Learning and Extracting Initial Mealy Automata With a Modular Neural Network Model

Peter Tiňo
Department of Informatics and Computer Systems
Slovak Technical University, Ilkovicova 3, 812 19 Bratislava, Slovakia

Jozef Šajda
Institute of Control Theory and Robotics
Slovak Academy of Sciences, Dubravska cesta 9, 842 37 Bratislava, Slovakia

Neural Computation, 7(4), pp. 822-844, 1995.

A hybrid recurrent neural network is shown to learn small initial mealy machines (that can be thought of as translation machines translating input strings to corresponding output strings, as opposed to recognition automata that classify strings as either grammatical or nongrammatical) from positive training samples. A well-trained neural net[1] is then presented once again with the training set and a Kohonen self-organizing map with the "star" topology of neurons is used to quantize recurrent network state space into distinct regions representing corresponding states of a mealy machine being learned. This enables us to extract the learned mealy machine from the trained recurrent network. One neural network (Kohonen self-organizing map) is used to extract meaningful information from another network (recurrent neural network).

1 Introduction

Considerable interest has been shown in language inference using neural networks. Recurrent networks were shown to be able to learn small regular languages (Das and Das, 1991; Watrous and Kuhn, 1992; Giles *et al.*, 1992; Zeng *et al.*, 1994). The recurrent nature of these networks is able to capture the dynamics of the underlying computation automaton (Das, Giles and Sun, 1992). Hidden units' activations represent past histories and clusters of these activations can represent the states of the generating automaton (Giles *et al.*, 1992). The training of the first-order recurrent neural networks that recognize finite state languages is discussed in (Elman, 1990), where the results were obtained by training the network to predict the next symbol, rather than by training the network to accept or reject strings of different lengths (Watrous and Kuhn, 1992). The problem of inducing languages from examples has also been approached using second-order recurrent networks (Giles *et al.*, 1992; Watrous and Kuhn, 1992).

The orientation of this work is somewhat different. An initial mealy machine (IMM) transforms a nonempty word over an input alphabet to a word of the same length over an output alphabet. So far, there has not been much work on learning IMMs. Chen, *et al.* (1992) trained the so called recurrent neural network sequential machine to be some simple IMM. In (Das and Das, 1991) the task of learning to recognize regular languages was approached by training Elman's recurrent network (Elman, 1990) to simulate an IMM with binary output alphabet. Such an IMM can function as a regular language recognizer (see section 5).

[1] A neural network that has learned to process all the training input samples as specified by the training set

During our experiments a hybrid recurrent neural network (with both first-, and second-order neural units) was presented with a training set consisting of input strings (nonempty words over an input alphabet, more details in (Tiňo and Collingwood, 1994)) and corresponding output strings (transformations of input strings performed by a given IMM). The network learns to perform the same task as the given IMM. In section 4 a method for extraction of learned IMM from a trained recurrent network is presented.

2 Mealy Machines

A brief introduction to mealy machines will be given. For a more detailed explanation see (Shields, 1987).

We define an *Initial Mealy Machine* as a 6–tuple $M = (X, Y, Q, \delta, \lambda, q_0)$, where X and Y are finite sets of symbols called *input* and *output alphabets* respectively, Q is a finite *set of internal states* , δ is a map $\delta : Q \times X \rightarrow Q$, λ is a map $\lambda : Q \times X \rightarrow Y$, and $q_0 \in Q$ is an *initial state.*

For each finite set A the number of elements of A is denoted by $|A|$. An IMM M can be thought of as a directed graph with $|Q|$ nodes. Every node has $|X|$ outgoing arcs labeled with $x|y$ ($x \in X,\ y \in Y$) according to the rule:
The arc from the node labeled with $q \in Q$ to the node labeled with $p \in Q$ is labeled with $x|y$ *iff* $p = \delta(q, x),\ and\ \ y = \lambda(q, x)$. The node corresponding to the initial state is indicated by an arrow labeled with START. Such a graph is referred to as a *state transition diagram* (STD) of the IMM M.

The machine operates as follows. A sequence of inputs $x_1 x_2 ... x_n,\ x_i \in X,\ i = 1, ..., n$, is given to the machine. These cause it to change state from q_0 (the initial state) successively to states $q_1, ..., q_n$, where for each $k \in \{1, ..., n\}$:

$$q_k = \delta(q_{k-1}, x_k).$$

An output occurs during each transition, that is, a sequence of outputs $y_1 y_2 ... y_n,\ y_i \in Y,\ i = 1, ..., n$, is obtained, where for each $k \in \{1, ..., n\}$:

$$y_k = \lambda(q_{k-1}, x_k).$$

Hence, for a given sequence of inputs there is defined a sequence of outputs, to which the sequence of inputs is transformed by the machine.

