Automated Abstraction of Dynamic Neural Systems for Natural Language Processing

Henrik Jacobsson, Stefan L. Frank & Diego Federici

***Abstract*— This paper presents a variant of the Crystallizing Substochastic Sequential Machine Extractor (`CrySSMEx`), an algorithm capable of extracting finite state descriptions of dynamic systems, such as recurrent neural networks, without any regard to their topology or weights. The algorithm is applied to a network trained on a language prediction task. The extracted state machines provide a detailed view of the operations of the RNN by abstracting and discretizing its functional behaviour. Here we extend previous work and extract state machines in Moore, rather than in Mealy, format. This subtle difference opens up the rule extractor to more domains, including sensorimotor modelling of autonomous robotic systems. Experiments are also conducted on far more input symbols, providing a greater insight into the behaviour of the algorithm.**

I. Background

`CrySSMEx` (Crystallizing Substochastic Sequential Machine Extractor) [1] is an algorithm which extracts finite state descriptions of recurrent neural networks operating in symbolic domains. The algorithm employs an iterative approach where the weakest point of the currently strongest model is continuously targeted for improvement through active data selection. The algorithm also constantly keeps the extracted model minimal through merging of equivalent states. The result is a state space quantization which takes into account functional rather than the static geometric properties of the state vectors (i.e. state vectors are interpreted as elements of a state space of a dynamic system and not simply of a Euclidean space).

It was shown that `CrySSMEx` can efficiently extract (stochastic or deterministic) finite state machines from recurrent neural networks (RNNs) trained on deeply embedded grammars ($\mathbf{a}^n\mathbf{b}^n$), highdimensional RNNs (10^3 state nodes) and chaotic systems. `CrySSMEx` was, however, defined as to extract Mealy state machines from Mealy type neural networks. That is, the output of the network and of the extracted machine is defined over *transitions* between states. In a Moore machine, however, the output is defined as a function of the state [2]. If, for each state, there is a unique output mapping, the system is conceptually more correctly described as a Moore system and the conceptual gap between the system and its extracted model is smaller if the latter is also of Moore type. Also, the extracted model will be more readable and compact. The following description will be kept brief as more details can be found in [1].

Henrik Jacobsson is with the German Research Center for Artificial Intelligence (DFKI) In Saarbrücken, Germany, `henrik.jacobsson@dfki.de`. Stefan Frank is with the Nijmegen Institute for Cognition and Information, Raboud University Nijmegen, The Netherlands, `S.Frank@nici.ru.nl`. Diego Federici worked at the university of Skövde in the context of the ICEA project and is with Google Switzerland, `diegofederici@google.com`

II. Models and algorithms

To not only include neural networks, a more generalized form of Mealy system was suggested in [1], a situated discrete time dynamic system (SDTDS). In the same manner, we now define a *Moore Situated Discrete Time Dynamic System* ($\text{SDTDS}_{\text{Moore}}$), as a quadruple $\langle S, I, O, \gamma \rangle$ where $S \subseteq \mathbb{R}^{n_s}$ is a *set of state vectors*, $I \subseteq \mathbb{R}^{n_i}$ is a *set of input vectors*, $O \subseteq \mathbb{R}^{n_o}$ is a *set of output vectors*, $\gamma_s : S \times I \rightarrow S$ is the *transition function*, and $\gamma_o : S \rightarrow O$ is the *state interpretation function*, and $n_s, n_i, n_o \in \mathbb{N}$ are the *dimensionalities* of the state, input, and output spaces respectively.

The interpretation of the $\text{SDTDS}_{\text{Moore}}$ is that, if the system, at time t occupies a state $\vec{s}(t)$, it will have output $\vec{o}(t) = \gamma_o(\vec{s}(t))$. If it is fed an input $\vec{i}(t)$, then the resulting next state is determined by $\vec{s}(t+1) = \gamma_s(\vec{s}(t), \vec{i}(t))$.

The goal of the extraction of rules from the system is to create a model (rule set) that mimics the underlying system but is more comprehensible [3], [4]. This induced model is a substochastic finite state machine [5] in which states are occupied with a probability between zero and one (the sum of probabilities is ≤ 1, which is the rationale of the *sub* prefix). The machine is called a Moore substochastic sequential machine ($\text{SSM}_{\text{Moore}}$) and is defined as $\langle Q, X, Y, \mathcal{P}_q = \{p_q(q_j|q_i, x_k)\}, \mathcal{P}_y = \{p_y(y_l|q_i)\}\rangle$ where Q is a finite set of state elements (SEs, named so to distinguish them from the states of the $\text{SDTDS}_{\text{Moore}}$), X is a finite set of input symbols, Y is a finite set of output symbols, and $\mathcal{P}_q$ and $\mathcal{P}_y$ are finite sets of *subconditional* probabilities where $q_i, q_j \in Q$, $x_k \in X$ and $y_l \in Y$. The term subconditional here denotes the fact that if $p(q_i, x_k) = 0$ then $\{p(q_j|q_i, x_k)\}$ is *defined* to be 0, despite the convention of letting it be undefined (due to a $\frac{0}{0}$) (more discussion of this in [1] and [6]).

