

CrySSMEx, a Novel Rule Extractor for Recurrent Neural Networks : Overview and Case Study

Henrik Jacobsson and Tom Ziemke

University of Skövde, School of Humanities and Informatics
P.O. Box 408, SE-541 28 Skövde, Sweden
{henrik.jacobsson,tom.ziemke}@his.se

Abstract. In this paper, it will be shown that it is feasible to extract finite state machines in a domain of, for rule extraction, previously unencountered complexity. The algorithm used is called the Crystallizing Substochastic Sequential Machine Extractor, or **CrySSMEx**. It extracts the machine from sequence data generated from the RNN in interaction with its domain. **CrySSMEx** is parameter free, deterministic and generates a sequence of increasingly deterministic extracted stochastic models until a fully deterministic machine is found.

1 Introduction & background

The problem of extracting rules, or finite state machines, from recurrent neural networks (RNN Rule Extraction, or RNN-RE) has occupied a number of researchers on and off during the last 15 years. The achievements of this research have recently been compiled into a review [1] which identified four common ingredients of RNN-RE algorithms:

1. *quantisation* of the continuous state space of the RNN, resulting in a discrete set of states,
2. state and output *generation* (and observation) by feeding the RNN input,
3. rule *construction* based on the observed state transitions,
4. rule set *minimisation*.

These four constituents are often quite distinguishable in the algorithms. For example, in the most commonly used algorithm [2] (1) an equidistant grid partitioning of the state space was used for quantisation, (2) states were generated by a breadth-first search, (3) the rules were constructed by transforming the transitions in the quantised space into a deterministic finite automata, and (4) the rules were minimised using a standard minimisation algorithm. In another example [3] (1) a self-organising map was used to quantise, (2) states were generated by observing the network in interaction with its domain, and (3) stochastic rules were induced from these observations (no minimisation in this case).

As pointed out in [1], none of the previously tested quantisation functions have been tailor-made to comply with the specific demands of quantising the state space of a dynamic system, where the state is recursively enfolded onto

itself in interaction with a domain. The used quantisers all build (roughly) on the assumption that spatial neighbours should be merged and spatially separated points kept apart. The problem with this approach is that in the RNN, states that are very similar, spatially, may be very different, functionally in the RNN.

We came to the conclusion that the main problem of earlier solutions is in fact the lack of integration of the above presented constituents. More specifically, the quantiser should take into account the dynamics of the RNN through closer integration with the other constituents, so that the state space is quantised based on its functional context as set of a states of a dynamic system in interaction with a domain rather than ordinary points of a Euclidean space.

This realisation was the ground for the development of a novel algorithm named **CrySSME_x**¹ (Crystallizing Substochastic Sequential Machine Extractor, see Algorithm 1) which builds on a novel hierarchical quantisation algorithm (named Crystalline Vector Quantiser, CVQ) which can merge and split states based on their dynamical properties in the RNN. By the introduction of this algorithm, a novel form of state machine, a substochastic sequential machines (SSM) is also introduced. SSMs can take into account that some data may be missing in the data collected from the RNN. **CrySSME_x** is parameter free and generates a list of SSMs with monotonously increasing determinism.

2 Experiments

The main purpose of the experiments in this paper is simply to show that it is possible to extract a deterministic finite automata from networks in a challenging domain. That it is possible, in theory, to extract finite machines if the RNN is robustly mimicking a regular language recogniser has already been shown [4]. But previous RNN-RE techniques have predominantly been used on quite simple regular binary language classification tasks with relatively few states. The selected domain for this paper is the prediction of the $\mathbf{a}^n\mathbf{b}^n$ -language which has been studied extensively in the RNN domain [5–12]. The network which has been chosen for analysis with **CrySSME_x** is a simple recurrent network (SRN) trained using a genetic algorithm with a fitness proportional to how many of the predictable symbols that were correctly predicted. Strings from the $\mathbf{a}^n\mathbf{b}^n$ -language with $1 \leq n \leq 10$ were generated and augmented in random order both during training and during generation of data (Ω) for **CrySSME_x** to analyse. See [12] for more details on the training (the SRN in question here is actually one of those behind the statistics in that paper) and for a discussion on the importance of random string order for the analysis of RNNs in the $\mathbf{a}^n\mathbf{b}^n$ -domain.

To generate data, the RNN was exposed to a sequence of 5500 symbols (corresponding to 50 strings of each length). The resulting extracted deterministic

¹ Unfortunately, the constituents of the algorithm are quite complex and there is no room for these details here. An open source distribution and an article on **CrySSME_x** are under preparation at the time of submission of this paper. The purpose of the paper is not to present the algorithm as such, but to present the underlying principle of functionally based quantisation and to acknowledge the possibility of extracting rules from domains of a complexity previously not considered in RNN-RE contexts.

$\text{CrySSMEx}(\Omega, \Lambda_i, \Lambda_o)$

Input: Time series data from the RNN, Ω , an input quantisation function, Λ_i , and an output quantisation function, Λ_o .