3 Recurrent Neural Network

3.1 Architecture

Das and Das (1991) used Elman's Simple Recurrent Network (Elman, 1990) possessing first-order neurons to simulate IMMs. Miller and Giles (1993) studied the effect of order in recurrent neural networks in context of grammatical inference. They reported that for small regular languages, the performance of first- and second- order recurrent networks was comparable, whereas for a larger randomly generated 10-state grammar, second-order networks outperformed first-order ones. Goudreau, *et al.* (1992) showed that, in case of hard threshold activation function, the introduction of second-order neurons brought a qualitative change to the representational capabilities of recurrent neural networks in that second-order single layer recurrent neural networks (SOSLRNNs) are strictly more powerful than first-order single layer recurrent neural networks (FOSLRNNs). In particular, they showed that a FOSLRNN can not implement all IMMs, while a SOSLRNN can implement any IMM. If the FOSLRNN is augmented with output layers of feedforward neurons, it can implement any IMM, provided a special technique, called state splitting, is employed (Goudreau *et al.*, 1992). The situation is different when the activation function is a continuous-valued sigmoid-type function (e.i. a saturated linear function), in which case a FOSLRNN is turing uquivalent (Siegelmann and Sontag, 1991; Siegelmann and Sontag, 1993).

Our network architecture designed for learning IMMs is essentially a second-order recurrent network connected to an output layer which is a multilayer perceptron. In particular, the recurrent neural network (RNN) used in our experiments has

- N input neurons labeled I_n
- K nonrecurrent hidden neurons labeled H_k
- M output neurons labeled O_m
- L recurrent neurons, called state neurons, labeled S_l
- $L^2 \cdot N$ real-valued weights labeled W_{iln}
- $K \cdot L \cdot N$ real-valued weights labeled Q_{jln}
- $M \cdot K$ real-valued weights labeled V_{mk}.

There are nontrainable unit weights on the recurrent feedback connections.

We refer to the values of the state neurons collectively as a state vector S. The approach taken here involves treating the network as a simple dynamical system in which previous state vector is made available as an additional input.

The recurrent network accepts a time–ordered sequence of inputs and evolves with dynamics defined by the equation:

$$S_i^{(t+1)} = g(\sum_{l,n} W_{iln} \cdot S_l^{(t)} \cdot I_n^{(t)}), \tag{1}$$

where g is a sigmoid function $g(x) = 1/(1 + e^{-x})$.

The quadratic form $\sum_{l,n} W_{iln} \cdot S_l^{(t)} \cdot I_n^{(t)}$ directly represents the state transition process : [state,input] $\rightarrow$ [next state] (Giles *et al.*, 1992).

The output of the network at time t is defined by the equations:

$$O_m^{(t)} = g(\sum_k V_{mk} \cdot H_k^{(t)}) \tag{2}$$

$$H_j^{(t)} = g(\sum_{l,n} Q_{jln} \cdot S_l^{(t)} \cdot I_n^{(t)}) \tag{3}$$

Again, the quadratic form $\sum_{l,n} Q_{jln} \cdot S_l^{(t)} \cdot I_n^{(t)}$ represents the output function diagrams: [state,input] $\rightarrow$ [output].

Each input string is encoded into the input neurons one symbol per discrete time step t. The evaluation of (2) and (3) yields the output of the network at the time step t, while by means of (1) the next state vector is determined.

The basic scheme of the used RNN is presented in figure 1.

An unary encoding of symbols of both the input and output alphabets is used with one input and one output neuron for each input and output symbol respectively. In case of large alphabets, this might be restrictive (Giles and Omlin, 1992).

3.2 Training procedure

The error function E is defined as follows:

$$E = \frac{1}{2} \sum_m (D_m^{(t)} - O_m^{(t)})^2 \tag{4}$$

where $D_m^{(t)} \in \{0, 1\}$ is the desired response value for the m–th output neuron at the time step t.

The training is an on–line algorithm, that updates the weights after each input symbol presentation, with a gradient–descent weight update rules:

$$\Delta V_{mk} = -\alpha \frac{\partial E}{\partial V_{mk}} \tag{5}$$

$$\Delta Q_{jln} = -\alpha \frac{\partial E}{\partial Q_{jln}} \tag{6}$$

$$\Delta W_{iln} = -\alpha \frac{\partial E}{\partial W_{iln}} \tag{7}$$

α is a positive real number called *learning rate.* During the experiments its value ranged from 0.1 to 0.5.

After the choice of initial weight values, the gradient of E can be estimated in real time as each input $I^{(t)} = (I_1^{(t)}, ..., I_N^{(t)})$ enters the network.

To overcome the problem of local minima (at least to some extend), the "momentum" terms were incorporated into (5), (6), and (7). The effect of the momentum is that the local gradient deflects the trajectory through parameter space, but does not dominate it (Miller and Giles, 1993).

3.3 Presentation of Training Samples

Let $A = (X, Y, Q, \delta, \lambda, q_0)$ be an IMM the RNN is expected to learn. To train the network "to mimic the behaviour" of A, the network is presented with the training data consisting of a series of stimulus–response pairs. The stimulus is a nonempty string $w = x_1...x_n$ over the input alphabet X (a finite sequence of symbols $x_i \in X$, containing at least one symbol), and the response is the corresponding output string, i.e. the sequence $\lambda(q_0, x_1)\ \lambda(q_1, x_2)\ ...\ \lambda(q_{n-1}, x_n)$, where $q_i = \delta(q_{i-1}, x_i),\ \ i = 1, ..., n$.

Transformation of each stimulus into the corresponding response starts with the IMM A being in the initial state q_0. Since the training set consists of a series of stimulus–response pairs, it is desirable to ensure that before presentation of a new stimulus the network is forced to change its state[2] to a state corresponding to the initial state of the IMM A. This leads to introduction of a new input "reset" symbol #. In the IMM used for training the network, all states make a transition to the initial state q_0 when the input symbol # is presented. Also, a special output symbol $ is added to the IMM output alphabet. Due to the unary coding, it is necessary to add another neuron to both input and output neurons.