The $\text{SDTDS}_{\text{Moore}}$ is translated into an $\text{SSM}_{\text{Moore}}$ by selecting one or more initial states and feeding the system input sequences from a finite set of input exemplars. This set of all observations of the SDTDS will henceforth be named Ω. The observed states, inputs, and outputs are enumerated by quantization functions Λ_s, Λ_i, and Λ_o, respectively, that collapse the uncountable vector spaces into a finite set of natural numbers (or symbols), i.e. each Λ_* is a function $\Lambda_* : \mathbb{R}^n \rightarrow \mathbb{N}$. These numbers in turn correspond to Q, X, and Y respectively, of the $\text{SSM}_{\text{Moore}}$.

The subconditional probabilities are based directly on the

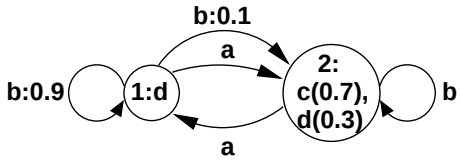

Fig. 1. An example of an $\text{SSM}_{\text{Moore}}$ with $Q = \{q_1, q_2\}$ (denoted only 1 and 2 in figure), $X = \{a, b\}$, $Y = \{c, d\}$, $\mathcal{P}_y = \{p_y(d|q_1) = 1, p_y(c|q_2) = 0.7, p_y(d|q_2) = 0.3\}$ and $\mathcal{P}_q = \{p_q(q_2|q_1, a) = 1, p_q(q_1|q_1, b) = 0.9, p_q(q_2|q_1, b) = 0.1, p_q(q_1|q_2, a) = 1, p_q(q_2|q_2, b) = 1\}$. Zero probability events are omitted and probabilities of exactly one are implied if no probability is written.

observed frequencies of occurrences in Ω, e.g.[1] if $Y = \mathbf{c}$ is always observed when $Q = q_0$ then $p_y(Y = \mathbf{c}|Q = q_0) = 1.0$ and if Q is *either* q_1 or q_2 equally often after transition from q_0 over $X = \mathbf{a}$, then $p_q(q_1|q_0, \mathbf{a}) = p_q(q_2|q_0, \mathbf{a}) = 0.5$. Importantly, if, for example, a transition from $Q = q_1$ over $X = \mathbf{b}$ is never observed, then $p_q(Q = q_i|Q = q_1, X = \mathbf{b}) = 0.0$ for all $q_i \in Q$. The transition from q_1 over input symbol $\mathbf{b}$ will in such cases be called *dead*.

A. SSM_{Moore} parsing

The idea behind the SSM is to model the probabilities of the (enumerated) state and output of the underlying system. If, for example, the state enumeration is known at time $t = 0$, this known enumeration has probability 1.0, whereas all others are 0.0. As the SSM is fed input symbols, the probability distributions over the state elements and output symbols are updated accordingly.

For notational purposes, we denote the distribution over Q as a vector where $\vec{q}_i$ corresponds to $p(Q = q_i)$ and $\vec{q}$ to the full distribution $p(Q)$. Likewise for X and Y. The SE distribution at $t + 1$, resulting from a transition over input symbol x_k, is calculated from the distribution at t as

$$\vec{q}(t+1) = \mathcal{P}_q(\vec{q}(t), x_k(t)) \tag{1}$$

where each element $\vec{q}_j(t+1)$ of $\vec{q}(t+1)$ corresponds to the probability of a SE and is calculated by

$$\vec{q}_j(t+1) = \sum_{i=1}^{|Q|} \vec{q}_i(t) \cdot p_q(q_j|q_i, x_k) \tag{2}$$

The distribution of output symbols $\vec{y}(t)$ over Y is generated from the SE distribution at any time t by

$$\vec{y}(t) = \mathcal{P}_y(\vec{q}(t)) \tag{3}$$

where each element $\vec{y}_l(t)$ of $\vec{y}(t)$ corresponds to the probability of an output symbol and is calculated by

$$\vec{y}_l(t) = \sum_{i=1}^{|Q|} \vec{q}_i(t) \cdot p_y(y_l|q_i) \tag{4}$$

Due to the possibility of dead transitions, the sum of probabilities may be between 0 and 1. To counter this, it is possible to divide the probabilities by their sum. This will be denoted $\hat{\mathcal{P}}_q$ (and is referred to in [1] as *normalized* parsing).

