Rule Extraction from Recurrent Neural Networks

Submitted to the Department of
Computer Science, University of Sheffield,
for the degree of Doctor of Philosophy

Henrik Jacobsson

June 2006

Abstract

This thesis investigates rule extraction from recurrent neural networks, which takes the form of automated construction of models of an underlying network. Typically the models are expressed as finite state machines and they should mimic the network while being more intelligible. It is argued that rule extraction allows a deeper and more general form of analysis than other, more or less *ad hoc*, methods which are typically applied after the training of the recurrent networks. The first part of this thesis reviews and analyses the development of related techniques. The second part presents a novel algorithm, the *Crystallizing Substochastic Sequential Machine Extractor* (`CrySSMEx`), which efficiently generates a sequence of increasingly refined stochastic finite state models of an underlying system. Novel features of `CrySSMEx` include, for example, freedom from parameters, deterministic extraction, a hierarchical vector quantizer, and a stochastic finite state model which can be constructed also when some data is missing. Experiments show that `CrySSMEx` is, compared to other methods, applicable to a wider range of problems (such as high-dimensional or chaotic dynamic systems). Finally, the field is discussed from a more theoretical perspective in terms of scientific methodology targeted at simulated systems. It is suggested that a rule extractor (or Empirical Machine) can actively select data from the system it is set to model by continuously targeting the weakest point of its currently strongest model. These automated experimenters can, in turn, be made part of a framework (or Popperian Machine) in which theories about populations of systems are generated and tested in order to establish falsifiable statements. These statements should have a high empirical content and thus concisely describe emergent, and previously unknown, properties of the systems.

Acknowledgements

First, I want to thank my supervisors, Tom Ziemke and Amanda Sharkey for supporting me and believing in me all these years, even when I sometimes doubted I would get finished. I also want to thank everybody else who has ever expressed any critical and insightful opinion about my work. While writing this thesis, I gradually *became* the thesis. To work on this thesis was the loneliest thing I have ever done. The slightest indication of someone giving thoughts about my work has therefore also been an act of recognizing a part of my existence. Primarily I want to thank André Grüning for many inspiring discussions and for giving me the chance to visit the Max Planck Institute for Mathematics in the Sciences in Leipzig and the Department of Psychology, University of Warwick (thanks also to Jürgen Jost and Nick Chater from respective organization). Others, that have also given active feedback on my work, include Peter Tiňo, Ron Sun, Gunnar Buason, Andreas Hansson, Anders Jacobsson, Claudina Riguetti, Katarzyna Ziółkowska, many members of our department research groups, and a number of anonymous reviewers. I must thank also Vera Lindroos for helping me to significantly refine the language in the final version of the thesis (and also Diego Federici helped me making the abstract more concise). For helping me mature as a C++ programmer, Henrik Grimm has been a significant support. Peter Bengtsson has also helped me greatly by inspiring conversations and by implementing the command line parser, `nuqneH`. In this respect I must also thank the whole open source community for providing almost every software tool I have needed, e.g., `Xemacs`, `GCC`, LaTeX, BibTeX, `Graphviz`, Boost C++ libraries, `Cygwin`, `CVS` etc. I must thank the Swedish AI Society and the (former) Swedish Society for Learning Systems for providing national networks. In addition, I owe Lars Niklasson for giving me the opportunity to "infiltrate" these organizations.

I must also thank all my friends and office mates who helped me to do other things than just working on my research (you know who you are). Thanks also to my family for supporting me all the time!

Primarily, however, I must thank the University of Skövde for providing a great platform for my research. The university deserves my greatest gratitude for financing my studies and for the faith in me and my work. I have been given enormous freedom to follow my own ideas, a freedom I have just recently begun to understand the uniqueness of. This freedom had some interesting side effects, though. The former head of department, Stig Emanuelsson, told me more than a year before I started my PhD studies in Sheffield (based on recollection): "A supervisor? But you seem to do well without a supervisor!" (I disagreed). I must therefore also especially thank Fredrik Linåker, who in the beginning of this project was a semi-supervisor and continued to be a great support.

List of Publications

Journal publications

Jacobsson, H. (2006). The Crystallizing Substochastic Sequential Machine Extractor – `CrySSMEx`. *Neural Computation*, 18(9), 2211–2255.