In fact the network is trained to mimic the behaviour of the initial mealy machine $A' = (X \cup \{\#\}, Y \cup \{\$\}, Q, \delta', \lambda', q_0)$, where $X \cap \{\#\} = Y \cap \{\$\} = \emptyset$, and δ', λ' are defined as follows:

$$\forall q{\in}Q, \forall x{\in}X;\ \delta'(q, x) = \delta(q, x)\ \ and\ \ \lambda'(q, x) = \lambda(q, x),$$

$$\forall q{\in}Q;\ \delta'(q, \#) = q_0\ \ and\ \ \lambda'(q, \#) = \$.$$

Stimulus strings in the training set representing the IMM A' are of the form $s\#$, where s is a stimulus string from the training set representing the IMM A.

To construct the training set representing the IMM A one has to specify the number of training samples to be included in the training set, as well as the maximum length $\mathcal{L}_{max} \geq 1$ of the stimulus string. Furthermore, for each stimulus length $\mathcal{L} \in \{1, ..., \mathcal{L}_{max}\}$ the portion (expressed in percentage) of training samples of length $\mathcal{L}$ in the whole training set has to be specified. A typical example could be ($\mathcal{L}_{max} = 10$):

stimulus length	**percentage**
1	2
2	2
3	5

[2]The state vector S of the network is also referred to as a state of the network

4	7
5	9
6	10
7	15
8	15
9	15
10	20

Each training sample is constructed by randomly generating a stimulus with prescribed length $\mathcal{L}$ and determining the response of the IMM A to that stimulus. In the training set, training samples are ordered according to their stimulus lengths. As an example, here are the first five training samples from the training set representing the IMM shown in figure 2a. At the end of each stimulus there is the reset symbol #, which causes transition to the initial state, so that the processing of the next stimulus can start.

stimulus	**response**
a#	1$
b#	0$
b#	0$
ba#	00$
aa#	11$

Let $\epsilon_m, \quad m=1,...,M$, be the maximum of the absolute errors $|O_m - D_m|$ on the m-th output neuron that appeared during the presentation of the training set.
Denote $\max_m\{\epsilon_m\}$ by ϵ. The net is said to correctly classify training data if $\epsilon < \mu$, where μ is a pre–set "small" positive real number. During our simulations μ varied from 0.04 to 0.1. If the net correctly classifies the training data, the net is said to *converge*.

4 Extraction of IMM

Consider an output symbol $y \in Y$ coded as $[y_1, ..., y_M]$. Due to an unary encoding of symbols, there exists exactly one $i \in \{1, ..., M\}$, such that $y_i = 1$, and for all $j \in \{1, ..., M\} \setminus \{i\}$, $y_j = 0$[3]. The output $O = (O_1, ..., O_M)$ of the network is said to correspond to the symbol y with the uncertainty $\Delta \in (0, \frac{1}{2})$, if $O_i \in (1-\Delta, 1)$, and for each $j \in \{1, ..., M\} \setminus \{i\}$, $O_j \in (0, \Delta)$ holds. During our experiments Δ was set to 0.1.

Since the patterns on the state units are saved as context, the state units must develop representations which are useful encodings of the temporal properties of the sequential input (Elman, 1990). The hypothesis is, that during the training, the network begins to partition its state space into fairly well–separated distinct regions, which represent corresponding states of the IMM being learned (Giles *et al.*, 1992; Watrous and Kuhn, 1992; Zeng *et al.*, 1993; Cleeremans *et al.*, 1989; Tiňo and Collingwood, 1994). Once the network has converged, then the network generalized the training set correctly, only if the "behaviour" of the net and the "behaviour" of the IMM used for training are the same (for a more rigorous treatment of the concept of behaviour see (Tiňo and Collingwood, 1994)).

Each state of the IMM is represented by one, or several clusters in the network's state space. The crucial task is to find those clusters. The internal states (represented by activations of state units) the network is in as it processes various input sequences are inspected. In practical terms this involves passing the training set through the converged network (weights are frozen), and saving the state units' activation patterns that are produced in response to each input symbol. This is followed by the cluster analysis of thus acquired activation patterns. A modification of the *Self-Organizing Map* (SOM) introduced by Kohonen (1990) was used for that purpose. Since the original Kohonen's SOM is generally well-known, we will focus only on its main principles.

[3] $\setminus$ denotes the operation of set subtraction.

4.1 Self-Organizing Map

SOM belongs to the class of *vector coding algorithms*. In vector coding the problem is to place a fixed number of vectors (called codevectors) into the input space. Each codevector represents a part of the input space – the set of points of the input space that are closer in distance to that codevector than to any other codevector. This produces a Voronoi tesselation (Aurenhammer, 1991) into the input space. The codevectors are placed so as the average distances from the codevectors to the input points belonging to their own Voronoi compartment[4] are minimized.

In neural network implementation each codevector is the weight vector of a neural unit. Neurons in a SOM constitute the so called *neuron field* (NF). Neurons in the NF are usually organized in a regular neighborhood structure. Line or a rectangular 2-dimensional grid of neurons are the most obvious examples of such a structure. A well-trained map responds to the input signals in an orderly fashion, i.e. the topological ordering of neurons reflects the ordering of samples from the input space (Cherkassky and Lari-Najafi, 1991). Furthermore, if input signals are given by the probability density function, then weight (reference) vectors try to imitate it (Kohonen, 1990).