[1]Let us assume $Q = \{q_0, q_1, q_2 \dots\}$, $X = \{\mathbf{a}, \mathbf{b}\}$ and $Y = \{\mathbf{c}, \mathbf{d}\}$.

`NDI_equ`$(ssm, \vec{u}, \vec{v}, H)$
Input: an $\text{SSM}_{\text{Moore}}$ ssm, SE distributions $\vec{u}$ and $\vec{v}$, and history of state support sets H.
Output: returns true if $\vec{u}$ and $\vec{v}$ are not decisively inequivalent given any possible future input.

```
begin
    if P_y(u) ≠ P_y(v) then return false;
    /*i.e. the output must be the same for both SE
      distributions.                                   */
    else if u = v then return true;
    /*i.e. if the distributions are identical, they are equivalent.
      */
    else if ⟨sup(u), sup(v)⟩ ∈ H then return true;
    /*i.e. a loop has been encountered. Eventual
      inequivalence will be encountered in another branch of
      the recursion tree.                              */
    else
        /*If the NDI_equ cannot be asserted, subsequent
          inputs must be tested.                       */
        R := true;
        k := 1;
        while R = true ∧ x_k ∈ X do
            /*If no inequivalence has been shown ...   */
            u' = P̂_q(u, x_k);
            v' = P̂_q(v, x_k);
            if sup(u') ≠ ∅ ∧ sup(v') ≠ ∅ then
                /*... continue testing recursively.    */
                H' = H ∪ ⟨sup(u), sup(v)⟩;
                R := NDI_equ(ssm, u', v', H');
            end
            k := k + 1;
        end
        return R;
    end
end
```

Alg. 1. The recursive function `NDI_equ`$(ssm, \vec{u}, \vec{v}, \emptyset)$ returns true if and only if there is no evidence that the future output distributions could differ depending on which of SE distributions $\vec{u}$ or $\vec{v}$ are occupied in the $\text{SSM}_{\text{Moore}}$.

If the sum is zero then $\hat{\mathcal{P}}_q$ is defined to be 0. An example SSM is shown in Figure 1.

B. Equivalence and determinism of SEs

In automata theory, it is important to distinguish between states that are equivalent from those that are not, since equivalent states can be merged to create a smaller but equivalent machine. Two states of a finite state machine are defined equivalent if it is impossible to tell them apart by observing the output of the machine. Likewise, the SEs of an SSM (Mealy or Moore) have similar equivalence relations, complicated by the fact that there are potentially dead transitions in an SSM. This means that it is unknown what outcome this transition would have produced in the underlying system.

To avoid infinite recursions, we detect loops in sequences of the support sets of SEs. The support set, $\sup(\vec{v})$, of a distribution $\vec{v}$ (i.e. distribution written as a vector) is the set $\{i : \vec{v}_i > 0\}$, i.e. a set which mirrors all elements with a non zero probability in the distribution. The algorithm `NDI_equ` (for *not decisively inequivalent*) shown in Algorithm 1 returns true if none of the (undead) transitions lead to different output distributions.

`CrySSMEx`(Ω, Λ_o)
Input: An SDTDS transition event set, Ω, and an output quantization function, Λ_o.
Output: A deterministic $\text{SSM}_{\text{Moore}}$ mimicking the SDTDS within the domain Ω as described by Λ_o.
begin
  let Λ_i be an invertible quantizer for all I in Ω;
  $i := 0$;
  $ssm^0 :=$ `create_machine`$(\Omega, \Lambda_i, \Lambda_{cvq^0}, \Lambda_o)$;
  /*ssm^0 *has* $Q = \{q_1\}$ *with all transitions to itself.* */
  repeat
    $i := i + 1$;
    $D :=$ `collect_split_data`$(\Omega, ssm^{i-1}, \Lambda_i, \Lambda_{cvq^{i-1}}, \Lambda_o)$;
    $cvq^i :=$ `split_cvq`(cvq^{i-1}, D);
    $ssm^i :=$ `create_machine`$(\Omega, \Lambda_i, \Lambda_{cvq^i}, \Lambda_o)$;
    if ssm^i *has UNDI-equivalent states* **then**
      /*Merge SEs if possible. */
      $E :=$ `generate_UNDI_equivalence_sets`(ssm^i);
      $cvq^i :=$ `merge_cvq`(cvq^i, E);
      $ssm^i :=$ `create_machine`$(\Omega, \Lambda_i, \Lambda_{cvq^i}, \Lambda_o)$;
    end
  until ssm^i *is deterministic*;
  return ssm^i;
end

Alg. 2. The main loop of `CrySSMEx`. `create_machine` samples and quantizes the input, state and output domains of the SDTDS, counts the occurrences of all transitions, and translates these observations into an SSM. `collect_split_data` is described in Algorithm 3. `generate_UNDI_equivalence_sets` generates a set of equivalence sets based on `NDI_equ` in Algorithm 1. `split_cvq` and `merge_cvq` are operations on the state space quantizer which results in either a more fine grained or coarse grained quantization of the state space [1].

