *SAR and QSAR in environmental research*, 1, 1993.

[10] H. T. Siegelmann. The simple dynamics of super Turing theories. *Theoretical Computer Science*, 168, 1996.

[11] H. T. Siegelmann and E. D. Sontag. Analog computation, neural networks, and circuits. *Theoretical Computer Science*, 131, 1994.

[12] H. T. Siegelmann and E. D. Sontag. On the computational power of neural networks. *Journal of Computer and System Sciences*, 50, 1995.

[13] E. D. Sontag. Neural nets as systems models and controllers. In *7th Yale Workshop on Adaptive and Learning Systems*, 1992.

[14] R. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1, 1989.