Jacobsson, H. (2005). Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review. *Neural Computation*, 17(6), 1223–1263.

Jacobsson H. and Ziemke T. (2003). Improving Procedures for Evaluation of Connectionist Context-Free Language Predictors[1]. *IEEE Transactions on Neural Networks*, 14(4), 963–966.

Linåker, F. and Jacobsson, H. (2001). Learning Delayed Response Tasks through Unsupervised Event Extraction. *International Journal of Computational Intelligence and Applications*, 1(1), 413–426.

Conference and workshop publications

Jacobsson, H. and Ziemke, T. (2005). Rethinking rule extraction from recurrent neural networks. In A. d'Avila Garcez, J. Elman & P. Hitzler (Eds.), *IJCAI-05 workshop on neural-symbolic learning and reasoning.*

Jacobsson, H. and Ziemke, T. (2005). `CrySSMEx`, a novel rule extractor for recurrent neural networks : Overview and case study. In W. Duch, J. Kacprzyk, E. Oja & S. Zadrozny (Eds.), *Artificial neural networks: Formal models and their applications - ICANN 2005 - part II* (pp. 503-508). Berlin: Springer.

Stening, J., Jacobsson, H. and Ziemke, T. (2005). Imagination and Abstraction of Sensorimotor Flow: Towards a Robot Model. In: *AISB'05: Proceedings of the Symposium on Next Generation Approaches to Machine Consciousness - Imagination, Development, Intersubjectivity and Embodiment* (pp. 50–58). The Society for the Study of Artificial Intelligence and the Simulation of Behavior, UK. ISBN 1-902956-46-8.

Linåker, F. and Jacobsson, H. (2001). Mobile Robot Learning of Delayed Response Tasks through Event Extraction: A Solution to the Road Sign Problem and Beyond. In *IJCAI'01: Seventeenth International Joint Conference on Artificial Intelligence*, pp. 777–782, San Fransisco: Morgan Kaufmann.

[1]Included verbatim in Appendix C.

Bodén, M., Jacobsson, H. and Ziemke, T. (2000). Evolving context-free language predictors. In In D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee & H.-G. Beyer (Eds.), *GECCO'00: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1033–1040, San Fransisco: Morgan Kaufmann.

Jacobsson, H. and Olsson, B. (2000). An Evolutionary Algorithm for Inversion of ANNs. In Wang, P.P., ed., *JCIS'00: Proceedings of The Fifth Joint Conference on Information Sciences*, pp. 1070–1073, Association for Intelligent Machinery.

Technical reports

Jacobsson, H. and Ziemke, T. (2003). Reducing Complexity of Rule Extraction from Prediction RNNs trough Domain Interaction[2], Tech. report no. *HS-IDA-TR-03-007.*

[2]Included verbatim in Appendix D.

Contents

List of Figures

List of Tables

List of Algorithms

Seek simplicity, and distrust it.

—

Alfred North Whitehead

Chapter 1

Introduction

Computer simulations are conducted for many reasons. Simulations are in many ways playgrounds for entertainment and education in more or less serious contexts. Computer games may provide realistic simulated environments that entail an entertainment value as well as the possibility to put yourself in situations in which you may learn new things. For example, airline pilots, surgeons and military commanders can safely practise their skills in simulated environments where their actions will not have any fatal consequences. In a similar fashion, scientists can create simulations in which they can test their theories in ways that are impossible or very expensive in the real world. They may even test theories by simulating systems with no obvious counterpart in the physical world. This thesis focuses on these last types of simulated systems in which there is no human intervention during the execution of the simulation and where an understanding of the system requires a systematic analysis of it.

These simulations are in themselves small *artificial universes* with their own laws and their own emergent orders stemming from these laws. Many (but far from all) of these universes are created to reflect phenomena in our own Universe[1]. They are then carefully designed so that their laws mimic the natural laws of our Universe to our best understanding. The emergent dynamics of these simulators are consequently used to validate our theories of the laws of the Universe. If an emergent behaviour can be observed in the simulated universe as well as in Reality, the theories underlying the construction of the simulator become validated. Moreover, as we are omniscient gods from the perspective of the simulated universe, we can observe phenomena not readily apparent in the physical Universe.