`collect_split_data`$(\Omega, ssm, \Lambda_i, \Lambda_s, \Lambda_o)$
Input: A transition event set, Ω, an SSM, ssm, an input quantizer, Λ_i, a state quantizer, Λ_s, an output quantizer, Λ_o.
Output: A list of data sets D, one data set per $q \in Q$. The element of each data set is a pair $\langle \vec{v}, \ell \rangle$ where $\vec{v} \in \mathbb{R}^n$ is a data vector and $\ell \in \mathbb{N}$ is the assigned label of the vector.
begin
  $D := [\emptyset, \emptyset \ldots \emptyset]$;
  for $\forall \langle \vec{s}(t), \vec{\imath}(t), \vec{o}(t+1), \vec{s}(t+1) \rangle \in \Omega$ **do**
    $q_i := \Lambda_s(\vec{s}(t))$;
    $x_k := \Lambda_i(\vec{\imath}(t))$;
    $y_l := \Lambda_o(\vec{o}(t+1))$;
    $q_j := \Lambda_s(\vec{s}(t+1))$;
    /*If q_i *is indeterministic, the state vector should be stored in* D *with an appropriate labelling* */
    if $H_{ssm}(Y|Q = q_j) > 0$ **then**
      /*If Y *indeterministic with respect to* q_i*, label the state vector with the output symbol,* y_l. */
      $D_i := D_i \cup \langle \vec{s}(t), y_l \rangle$;
    else if $\exists x_m : H_{ssm}(Q|Q = q_i, X = x_m) > 0$ **then**
      /*If output is determined from q_i, but next state is not, label state vector using next SE, q_j. */
      if $x_k = \underset{x_m \in X}{\texttt{argmax}}\ H_{ssm}(Q|Q = q_i, X = x_m)$ **then**
        $D_i := D_i \cup \langle \vec{s}(t), q_j \rangle$;
      end
    endif
  end
  return D;
end

Alg. 3. `collect_split_data` collects one data set for each $q_i \in Q$ such that an adaptable quantization function can be trained to properly separate the labelled points of each data set. (Empty data sets are simply ignored.) Entropies $H_{ssm}(Y|Q)$ and $H_{ssm}(Q|Q, X)$ reflect the SSM's level of ambiguity regarding SE interpretation or transition, respectively (more details in [1]).

III. The CrySSMEx-algorithm

The goal of the `CrySSMEx` is to optimize Λ_s so that the $\text{SSM}_{\text{Moore}}$ is minimal and as correct as possible with respect to the observed $\text{SDTDS}_{\text{Moore}}$ behaviour in Ω. The SSM should in principle be able to replace the SDTDS, at least to replicate the enumerated data of Ω. To do this, SEs that seem equivalent in the SSM are merged in Λ_s and SEs that give rise to ambiguities are split. This procedure is iterated until a deterministic machine (i.e. one in which all probabilities are exactly zero or one) is generated. Algorithm 2 describes the main loop. The details of the main algorithm are of less importance as it remains identical to the main loop described in [1], where, instead of a Moore SSM, a Mealy SSM was extracted. The main change to `CrySSMEx` is found in the split (Algorithm 3) and merge (Algorithm 1) operations.

To do the splits and merges, a novel quantizer was suggested in [1], a *Crystalline Vector Quantizer* (CVQ). The CVQ allows splitting enumerated partitions by resampling the state vector that gives rise to the ambiguity (cf. Algorithm 3). The CVQ is capable of merging any arbitrary set of SE enumerations. The NDI-equivalence relation, however, is not transitive, i.e. NDI-equivalence of q_i and q_j, and of q_j and q_k, does not imply the same relation between q_i and q_k. This creates the problem of choosing equivalence sets based on lacking information, as there may not be enough information in Ω to choose the equivalence set: $\{\{q_i\}, \{q_j\}, \{q_k\}\}$, $\{\{q_i, q_j\}, \{q_k\}\}$ or $\{\{q_i\}, \{q_j, q_k\}\}$. Therefore, only SEs that are NDI-equivalent to the same subset of SEs were merged in [1][2]. In the presented experiments, however, $\{q_i, q_j, q_k\}$ will be merged together despite q_i being inequivalent with q_k (which is significantly different from how it was done in [1]). This method will not work all the time, since a SE with no outgoing transitions is NDI-equivalent with all other SEs. Consequentially, the occurrence of such an SE would merge all states into one, returning the algorithm to ssm^0. It turned out to be a feasible approach in the presented experiments, however.