Each neuron has an associated index i characterizing its position in the neighborhood structure of NF. Input samples $x = (x_1, ..., x_n)$ are presented to the j-th neuron through the set of n connection links weighted by $w_j = (w_{j1}, ..., w_{jn})$. During the training phase, the map is formed by successive adjustments of the vectors w_j. Given a randomly chosen vector x in the input space, the unit i which is most similar to x is detected. Among many measures of similarity, the Euclidean distance is most favored by researchers and is also used in this study:

$$||x - w_i|| = \min_j \{||x - w_j||\}.$$

After that, the weight vector w_i and all the weight vectors w_j of neurons in the vicinity of i are shifted a small step toward x:

$$\Delta w_j = \epsilon \; h_{j,i}(x - w_j).$$

The function $h_{j,i}$ determines the size of the vicinity of i which takes part in the learning. It depends on the distance $d(j,i)$ between neurons j and i in the NF. The function decreases with increasing $d(j,i)$. A typical choice for $h_{j,i}$ is

$$h_{j,i} = exp\left(\frac{-d^2(j,i)}{2\sigma^2}\right).$$

The complete learning phase consists in a random initialization of the w_j followed by a number of the above-described learning steps. For the learning to converge, it is helpful to slowly decrease the step-size $\epsilon(t)$ as well as the width $\sigma(t)$ of $h_{j,i}$, during the learning process. An exponential decay

$$\epsilon(t) = \epsilon_0 \; exp\left(\frac{-t}{\tau_\epsilon}\right)$$

$$\sigma(t) = \sigma_0 \; exp\left(\frac{-t}{\tau_\sigma}\right)$$

often works well.

A NF is considered to be a set of vertices in the graph, and the topology of neurons (vertices) (i.e. their mutual arrangement) to be edges in an undirected graph. The lengths of edges are equal to 1. This is an unnecessary restriction and arbitrary positive lengths of edges can be used. The distance $d(j,i)$ between units i and j in the NF is defined to be the shortest path from the vertex i to the vertex j in the graph representing the NF.

If no neighborhood structure is introduced to the NF, i.e. $h_{j,i} = \delta_{ji}$, where

$$\delta_{ji} = \begin{cases} 1 & \text{if } \; j = i \\ 0 & \text{otherwise,} \end{cases}$$

[4] A part of the input space represented by a codevector is referred to as the Voronoi compartment of that codevector.

the well-known statistical technique of *vector quantization* is obtained (Lampinen and Oja, 1992). Only the weight vector that is most similar to the current input sample is allowed to be modified. All the other weight vectors are left untouched. This can have serious consequences, since if some of the weight vectors initially lie somewhere far from the input vectors, they do not lend themselves to modification and so do not take part in the learning process. This makes the map inefficient and strongly sensitive to the initial values of the weight vectors (Kia and Coghill, 1992). The concept of neighborhood preservation in the SOM can help us to overcome this difficulty. The goal of SOM learning is to find the most representative codevectors for the input space (in mean square sense), as well as, to realize a *topological mapping* from the input space to the NF. By a topological mapping the following property is meant: If an arbitrary point from the input space is mapped to unit i, then all points in a neighborhood of that point are mapped either to the unit i itself, or to one of the units in the neighborhood of i in the NF[5].

On the other hand, the standard regular-grid topologies are often too "rigid" to capture the distribution of input samples characterized by several well-separated distinct clusters. Indeed, it has been already pointed out (Harp and Samad, 1991) that for clustering purposes, 1-D map organizations are often better than 2-D ones, even when the input space is 2-D.

For the task of cluster detection in the space of RNN state units' activations the SOM with the co called *star topology* of NF was used. The star topology of NF is characterized by one "central" neuron connected to several "branches" of neurons (figure 3). If the number of neurons in each branch is the same, the star topology with b branches, each having n neurons, is denoted by $St(b/n)$.

The star topology of neurons constitutes a compromise. The idea is that during the training, the branches of neurons move quickly toward clusters of samples, not devoting too much attention to regions of the input space without any sample. During the training, individual neurons do not adapt independently of each other, while the loose neighborhood structure makes the training process less problematic and faster (Tiňo, Jelly and Vojtek, 1994).

The final map can be visualized in the input space. Weights w_j are shown as circles (to distinguish them from the input samples shown as crosses), and weights of neighboring neurons are connected with dashed-lines.

In figure 4 it can be seen how a SOM with $St(6/2)$ topology of NF "capture" the distribution of RNN state neurons' activations, after the RNN with two state neurons was successfully trained the IMM shown in figure 2b.

4.2 Extraction Procedure

Once the set of saved state units' activation patterns is partitioned into clusters, each defined by the corresponding codevector in the NF of a SOM, the state transition diagram of the IMM being extracted can be generated as follows:

1. Let the set of the IMM states be the set of codevectors obtained during the learning of SOM.

2. The initial state corresponds to the codevector representing the cluster to which the state units' activation pattern belongs after the symbol # has been presented at the recurrent network input.

3. Let the activation pattern of state units be s_1. If, after the presentation of an input symbol x, the recurrent network's state units constitute an activation pattern s_2, and network's output corresponds to an output symbol y, then, provided s_1 and s_2 belong to the clusters defined by the codevectors C_{s_1} and C_{s_2} respectively, the arc from the node labeled by C_{s_1} to the node labeled by C_{s_2} is labeled by $x|y$.