For example, consider a helioseismologist, a scientist who analyses the interior of the Sun through observations of the oscillations and sound waves that can be

[1]Universe (and sometimes Reality) is here written with capital letters to emphasize its importance and to clearly separate it from simulated counterparts.

observed on the surface of the Sun. There is of course no direct way to observe the interior of the Sun. If a detailed theory of the Sun is constructed, however, with specifications of constituents, densities, magnetic fields, temperatures etc., then this theory can be the basis of a simulation. The scientist thus may visualize the interior of the simulated Sun by projecting selected slices of the Sun. These visualizations may help the scientist to understand the Sun at a significantly deeper level than would otherwise be possible. The scientist may even become genuinely surprised by the results of the simulation, despite them being created by the scientist in the first place. Unforeseen predictions stemming from the simulation may later be corroborated with observations of the Sun. Observations that without the simulator would perhaps lack a proper explanation.

The situation for the helioseismologist can be taken as an example of how scientists typically need to study nature only through indirect observations of the underlying system. The true nature of what is studied may always be hidden. Plato's "allegory of the cave" is often used to illustrate this situation in scientific studies:

> Behold! human beings living in a underground den [...]; here they have been from their childhood, and have their legs and necks chained so that they cannot move, and can only see before them, being prevented by the chains from turning round their heads. Above and behind them a fire is blazing at a distance, and between the fire and the prisoners there is a raised way; and you will see, if you look, a low wall built along the way, like the screen which marionette players have in front of them, over which they show the puppets. [...] And do you see, I said, men passing along the wall carrying all sorts of vessels, and statues and figures of animals made of wood and stone and various materials, which appear over the wall? [...] To them, I said, the truth would be literally nothing but the shadows of the images. (*The Republic - book VII* (Plato, 1991), pp. 253–354)

Just as Plato's prisoners, scientists can typically only observe secondary phenomena, e.g., the sound waves on the surface of the Sun, stemming from hidden activity deep within it. The shadows on the wall are clues about the "real" objects that we may never observe directly. We can try to explain the nature of the real objects, but not entirely as we please. Scientific methodology strictly governs what explanations and guesses that are acceptable and the very fact that the "true" Reality is not directly observed colours the scientific meted. Since Reality cannot simply be scientifically described just as we immediately perceive it, and intuitively understand it, we restrict scientific explanations to the ones that can be tested. More precisely, we may restrict ourselves to accepting only explanations that could be *falsified* through experiments (Popper, 1990). Therefore, scientists do not entirely comply with the description of Plato's prisoners since they have an urge (without taking their eyes of the projection) to indirectly interact with the objects between the fire and the wall. That is how scientists learn, by active interaction through experiments. Based on previous experience, conjectures are born in the mind of the imprisoned scientists, and further experiments helps to refine the scientific knowledge by refuting ideas of Reality that are false (Popper, 1990).

In simulated universes, however, we are no longer the prisoners, we are instead the creators of the cave, the fire, the passage way and the wall onto which images of objects of our choice are projected. Yet, one may argue that we still put ourselves in the position of the prisoners once we have constructed the cave. We are still bounded by our desire to understand the Reality which inspired us to build the cave in the first place. But we do not belong in our simulated caves such as the prisoners envisioned by Socrates in Plato's text. They are imprisoned since childhood and their entire perception of the universe is the projection. When we analyse our simulated systems, i.e. when we "enter the cave" of our creation, we bring with us the experience built from the experience in our own "cave" in which we are the true prisoners; the Universe. This experience may largely overlap with the experience of an imagined life-long cave occupant, but the simulator may be entirely new to us. For example, if the simulator is not designed to encompass any real phenomena, but is instead an abstract mathematical construction, then it may become difficult for us to fully interpret any projection. Some simulations, such as of artificial neural networks, or of artificial life, are not meant to reflect Reality more than in a very abstract sense. The projection in this case may be visualizations of the system, or data logs from simulating the system. Since the projection is chosen by ourselves and our limited understanding of the system, it may not be the most informative of possible projections. For example, the helioseismologist may visualize the Sun by slicing it up like an onion whereas he/she could learn more by other more counter-intuitive projections in space-time that perhaps more appropriately preserve information about relevant dynamics of the Sun.