IV. Experiments

A. The language

In the current experiments, we use the language introduced in [7]. Sentences are composed of 30 different words: 9 singular and 9 plural nouns (N_{sing} and N_{plu}, respectively), 5 singular and 5 plural transitive verbs (V_{sing} and V_{plu}), the relative pronoun 'that', and an end-of-sentence marker denoted [end] (and also considered a word). Words are

[2]This is a restrictive approach, reported a problematic future work issue by [1]. Another potential hazard is the use of conditional entropy to determine which is the most informative input symbol (in Algorithm 3).

TABLE I

PRODUCTION RULES FOR GENERATING SENTENCES. VARIABLE n DENOTES NUMBER: SINGULAR (SING) OR PLURAL (PLU). VARIABLE r DENOTES NOUN ROLE: SUBJECT (SUBJ) OR OBJECT (OBJ).

Head		Production
S	→	S_{sing} | S_{plu}
S_n	→	$NP_{n,subj}$ VP_n [end]
$NP_{n,r}$	→	$N_{n,r}$ | $N_{n,r}$ SRC_n | $N_{n,r}$ ORC
VP_n	→	V_n $NP_{sing,obj}$ | V_n $NP_{plu,obj}$
SRC_n	→	that VP_n
ORC	→	ORC_{sing} | ORC_{plu}
ORC_n	→	that $N_{n,subj}$ V_n
$N_{sing,subj}$	→	woman | girl | dog | cat | mouse | duck | cow
$N_{plu,subj}$	→	women | girls | dogs | cats | mice | ducks | cows
$N_{sing,obj}$	→	man | boy | dog | cat | mouse | duck | cow
$N_{plu,obj}$	→	men | boys | dogs | cats | mice | ducks | cows
V_{sing}	→	likes | sees | swings | loves | avoids
V_{plu}	→	like | see | swing | love | avoid

combined into sentences according to the grammar of Table I. Note that nouns refer to either females, males, or animals. Females occur only as subjects while males are always in object position. Animals can occur in either position.

There is no upper limit to sentence length because of relative clauses that can be recursively embedded in the sentence. Relative clauses (i.e., phrases beginning with 'that') come in two types: subject relatives (SRC) and object relatives (ORC). In a SRC, the modified noun is in subject position (as in 'girl that sees boy'), while in an ORC it is in object position (as in 'girl that boy sees'). Noun phrases (NP) have a 0.4 probability of containing a relative clause, resulting in an average sentence length of 7 words.

B. The network

The RNN was based on Jaeger's [8] Echo State Network (ESN) in that the weights of connections to and within the recurrent layer (or Dynamical Reservoir; DR) remain untrained. In contrast to standard ESNs, the RNN used here had an additional hidden layer between the DR and the output, since [9] showed that this greatly improves the network's ability to generalize to new inputs. This means that the network consisted of four layers of units: The 30-unit (one for each word) input layer was fully connected to the DR with 1000 linear units. Of all possible connections between DR units, 85% had a zero weight. The weights of the other 15% were chosen randomly from a uniform distribution and rescaled such that the spectral radius of the weight matrix was 0.7. The DR was connected to the hidden layer, which had 10 sigmoidal units. This layer in turn was connected to the output layer containing one unit for each word.

The RNN received a long sequence of sentences, generated at random by the grammar in Table I, and onnections to the hidden and output layers were trained to predict the following word at each point in the sequence (see [7] for details). This task was mastered nearly perfectly on training sentences.[3]

[3]It is, of course, impossible to always predict the exact next word. Therefore, performance is considered perfect if only grammatically correct predictions are made.

C. Generalization and systematicity

Fodor & Pylyshyn [10] make a strong case against connectionist models of human cognition by arguing that neural networks cannot display the level of systematicity present in human language. A definition of systematicity in terms of generalization is given by Hadley [11], who defines *strong syntactic systematicity* as a network's ability to correctly process novel sentences containing words in new syntactic roles, both in the main clause and in embedded clauses. This means that a network is strongly systematic if it is trained on sentences generated by the grammar of Table I and generalizes to sentences in which the syntactic roles of males and females are reversed. Indeed, this is how [7] showed that the RNN can display strong syntactic systematicity. After training, it was tested on sentences that had male subjects and female objects.[4] For instance, the first word of a test sentence (i.e., the word following the previous sentence's [end]) could be 'boy'. No training sentence starts with the word 'boy', so the RNN needs to be systematic to make correct next-word predictions at this point. As it turned out, it did generalize to different types of test sentences: Predictions were grammatically correct after receiving male nouns in subject position or female nouns in object position, in both main clauses and embedded clauses. This demonstration of strong syntactic systematicity counters the claim in [10] that cognition cannot be explained by connectionist models.