[5]Whether the topological property can hold for all units depends on the dimensions of the input space and the neuron lattice, since no genuine topological map between two spaces of different dimensions can exist. For example, a two-dimensional neural layer can only locally follow two dimensions of the multi-dimensional input space. The *topographic product* - a measure of the preservation of neighborhood relations in maps between spaces of possibly different dimensionality - can be used to quantify the neighborhood preservation of a SOM (Bauer and Pawelzik, 1992).

This process terminates after presentation of the whole training set to the recurrent network input. However, the transition diagram generated by this process need not be the transition diagram of an IMM. There may exist a state from which arcs to various different states have the same label. The following procedure is a remedy for such a situation.

1. Let the set of all states of the extracted transition diagram be denoted by $Q^{(1)}$. Set the value of k to 1.

2. **case A:** There exist states $q, p_1, ..., p_m \in Q^{(k)}$, $m > 1$, such that there exist arcs from q to $p_1, ..., p_m$ with the same label.

   - The set $Q^{(k+1)}$ is constructed from the set $Q^{(k)}$ as follows:
     $$Q^{(k+1)} = \{g \mid g \in Q^{(k)} \text{ and } g \notin \{p_1, ..., p_m\}\} \cup \{p\},$$
     where $p \notin Q^{(k)}$. If one of the states p_i, $i = 1, ..., m$, is the initial state, then p will be the new initial state. Otherwise, the initial state remains unchanged. Each arc from $g \in Q^{(k)}$ to $h \in Q^{(k)}$ is transformed to the arc from $g' \in Q^{(k+1)}$ to $h' \in Q^{(k+1)}$ (the label remains unchanged), where
     $$g' = \begin{cases} g & \text{if } g \notin \{p_1, ..., p_m\} \\ p & \text{otherwise,} \end{cases}$$
     and
     $$h' = \begin{cases} h & \text{if } h \notin \{p_1, ..., p_m\} \\ p & \text{otherwise.} \end{cases}$$
   - $k := k + 1$
   - go to step 2

   case B: No such states $q, p_1, ..., p_m \in Q^{(k)}$ exist.

   - If there exist $g, h \in Q^{(k)}$, $x \in X$, $y_1, y_2 \in Y$, $y_1 \neq y_2$, such that there exist two arcs from g to h labeled with $x|y_1$ and $x|y_2$ respectively, then the extraction procedure has failed. Otherwise, the extraction procedure has successfully ended.

The extracted IMM can be reduced using a well-known minimization algorithm (Shields, 1987). The minimization process rids the IMM of redundant, unnecessary states.

The idea of quantization of recurrent neural network state space in order to establish a correspondence between regions of equivalent network states and states of a finite state machine (the network was trained with) is not a new one. In (Cleeremans *et al.*, 1989) a hierarchical cluster analysis was used to reveal that patterns of state units' activations are grouped according to the nodes of the grammar used to generate the training set. Giles *et al.* (1992) divide the network state space into several equal hypercubes. It was observed, that when the number of hypercubes was sufficiently large, each hypercube contained only network states corresponding to one of the states of the minimal acceptor defing the language the network was trained to recognize. Hence, even with such a simple clustering technique a correct acceptors could be extracted from well-trained networks.

Somewhat different approach was taken in (Zeng *et al.* 1993), where the quantization of network state space was enforced during the training process. In particular, state units' activation pattern was mapped at each time step to the nearest corner of a hypercube as if state neurons had had a hard threshold activation function. However, for determination of a gradient of cost function (to be minimized during training) a differentiable activation function was used (to make the cost function differentiable).

As pointed out in (Das and Mozer, 1994), in the approaches of Cleeremans *et al.* and Giles *et al.* learning does not consider the latter quantization and hence there is no guarantee that the quantization step will group together state units' activation patterns corresponding to the same state of the finite state machine used for training the net. On the other hand, quantization process presented in (Zeng *et al.* 1993) causes error surface to have discontinuities and to be flat in local neighborhoods of the weight space (Das and Mozer, 1994). Consequently, direct gradient-descent-based learning cannot be used and some form of heuristic has to be applied (e.i. the above mentioned technique of true gradient replacement by the pseudo-gradient associated with a differentiable activation function).

Das and Mozer (1994) also view the network state space quantization as an integral part of the learning process. In their most successful experiments a "soft" version of the gaussian mixture model[6] in a supervised mode was used as a clustering tool. The mixture model parameters were adjusted so as to minimize the overall performance error of the whole system (recurrent network + clustering tool). Furthermore, the cost function encoureges unnecessary gaussians to drop out of mixture model. The system itself decides the optimal number of gaussians it needs to solve the task properly.

While this is a promissing approach we note, that even the kind of "soft" clustering introduced in (Das and Mozer, 1994) (to ensure that the discontinuities in error surface caused by hard clustering are eleminated) does not eliminate all discontinuities in error function. As noted in (Doya, 93) and discussed in (Tiňo and Collingwood, 1994), when recurrent neural network is viewed as a set of dynamical systems associated with its input symbols, the training process can be described from the point of view of bifurcation analysis. In order to achieve a desirable temporal behaviour, the net has to undergo several bifurcations. At bifurcation points the output of the net changes discontinuously with the change of parameters. This seems to be a burden associated with recurrent neural networks trained via gradient descent optimization of a cost function continouosly depending on networks' output. For a discussion of special problems arising in trainig recurrent neural networks see (Doya, 1992) and (Pineda, 1988).