Output: A deterministic machine M mimicking the RNN.

begin

 Let M be the stochastic machine based on Ω resulting from an unquantised state space (i.e. only one state);

repeat

 Select data relevant for splitting indeterministic states;

 Split quanta in state quantiser according to split data;

 Create M using new state quantiser, Λ_i and Λ_o for quantisation;

if M has equivalent states **then**

 Merge equivalent states;

end

until M is deterministic;

return M ;

end

Algorithm 1: A simplified description of the main loop of **CrySSMEx**. M is created from the observed RNN input, output and state contained in Ω by quantisation of input, output and state space, of which the latter is optimised.

machine is shown in Figure 1 together with the two first indeterministic (and stochastic) machines. **CrySSMEx** always start with an initial machine of only one state, in which only the conditional distribution of output symbols given input symbol is modelled. The quantisation of the state space of the RNN was then refined such that it contained two states from which the output symbols could be uniquely determined given the input symbol. The algorithm continued to select states that gave rise to indeterminism and split them until a machine with only deterministic states were reached. The whole procedure took ten iterations in the main loop of Algorithm 1.

The extracted deterministic SSM and its stochastic predecessors can all parse the same sequence the RNN was tested on. To parse with an SSM, an initial state must be chosen. If nothing is known about the initial state of the RNN, the initial state of the SSM be a uniform distribution over the states (i.e. that all states are equally probable). When the SSM is then fed input symbols from the domain the stochastic state of the SSM “crystallises”, i.e. the SSM becomes more “certain” about what the state of the underlying RNN would have been given the same input history. In other words, the entropy of the state distribution decreases given “evidence” from the input sequence. In the final, deterministic SSM of Figure 1, the machine eventually narrows down the number of possible occupied states to one. A second reason for the word “crystalline” in this work is due to how the CVQ divides the state space gradually, in a process that visually resembles crystallisation. In the $\mathbf{a}^n \mathbf{b}^n$ -RNN of the experiment, the CVQ divides the state space as shown in Figure 2.

Apart from extracting rules from RNNs in the domain where $1 \leq n \leq 10$ (in which the RNN is trained to perfection) SSMs have also been extracted from the

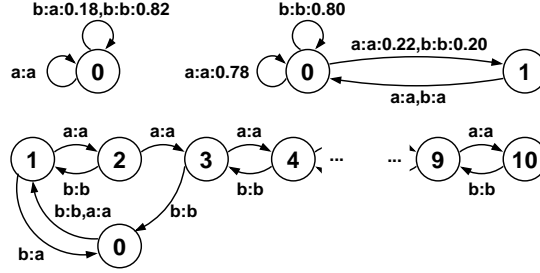


Fig. 1. The two first machines and the last (at iteration ten) in the sequence of machines extracted by **CrySSMEx**. A transition label $\mathbf{x}:\mathbf{y}:p$ is to be read as a transition with \mathbf{x} as input and \mathbf{y} as output and p as the probability of this transition. For example, a transition label “ $\mathbf{a}:\mathbf{b}:0.75$ ” from state 0 to 1 would correspond to that the conditional probability that the next state will be 1 and the output symbol \mathbf{b} , given that the prior state was 0 and the input symbol \mathbf{a} , is 0.75. Where p is 1.0, the probability is omitted from the label. For pairs of transitions between the same pair of states, the labels have in some cases been merged and comma separated to save space.

same RNNs with longer strings. The results varies with what kind of error the RNN makes for longer strings. If it makes no error, it is still trivial for **CrySSMEx** to extract the rules. But if the network cannot predict correctly when exposed to longer strings, extraction is either still trivial, or virtually impossible. In some cases the algorithm had to be aborted when the SSM grew indefinitely (having some 1000 states). Similar results are reached when testing **CrySSMEx** on chaotic systems². The “grammar of mistakes” can obviously be of staggering complexity and this can probably be explained by the near chaotic dynamics of successful RNNs in the $\mathbf{a}^n\mathbf{b}^n$ -domain [8, 9]. Fortunately, all SSMs that are extracted during the search for a deterministic model are in themselves also models of the RNN, and can parse the input sequence just as the final model can. And the more computational resources invested in the iterations of **CrySSMEx**, the more exact will the extracted model be with respect to the underlying RNN, within the domain. There is however a possibility of data starvation if an SSM of a thousand states is extracted from data of just a few thousand time steps.

3 Discussion

In this paper a simple demonstration has been given to show that it is possible to extract machines from an RNN trained on a task that requires it to embed its memory in deeply recursive manner. Apart from the experiment presented here, **CrySSMEx** has been tested on, for example, autonomous chaotic dynamic

² For currently unknown reasons, a similar problem also sometimes occur if Ω is too small.