D. Analysis of extracted SSMs

The set of observations Ω, forming the input to `CrySSMEx`, consisted of 200 different (non-training) sentences of the form 'N_{male} that V N_{female} V N_{female} [end]' (e.g., 'boys that see girl like woman [end]'), as well as the DR states resulting from processing these sentences. The network's output vectors were discretized by identifying the most active output unit. The word represented by this unit belongs to one of five word classes: noun (N), singular verb (V_{sing}), plural verb (V_{plu}), 'that', or [end]. These 'predicted word classes' are the output symbols of the SSMs in Figures 2 and 3, that show how the trained RNN processed the sentences.

The correct output symbol(s) at each point in the sentence are listed in Table II. It is important to keep in mind that the trained RNN did not make any errors on this type of sentence, that is, at each point in the sentence, the most active output unit represents a word that would make a grammatical continuation of the sentence.

The sequence of $ssm^0 \rightarrow ssm^N$ represents a series of increasingly specific machines (see Algorithm 2).

1) ssm^0*:* The first SSM, ssm^0, has only one SE with all transitions leading back to it. It provides only the probability distribution of output symbols and no syntactic information. By splitting up the state space so that the output of the RNN is unique for each SE, a more interesting SSM, called ssm^1, is generated.

[4]Animal nouns were not used in test sentences.

TABLE II

CORRECT PREDICTIONS AT EACH POINT OF SENTENCES IN Ω. VARIABLES n, m ($n \neq m$) DENOTE NUMBER (SINGULAR OR PLURAL).

word position	(class of) input word	possible (classes of) next word
1	$N_{male,n}$	that, V_n
2	that	V_n, N
3	V_n	N
4	$N_{female,m}$	that, V_n
5	V_n	N
6	$N_{female,n}$	that, [end]
7	[end]	N

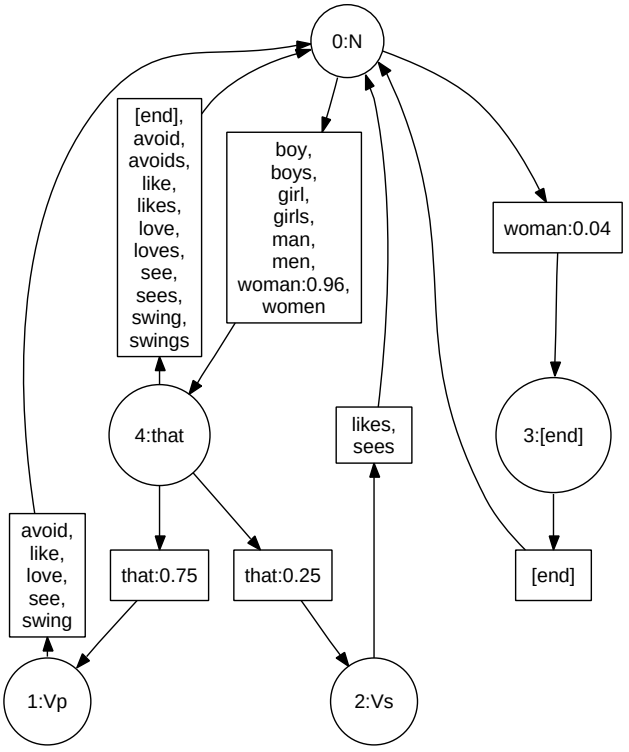

Fig. 2. ssm^1 as constructed by `CrySSMEx`. Circles denote SEs; Words in rectangles are input symbols on the transitions between SEs. See Figure 1 for a detailed account of how to interpret the SSM.

2) ssm^1: At the beginning of the sentence, ssm^1 (Figure 2) is in SE 0, where it produces N (i.e., the RNN predicts the next word to be a noun). Indeed, the first word of the sentence is a (male) noun, so the SSM moves to SE 4. Here, it correctly outputs 'that'. Since the next input is 'that', the machine enters SE 2 (where it produces V_{sing}) or SE 1 (and produces V_{plu}). The RNN makes no errors so the number of the output verb at this point matches the noun received before, but this is not apparent in the current SSM. In any case, the next input word is indeed of the predicted class, and the machine is back in SE 0 to produce a noun. This cycle allows the SSM to produce grammatical infinite embeddings of SRCs ('boy that sees girls that like boys that ...') even though it is based on sentences with only one relative clause.