We may also take the seat of the prisoner for other reasons than direct and aesthetic visualizations of the system. There may simply be a concrete need for projecting a complex simulated system onto a more comprehensible plane. One reason for this is that the system may be completely abstract to us. It could also consist of an enormous number of state variables, changing over long sequences of simulated time. The system must then typically be projected, through visualizations or otherwise, just to enable us to comprehend it. If, for example, the dynamically changing simulated Sun is represented by a finite set of elements in a 3D-lattice of $100 \times 100 \times 100$ elements of 10 values each (e.g., chemical composition, temperature etc.) and this system is simulated for 10^2 time steps, then the amount of data is on the order of 10^9. It can be argued that the scientific value of this data does not emerge until the scientist can put forward statements about the system. But what if the system is not a Sun? What if it was created for other reasons than simulating aspects of Reality? The terminology of the scientist, bounded by his/her perceived Reality, may not be appropriate or rich enough for the task of describing the 10^9 data points in a truly meaningful way. It may become difficult for us to remain the scientific prisoners in our simulated universes/caves. We may lack the intuition for it.

Moreover, we should not underestimate our ability to create such caves once we have the means of creating one of them. The helioseismologist may for example continue to generate hypothetical stars indefinitely, each one representing a new simulated cave. For each different star, different research questions may be of relevance. Perhaps the acoustics of a red dwarf have a rich variety of self-sustained harmonic sound waves that are never seen in a larger star? Each such potential richness of behaviour of each individual system may require the helioseismologist to assume the role of a scientifically reasoning prisoner in each cave. Even if the intuition of the helioseismologist may, after years of training, be sufficient to correctly understand each system, he may for every correctly analysed system create yet another one, e.g., a little more helium, some more metals, a somewhat younger star, etc.

In some areas of research the creation of novel systems lies at the very core of their methodology, especially for artificial neural networks (ANNs), artificial life, genetic algorithms/programming (GA/GP), etc. For these areas, the creation of the system need not be grounded in a sound solid theory of the Universe since they are not meant to correspond to any reality. The systems are therefore easy to create. In fact, long sequences of such systems are created constantly, just to evaluate their fitness in solving the task which they are supposed to solve. And, free from any basis in our Reality, they are not easy to understand, intuitively. In such cases we typically put ourselves directly in the prisoner's position only for a handful of examples. Instead, the analysis of the projection is typically reduced to a simple automated data collector, e.g., a numeric performance evaluation. For example, the results of genetic programming may be a couple of thousand potential systems that all solve, or partially solve, a particular problem. Typically only the best of these are further analysed for the purpose of describing and understanding them. But each one of these systems would require "a scientific prisoner" of its own, in order to be analysed with scientific methodology. The simple collection of data, in the same way for each system, corresponds to letting each cave be inhabited by *dummy prisoners*, not learning from the experience stemming from each system. Just as each of the helioseismologist's hypothetical stars may require its own research questions to be properly scientifically analysed, automatically created systems may be widely different from each other despite being created by the same mechanism. The dummy prisoner we put in our place, may therefore be inadequate.

In this thesis, I will suggest that, in the place of the passive data collecting prisoners of these caves, put in prisoners that can more actively interact with the system behind the fire. I suggest that instead of only being the creators of the cave, fire and projected objects, we should also create the prisoners themselves. One set of (artificial) prisoners per cave, situated in their own universe, analysing and interacting to learn about their world. Of course, in this thesis, the set of potential caves will be very modest. Although I will suggest that their behaviour is strictly regulated by the same principles of scientific methodology that govern human scientists, the intelligence of these prisoners will also be modest. However, the principle could apply in a broader sense: *analysis of artificial systems conducted by artificial scientific intelligence*. The "scientific" aspect of these prisoners will be more accentuated at the end of this thesis where Popper's *falsificationism* will be suggested as a guideline for evaluating statements about simulated universes.