Our approach falls into the category of first-train-then-quantizise approaches. However, our experiments suggest that clusters of equivalent network states that evolve during RNN training are well-separated and usually naturally "grasped" by SOM with star topology of NF. Furthermore, clusters are quickly reached by branches of neurons in SOM and normally 1–3 training epochs are sufficient for SOM to achieve desirable quantization of RNN state space. The number of neurons in SOM has to be pre-determined, but since the training procedure for SOM is very simple and fast[7] this does not represent a great problem.

5 Experiments

At the beginning of each training session the recurrent network is initialized with a set of random weights from the closed interval $[-0.5, 0.5]$. The initial activations $S_l^{(0)}$ of state units are also set randomly from the interval $[0.1, 0.5]$.

The RNN was trained with four relatively simple IMMs M_1, M_2, M_3, and M_4, represented by their STDs in figures 2a, 2b, 2c, and 2d respectively. The first three IMMs were chosen so as to train the RNN an IMM characterized by one of the following cases:

- one input symbol is associated only with loops in the STD (symbol a in M_1), while the other input symbol (symbol b in M_1) is associated only with a cycle in the STD
- there is an input symbol that is associated with both a loop and a cycle in the STD (symbol b in M_2)
- there are several "transition" states leading to a "trap" state, from which there is no way out (M_3)

We have studied how such features of STDs are internally represented in a well-trained RNN. In particular, there seems to be a one-to-one correspondence between loops in the state transition diagram of an IMM used for training, and attractive fixed points of dynamical systems corresponding to the well-trained RNN. A similar relationship between cycles and attractive periodic orbits was

[6] Instead of the center with greatest posterior probability given a pattern of state units' activation, a linear combination of centers is used, where each center is weighted by its posterior probability given current network state.

[7] The goal here is only to detect the clusters in the network state space as opposed to the usual goal of SOM - finding the best representation (characterized by minimal information loss) of the SOM input space with limited number of quantization centers.

found too. The more loops and cycles exist in a STD (i.e. the more "complex" is structure of the task to be learned), the more state neurons the RNN needs to reflect that structure. Consider the IMMs M_3 and M_4. Both of them have five states. Apparently, the structure of M_4 is more complex than that of M_3. To properly learn the IMM M_3, the RNN needs only two state neurons, while to learn the IMM M_4, four state neurons are needed. For more details see (Tiňo and Collingwood, 1994).

The IMMs M_1 and M_3 have binary output alphabets $\{0, 1\}$, and can thus be considered regular language recognizers: a string belongs to the language only if the output symbol after presentation of the string's last symbol is 1.

To detect clusters in the set of saved state units' activation patterns of a well-trained RNN, a Modified Kohonen Map with the "star" topology of neurons was used. IMMs were extracted from converged RNNs according to the process described in section 4. Clusters of network state units' activation patterns correspond to the states of extracted IMM. Each of those clusters is referred to by its codevector, or equivalently, by the position of the codevector's unit in the NF of a SOM.

Each extracted IMM was minimized to its reduced form. In all successful experiments the STDs of minimized extracted IMMs were found identical (up to labels on nodes)[8] to those of the IMMs originally used for training the RNN. The experiments failed in two cases.

1. the number of RNN state neurons was too small. In this case the RNN failed to converge, since the small number of state neurons did not allow the network to develop appropriate representations of internal states of the IMM used for training.

2. the number of neurons used for SOM was too small. This was the case when the given topology of neurons was too "rough" to capture the underlying distribution of activations of RNN state neurons. In such cases more "branches" of neurons are needed and/or several additional neurons have to be added to the individual "branches".

To illustrate how the extraction procedure works, consider the following example: The RNN with four state neurons has successfully learned the IMM M_4. To detect the clusters of state neurons' activity, a SOM with $St(5/2)$ topology of NF is used. During the training process, the SOM devoted the first, tenth, and eleventh neurons to the cluster representing the state C of the IMM M_4. The cluster representing the state E was detected by the sixth and seventh neurons. The way how the SOM "captured" the distribution of clusters in $(0, 1)^4$ can be seen in figure 5.

The whole training set was then again presented to the RNN. By partitioning the space of activations of state units with trained SOM we arrived at the diagram presented in figure 6.

Now, $Q^{(1)} = \{1, 2, 3, 5, 6, 7, 9, 10, 11\}$. There exist two arcs labeled by $a|0$, that lead from the state 1 to both the states 2 and 3. This means that the states 2 and 3 are merged together to form a new state 2'. The set of states is thus reduced to $Q^{(2)} = \{1, 2', 5, 6, 7, 9, 10, 11\}$. The new STD is constructed according to the procedure described in the last section. The existence of arcs labeled by $a|1$ leading from the state 2' to the states 1, 10, and 11 indicates that those three states should be collapsed into one state 1'. Furthermore, there exist two arcs labeled by $b|1$ that lead from the state 2' to both the state 6 and 7. Denoting the new state into which the states 6 and 7 are collapsed by 6', the new set of states $Q^{(3)} = \{1', 2', 5, 6', 9\}$ gives rise to the STD identical up to labels on nodes (states) to the STD of M_4.