The next input is a female noun. If it is 'woman', there is a small chance that the SSM ends up in SE 3 and produces [end]. This would be an ungrammatical prediction, so the RNN does not do this. Instead, the SSM should move to SE 4 and produce 'that'. This prediction does not match the actual next input, but it is grammatically correct (see Table II). After receiving a verb, the SSM is again back in SE 0 to produce a noun. The next input is a female noun, and the SSM is most likely to go to SE 4 and again produce 'that'. However, if the input is 'woman', there is a very small change that the output symbol is [end]. This is the actual next input, moving the SSM back to SE 0 where it is ready to receive the first word of the next sentence.

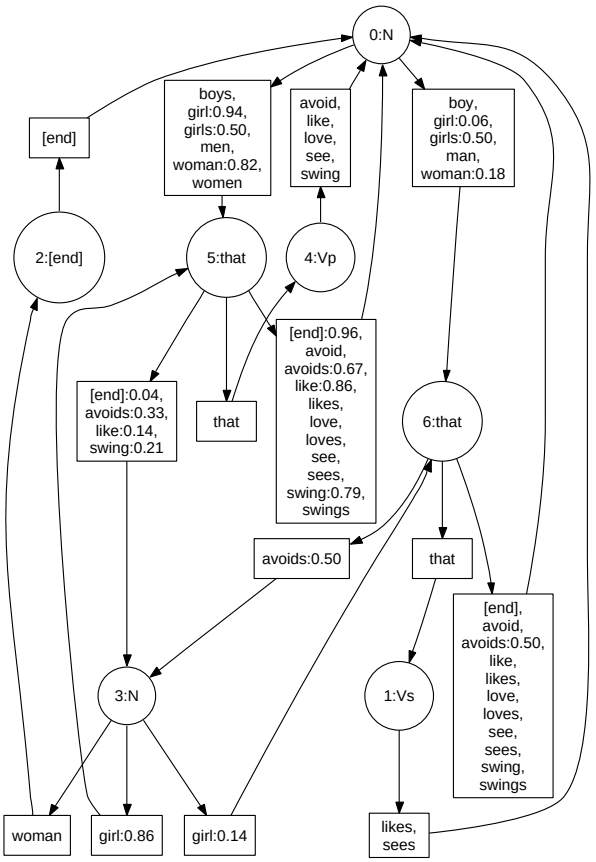

Fig. 3. ssm^2 as constructed by `CrySSMEx`. Circles denote SEs; Words in rectangles are input symbols.

The network's strategy for dealing with the new sentences turns out to be quite simple: Always predict 'that' after a noun. However, it does happen at least once that [end] is correctly predicted when the sentence actual does come to an end, following 'woman'. This is particularly interesting since none of the RNN's training sentences ends with a female noun. This means the network should be strongly biased *against* predicting [end] after 'woman'.

3) ssm^2: The third SSM, ssm^2 (Figure 3) gives a more detailed analysis of the RNN's behavior. At the start of a sentence, the SSM is in SE 0. It receives a male noun, moving it to SE 5 or 6 depending on the noun's number. Note that the previous SSM did not make this distinction. In any case, it produces (and receives) 'that'. The next output is a verb of the same number as the initial noun, and the machine is back in SE 0. As was the case for ssm^1, this allows for infinite

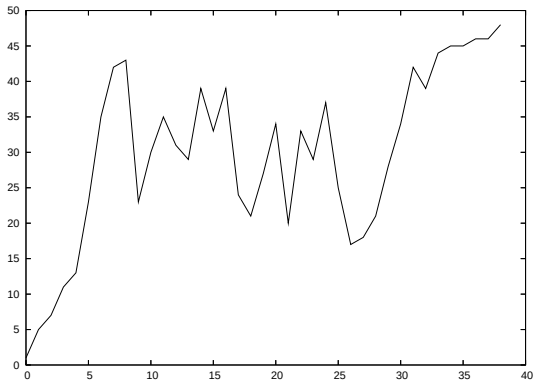

Fig. 4. Number of SEs of the generated SSMs up to iteration 38 when CrySSMEx terminates.

embeddings of subject-relative clauses.

The next input is a female noun, resulting in SE 5 or 6. Again, 'that' is produced. After receiving a verb, the machine moves to a state (3 or 0) where the output is a noun, which indeed is the class of the next input word. As always after receiving a noun, 'that' is predicted, although there is a very small chance that the output symbol at this point is [end], indicating that the sentence has come to an end.