But how should these prisoners be created? And is it feasible to create them at all? The problems of artificial intelligence are by no means trivial. However, as described above, simulated systems typically act as playgrounds for learning without any risks. Simulated systems are more easily controlled and are already used many times as development environments when creating artificial intelligences such as learning systems. For example, robot controllers that should learn from experience are often trained completely in simulation (e.g. Meeden, 1996; Ziemke, 2000). As will be discussed in Chapter 18, it is a simple matter of fact that simulated systems are much easier to integrate with AI learning techniques since there is no noisy Reality which must be indirectly interacted with, through sensors and actuators. This is also one of the main criticisms of early AI techniques; they were very successfully applied, but only on simple toy-world problems, e.g., the blocks world. Simulated systems of today, however, are not necessarily trivial, nor necessarily sufficiently analysed by human scientists either, since the very same scientists can easily create more systems than they can ever fully analyse in their life times. Many simulated systems may therefore both be the perfect *playground* for artificial scientific intelligence as well as domains where there is a *need* for such techniques.

In this thesis, the Plato caves in question are instantiations of simulated trained recurrent neural networks (RNNs). For RNNs (e.g. Kremer, 2001; Kolen & Kremer, 2001) it has been natural to analyse them as finite state machines (FSMs), partly due to their common source of origin (McCulloch & Pitts, 1943), and partly due to the fact that they have often been trained to perform regular language recognition (e.g. Cleeremans, McClelland & Servan-Schreiber, 1989; Christiansen & Chater, 1999). This has resulted in the development of algorithms for transforming one model into another, i.e. from RNNs into FSMs. These transformations are made through observation of the RNN and the generation of FSM descriptions of these observations. In other words, the "Plato prisoners" in this case are therefore algorithms that learn to create FSM descriptions of RNNs through observations of RNN projections. The projections are not indirect in the sense that the perception of Plato's prisoners is indirect. Instead, these "projections" typically contain every single aspect of the RNNs (but limited to the RNN as put in specific contexts).

The problem for human observers of RNNs is that their behaviour is not a consequence of the physical laws we are used to. They may be counter-intuitive and complex to understand, even if moderately sized. The FSM descriptions of some RNNs may indeed be fairly complex too, but they have the advantage of having a clearly defined syntax and semantics. With a clear formal specification, the FSMs can be used as a proxy, in place of the actual underlying RNN, for inference of new (falsifiable, Popper (1990)) statements about the RNNs (cf. Chapter 18).

The broad structure of this thesis is as follows: Part I presents a survey and critique of rule extraction algorithms that generate FSMs mimicking specific RNNs. In Part II a novel rule extraction algorithm is suggested and experiments are conducted to establish the efficiency of it. Finally, Part III discusses several, more or less, speculative future directions based on the connection to scientific methodology.

Part I

Rule Extraction from Recurrent Neural Networks - A Survey

Chapter 2

Introduction to Part I

In this part of the thesis, techniques for extracting rules (or finite state machines) from discrete-time recurrent neural networks (DTRNNS, or simply RNNs) are reviewed. A new taxonomy for classifying existing techniques will be suggested, and existing techniques will be presented and evaluated. A list of open research issues that need to be addressed will also be suggested[1].

By RNN-RE I refer to the process of finding/building symbolic computational models/machines that mimic the RNN to a satisfactory degree. The connection between RNNs and symbolic models of computation is almost as old as the study of RNNs themselves since the origins of these fields are largely overlapping. The study of neural networks once coincided with the study of computation in the binary recurrent network implementations of finite state automata of the theoretical work on nervous systems by McCulloch and Pitts (1943) (an interesting overview of this topic is found in Forcada, 2002.) This common heritage has been flavouring the development of the digital computer although our current computer systems are very far from being models of the nervous system.

In the early 1990s, the research on recurrent neural networks was revived. When Elman introduced his, quite well known, simple recurrent network (SRN) (Elman, 1990), the connection between finite state machines and neural networks was again present from the start. In his paper, the internal activations of the networks were explicitly compared to the states of a finite state machine.

In theory, RNNs are Turing machine equivalent[2] (Siegelmann & Sontag, 1995),

[1] Chapters 2–7 have been published in a very similar form in Jacobsson (2005).