Some details about five of the experiments are summarized in tables 1 and 2. Nt is used to denote the number of training examples in the training set, Lm stands for the greatest stimulus' length, lr, mr, and ϵ denote the learning rate, momentum rate, and training error respectively. We present only L and K, the numbers of RNN state units and nonrecurrent hidden units, since the numbers of input and output units are uniquely determined by unary encoding of input and output symbols respectively. The last column of table 1 presents the topology of NF in SOM used to extract learned state machine from well-trained RNN. – in table 2 means that the network did not converge after 200 epochs.

[8]In a successful experiment, the IMM extracted from a converged RNN was isomorphic with the reduced form of the IMM used for training. More details in (Tiňo and Collingwood, 1994).

#	IMM	trn. set Nt	Lm	RNN L	K	learning parameters lr	mr	ϵ	topol. of NF in SOM
1	M1	100	9	2	3	0.5	0.1	0.05	St(3/2)
2	M2	200	13	2	4	0.4	0.07	0.06	St(4/3)
3	M3	250	12	2	3	0.4	0.07	0.06	St(6/2)
4	M4	600	12	4	3	0.3	0.07	0.065	St(5/2)
5	M4	600	12	4	4	0.5	0.1	0.065	St(5/2)

Table 1: Details obout the experiments.

#	# epochs to converge in run 1	2	3	4	5	6	7	8	9	10
1	14	18	12	17	15	15	12	17	13	16
2	–	47	48	61	62	24	43	29	58	36
3	78	82	91	91	89	22	79	79	79	123
4	–	67	–	–	–	21	–	–	47	–
5	68	24	–	74	–	69	25	–	75	–

Table 2: Details obout the experiments - continuation.

6 Discussion

Our aim is to investigate the ability of the proposed type of RNN to learn small IMMs. Converged network exhibits the same behavior as the IMM it was trained to mimic. As pointed out in (Tiňo and Collingwood, 1994), the reduced, connected IMM corresponding to the IMM extracted from the network should be isomorphic with the reduced and connected IMM originally used for training the network. This is in accordance with our experiments. The quantization of state units' activation space can be achieved using a modified SOM with "star" topology of NF. Individual "branches" of neurons have a potential to quickly reach clusters. The "star" topology works well, because the clusters of state units' activations appear to be "well behaved", in that they form small, densely sited regions that are well separated from each other (Tiňo, Jelly and Vojtek, 1994).

Theoretically unnecessary layer $O^{(t)}$ of first order neurons was used to enhance the power of the RNN to code the relationship between network inputs and activities of state neurons representing network memory. Actually, all our experiments were also performed using a RNN structure without the layer $O^{(t)}$. In this case $H^{(t)}$ functions as an output layer. Such a structure was introduced in (Chen *et al.* , 1992) as a Neural Mealy Machine (NMM).

Assume that to learn the same task, using the same coding of input and output symbols, the NMM and our model of RNN[9] need $N \cdot L_1 \cdot M + N \cdot L_1^2$ and $N \cdot L_2 \cdot K + K \cdot M + N \cdot L_2^2$ modifiable weights respectively[10]. Assume further that the numbers of state neurons in NMM and RNN are the same, i.e. $L_1 = L_2 = L$. L reflects the "complexity" of the task to be learned[11]. The number of modifiable weights in NMM is higher than that of RNN only if

$$K < \frac{1}{\frac{1}{M} + \frac{1}{N \cdot L}} = K_*.$$

[9] In what follows we will refer to our model of RNN simply as RNN.

[10] In the NMM, the number of neurons in the layer $H^{(t)}$ has to be M, since $H^{(t)}$ functions as an output layer. L_1 is the number of state neurons of NMM. The number of RNN state neurons is L_2.

[11] The complexity of an IMM is intuitively determined by the number of loops, and cycles in the STD of the reduced form of the IMM, and their mutual relationship. To learn a more "complex" IMM, more state neurons are needed. For more details see (Tiňo and Collingwood, 1994). It may also happen, that due to the additional layer $O^{(t)}$, the number of RNN state neurons is lower than that of NMM, when learning the same task. This case is not discussed here.

This implies that if the task is characterized by a high "complexity", and a great number of input and output symbols, then there is a chance that the number of neurons in the layer $H^{(t)}$ needed for the RNN to converge will be less than K_*. It should be noted that the smaller number of modifiable weights does not necessarily imply faster convergence of the training process. Nevertheless, using the unary coding of input an output symbols, to learn the IMM M_4, the RNN and the NMM needed 60 and 64 modifiable weights respectively. The RNN had 4 state neurons and 3 hidden nonrecurrent neurons. The number of NMM state neurons was 4. RNN converged in 3 out of 10 runs, after 67, 21, and 47 epochs respectively. NMM converged in 5 out of 10 runs, after 65, 99, 72, 110, and 149 epochs respectively. Hence, the introduction of an unnecessary layer $O^{(t)}$ can have a desirable effect on the training process, but further comparative experiments with more complex task are needed.

References

Aurenhammer, F. 1991. Voronoi Diagrams - Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys* **3**, 345–405.

Bauer, H., and Pawelzik, K. R. 1992. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on Neural Networks* **4**, 406-414.

Chen, D., Giles, C. L., Sun, G. Z., Chen, H. H., and Lee, Y. C. 1992. Learning Finite State Transducers with a Recurrent Neural Network. *IJCNN Int. Conference on Neural Networks, Beijing, China.* **vol I**, 129–134.