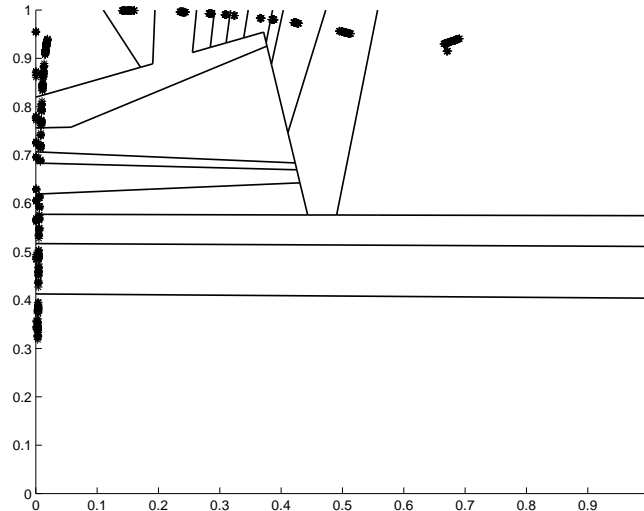


Fig. 2. The state space of the RNN as divided by the quantiser generated by `CrySSMEx` in order to describe the RNN as a finite deterministic system. The actual states occupied by the RNN when it is predicting $\mathbf{a}^n \mathbf{b}^n$ -sequences are also plotted. Note how some disparate points sometimes belong to the same regions while some quite nearby clusters of points are separated.

systems, RNNs (with 10^3 state nodes) with small random weights and RNNs trained on regular languages. The results are overall very promising in that SSMs are extracted reliably and efficiently from all successfully trained networks.

There are of course many open issues and possible enhancements. For example, in the current implementation, `CrySSMEx` requires a symbolic input domain. If this was not the case, `CrySSMEx` could be used on domains where the input is continuous, e.g. sequences of sensory data. There are also a number of experiments that needs to be done to compare `CrySSMEx` to earlier approaches more directly, e.g. how many model vectors would typically be required if k -means is used (as in [13]) instead of the CVQ as quantiser in the $\mathbf{a}^n \mathbf{b}^n$ -domain.

Extracted rules give us a unique window into the underlying system in that we can, in qualitatively new ways, analyse the rules in place of the RNNs. The next step is to let extracted rules be employed in our ambition to understand these networks. For example, it should be possible to query the rules concerning under what exact conditions an RNN commits mistakes and errors. Such information would be ideal for planning retraining of the RNN. The extracted rules can also be used to test whether RNNs are equivalent to each other, something which is not easily derivable from the weights alone. So, if there is a population of networks, trained on the same domain or not, it would be possible to divide them into families of equivalent networks and describe exactly what distinguishes these families from each other. Similar distinctions have previously been made from a dynamic systems theory standpoint [6].

In future work, we would like to cooperate with a number of researchers who together could provide us with access to a large corpus of recurrent neural networks, of different architectures and trained on different data domains, to be analysed with **CrySSMEx**. For the future of the research area as a whole, in order to facilitate cooperation, exchange of experimental setups, and validation of results between individual researchers, it would furthermore be desirable to establish a database of benchmark problems and trained (recurrent) neural networks, similar to the benchmark data repositories that are commonly used in other areas of machine learning research, but with the trained models as the main focus.

References

1. Jacobsson, H.: Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation* **17** (2005) 1223–1263
2. Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z.: Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation* **4** (1992) 393–405
3. Tiño, P., Vojtek, V.: Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences. *Neural Network World* **8** (1998) 517–530
4. Casey, M.: The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation* **8** (1996) 1135–1178
5. Wiles, J., Elman, J.L.: Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent neural networks. In: *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, Cambridge MA: MIT Press (1995) 482–487
6. Tonkes, B., Blair, A., Wiles, J.: Inductive bias in context-free language learning. In: *Proceedings of the Ninth Australian Conference on Neural Networks*. (1998)
7. Rodriguez, P., Wiles, J., Elman, J.L.: A recurrent network that learns to count. *Connection Science* **11** (1999) 5–40
8. Tonkes, B., Wiles, J.: Learning a context-free task with a recurrent neural network: An analysis of stability. In Heath, R., Hayes, B., Heathcote, A., Hooker, C., eds.: *Dynamical Cognitive Science: Proceedings of the Fourth Biennial Conference of the Australasian Cognitive Science Society*. (1999)
9. Bodén, M., Wiles, J., Tonkes, B., Blair, A.: Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units. In: *Proceedings of ICANN 99*, Edinburgh, IEEE (1999) 359–364
10. Bodén, M., Wiles, J.: Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science* **12** (2000) 196–210
11. Gers, F.A., Schmidhuber, J.: LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks* **12** (2001) 1333–1340
12. Jacobsson, H., Ziemke, T.: Improving procedures for evaluation of connectionist context-free language predictors. *IEEE Transactions on Neural Networks* **14** (2003) 963–966
13. Zeng, Z., Goodman, R.M., Smyth, P.: Learning finite state machines with self-clustering recurrent networks. *Neural Computation* **5** (1993) 976–990