In general, the RNN's behavior according to ssm^2 is much like that shown by ssm^1. Nouns are virtually always followed by the grammatical prediction of 'that', which is how the network deals with sentences in which nouns occur in new syntactic roles. Occasionally, the word 'woman' at the end of the sentence results in the prediction of [end], even though this goes against the RNN's training examples.

In contrast to ssm^1, ssm^2 has two SEs that can be reached after receiving the input [end]. Most often, the machine is in SE 0 after the end of a sentence, but it ends up in SE 3 at least once. The only way out of this state is by receiving a female noun. However, the SSM is based on sentences that always begin with a male noun. Therefore, the '[end]-entrance' into SE 3 must result from the very last sentence in Ω.

4) ssm^3 to ssm^{38}: As can be seen in Figure 4, CrySSMEx generates very large SSMs. In fact, they become very difficult to read. They do, however, become more and more accurate in terms of the RNNs actual behaviour[5]. The problem is that when the RNN has multiple grammatically correct paths to choose, the particular underlying rules that determine the choice, require many states to be fully and accurately described in the SSM. CrySSMEx does not take into account that a transition might only be occurring once in Ω. And many of the SEs are supported with little data.

V. Conclusions

We have shown how CrySSMEx manages to extract also Moore machines from RNNs. The illustrative experiment was conducted on a highdimensional RNN trained on performing a prediction task on a subset of natural language. The resulting state machines clearly showed how the RNN dealt with new sentences, that is, how it managed to be systematic. Shortcomings of CrySSMEx were apparent from the very large (but correct) machines generated from the extraction.

[5]By selecting SSMs at different points of CrySSMEx-iterations, one can choose SSMs of either high fidelity or comprehensibility. Further enhancement of comprehensibility of extracted SSMs is possible too, e.g. by omitting infrequent states.

VI. Further Work

An interesting application is related to the analysis of the dynamical reservoir of neural models such as ESN and Liquid State Machines. Given a computational task, CrySSMEx can be used to reconstruct the discrete regions of the reservoir connected to the processing of the output. The partitioning of the computational space of the reservoir can be used to understand how computation is internally organized. For example, to estimate the robustness of the networks. Also, in [12] the CVQ will be used to describe how perceptions from sensor modalities are fused to give rise to sensorimotor states.

Acknowledgments

This research was partly supported by the EU FP6 IST Cognitive Systems Integrated Project Cognitive Systems for Cognitive Assistants "CoSy" FP6-004250-IP (Henrik Jacobsson), by grant 451-04-043 of the Netherlands Organization for Scientific Research (NWO) (Stefan Frank) and by the ICEA EU project (IST-027819) (Diego Federici).

References

[1] H. Jacobsson, "The crystallizing substochastic sequential machine extractor - CrySSMEx," *Neural Computation*, vol. 18, no. 9, pp. 2211–2255, 2006.
[2] J. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Compilation*. Addison-Wesley, 1979.
[3] H. Jacobsson, "Rule extraction from recurrent neural networks: A taxonomy and review," *Neural Computation*, vol. 17, no. 6, pp. 1223–1263, 2005.
[4] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge Based Systems*, vol. 8, no. 6, pp. 373–389, 1995.
[5] A. Paz, *Introduction to probabilistic automata*. Academic Press, 1971.
[6] H. Jacobsson, "Rule extraction from recurrent neural networks," PhD thesis, Dept. of Computer Science, University of Sheffield, June 2006.
[7] S. L. Frank, "Strong systematicity in sentence processing by an Echo State Network," in *Proceedings of ICANN 2006, Part I*, ser. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2006, vol. 4131, pp. 505–514.
[8] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Advances in neural information processing systems*, S. Becker, S. Thrun, and K. Obermayer, Eds. Cambridge, MA: MIT Press, 2003, vol. 15, pp. 593–600.
[9] S. Frank, "Learn more by training less: systematicity in sentence processing by recurrent networks," *Connection Science*, vol. 18, no. 3, pp. 287–302, 2006.
[10] J. A. Fodor and Z. W. Pylyshyn, "Connectionism and cognitive architecture: a critical analysis," *Cognition*, vol. 28, pp. 3–71, 1988.
[11] R. F. Hadley, "Systematicity in connectionist language learning," *Mind & Language*, vol. 9, no. 3, pp. 247–272, September 1994.
[12] H. Jacobsson, G.-J. Kruijff, and M. Staudte, "From rule extraction to active learning symbol grounding," in *Proceedings of ICRA 2007 Workshop on Concept Learning for Embodied Agents*, Rome, Italy, April 2007.