Cherkassky, and V., Lari-Najafi, H. 1991. Constrained Topological Mapping for Nonparametric Regression Analysis. *Neural Networks* **4**, 27–40.

Cleeremans, A., Servan-Schreiber, D., and McClelland, J. 1989. Finite state automata and simple recurrent networks. *Neural Computation* **1(3)**, 372–381.

Das, S., and Das, R. 1991. Induction of Discrete-State Machine by Stabilizing a Simple Recurrent Network Using Clustering. *Computer Science and Informatics* **2**, 35–40.

Das, S., Giles, C. L., and Sun, G. Z. 1992. Learning Context-free Grammars: Capabilities and Limitations of a Recurrent Neural Network with an External Stack Memory. In *Proceedings of The Fourteenth Annual Conference of The Cognitive Science Society*, Indiana University.

Das, S., and Mozer, M. C. 1994. A Unified Gradient-Descent/Clustering Architecture for Finite State Machine Induction. *Advances in Neural Information Processing Systems* **6**, 19–26.

Doya, K. 1992. Bifurcations in the Learning of Recurrent Neural Networks. *Proc. of 1992 IEEE Int. Symposium on Circuits and Systems.*, 2777-2780.

Elman, J. L. 1990. Finding Structure in Time. *Cognitive Science* **14**, 179-211.

Harp, S. A., Samad, T. 1991. Genetic Optimization of Self-Organizing Feature Maps. *Proceedings of IJCNN-91, Seattle* .

Kia, S. J., Coghill, G. G. 1992. Unsupervised clustering and centroid estimation using dynamic competitive learning. *Bio. Cybern.* **67**, 433–443.

Kohonen, T. 1990. The Self-Organizing Map. *Proc. of the IEEE* **9**, 1464–1480.

Lampinen, J., Oja, E. 1992. Clustering Properties of Hierarchical Self-Organizing Maps. *J. Mathematical Imaging and Vision* , Preprint.

Giles, C. L., Miller, C. B., Chen, D.,Chen, H. H., Sun, G. Z.,and Lee, Y. C. 1992. Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks. *Neural Comp.* **4**, 393–405.

Giles, C. L., and Omlin, C. W. 1992. Inserting rules into recurrent neural network. In *Proceedings of The 1992 IEEE Workshop on Neural Networks for Signal Processing*, Copenhagen, Denmark.

Goudreau, M. W., Giles, C. L., Chakradhar, S. T., and Chen, D. 1992. Firs-Order vs. Second-Order Single Layer Recurrent Neural Networks. *Preprint, accepted for publication in IEEE Transactions on Neural Networks.*

Miller, C. B., Giles, C. L. 1993. Experimental Comparison of the Effect of Order in Recurrent Neural Networks. *International Journal of Pattern Recognition and Artificial Intelligence* **7(4)**, 849–872.

Pineda, F. J. 1988. Dynamics and architecture for neural computation. *Journal of Compexity* **4**, 216-245.

Shields, M. W. 1987. *An Introduction to Automata Theory*, Blackwell Scientific Publications, London.

Siegelmann, H. and Sontag, E. 1991. Neural networks are universal computing devices. *Technical Report SYCON-91-08*, Rutgers Center for Systems and Control.

Siegelmann, H. and Sontag, E. 1993. Analog computation via neural networks. *Accepted for Theoretical Computer Science*, preprint.

Tiňo, P., and Collingwood, P. C. 1994. Initial Mealy Machines and Recurrent Neural Networks. *unpublished manuscript.*

Tiňo, P., Jelly, I. E., and Vojtek, V. 1994. Non-Standard Topologies of Neuron Field in Self-Organizing Feature Maps. *Proceedings of the AIICSR'94 conference, Slovakia*, World Scientific Publishing Company, 391–396.

Watrous, R. L., and Kuhn, G. M. 1992. Induction of Finite-State Languages Using Second-Order Recurrent Networks. *Neural Comp.* **4**, 406-414.

Zeng, Z., Goodman, R., and Smyth, P. 1993. Learning finite state machines with self-clustering recurrent networks. *Neural Computation* **5(6)**, 976-990.

Zeng, Z., Goodman, R., and Smyth, P. 1994. Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks* **5(2)**, 320-330.

7 Legends to Figures

Figure 1: RNN model used for learning IMMs.
Figure 2: STDs of the IMMs M_1, M_2, M_3 and M_4 are presented in figures 2a, 2b, 2c, and 2d respectively. M_1 is shown with extra state transitions initiated by a special "reset" input symbol # (dashed-line arcs).
Figure 3: $St(4/4)$ topology of NF. Neighboring neurons are connected with lines.
Figure 4: The RNN with two state neurons was trained the IMM M_2 (figure 2b). After the net converged, it was once again presented with the training set, and the activations of Network state units (shown as crosses) were saved. The figure shows how distribution of state units' activations was "captured" by SOM with $St(6/2)$ topology of NF.
Figure 5: Diagram indicating how neurons in $St(5/2)$ topology of NF detected clusters in state units' activation space of the RNN with four state neurons. Numbers are indices of neurons in NF, letters indicate states of the IMM M_4 used for training the RNN that correspond to the clusters.
Figure 6: Extracted transition diagram from the converged RNN trained with the IMM M_4.