[2] Actually McCulloch and Pitts (1943) determined this equivalence already in 1943, for discrete

and can thus compute whatever function any digital computer can compute. But we also know that to get the RNN to perform the desired computations is very difficult (Bengio, Simard & Frasconi, 1994). This leaves us in a state of knowledge vacuum; we know that RNNs *can be* immensely powerful computational devices, and we also know that finding the instantiations of RNNs that perform these computations could very well be an insurmountable obstacle, but we do not have the means for efficiently determining the computational abilities of our current RNN instantiations. On a less theoretical level, we can simply evaluate the performance of different RNNs in order to see to which degree a learning problem is solved for a specific domain. Such studies are conducted in virtually all papers applying RNNs on a domain, and in some cases more systematic studies are presented (Miller & Giles, 1993; Horne & Giles, 1995; Alquézar, Sanfeliu & Sainz, 1997). But even something as simple as evaluating the performance of an RNN on a specific domain has some intrinsic problems since implicit aspects of the evaluation procedure can have a significant impact on the estimated quantitative performance (Jacobsson & Ziemke, 2003a; Jacobsson, 1999) (cf. Appendix C).

Actually, the analysis problems may lead to the use of too simplistic models, e.g., smaller networks and toy problem domains, just to be able to analyse (or visualize) the results. One may wonder how many published networks with just two or three state (or hidden) nodes had their specific topology chosen just to make the plotting of their internal activations possible. Thus, what is required is in-depth analyses of RNN instantiations to uncover the actual behaviour of RNN instantiations without the need for "manually" analysing visualizations of the RNN behaviour. An efficient rule extraction technique may be the best tool for such analyses.

2.1 Topic delimitation

Since the early nineties, an abundance of papers on recurrent neural networks has been written[3], and many of them have dealt explicitly with the connection between RNNs and state machines. Many contributions have been theoretical, establishing the connection between (analogue) RNNs (or other dynamic systems) and tradi-

networks (Medler, 1998).

[3] Many of these are summarized in Kremer (2001) and Barreto, Araújo and Kremer (2003).

tional (discrete) computational devices (e.g. Crutchfield & Young, 1990; Servan-Schreiber, Cleeremans & McClelland, 1991; Crutchfield, 1994; Kolen, 1994a; Horne & Hush, 1994; Siegelmann & Sontag, 1995; Casey, 1996; Tiňo, Horne, Giles & Collingwood, 1998; Jagota, Plate, Shastri & Sun, 1999; Omlin & Giles, 2000; Sima & Orponen, 2003; Hammer & Tiňo, 2003; Tiňo & Hammer, 2003). While these papers cover a wide spectrum of highly interesting and important theoretical insights, this thesis will not dwell on these theoretical issues. Firstly, because it is not the focus of the survey-part of this thesis. Moreover, some of these papers already resemble surveys themselves, summarizing earlier findings.

At a pragmatic level, these are papers describing techniques for transforming state machines into RNNs (rule insertion) and/or for transforming RNNs into state machines (rule extraction) (e.g. Omlin & Giles, 1992; Giles & Omlin, 1993; Das, Giles & Sun, 1993; Alquézar & Sanfeliu, 1994a; Omlin & Giles, 1996a, 1996c; Omlin, Thornber & Giles, 1998; Omlin & Giles, 2000; Carrasco, Forcada, Muñoz & Ñeco, 2000; Carrasco & Forcada, 2001). This thesis, however, deals exclusively with algorithms for performing *rule extraction* from RNNs.

Unfortunately, there is no space for a discussion of the analysis tools of RNNs other than just RE. Since there are a multitude of methods used to analyse RNNs, a survey on this issue should definitely be written as well. A brief (and most probably inconclusive) list of examples of other analysis tools that have been used on RNNs includes:

- Hinton diagrams (e.g. Hinton, 1990; Niklasson & Bodén, 1997),
- hierarchical cluster analysis (e.g. Cleeremans et al., 1989; Elman, 1990; Servan-Schreiber, Cleeremans & McClelland, 1989; N. E. Sharkey & Jackson, 1995; Bullinaria, 1997),
- simple state space plots (e.g. Giles & Omlin, 1993; Zeng, Goodman & Smyth, 1993; Gori, Maggini & Soda, 1994; Niklasson & Bodén, 1997; Tonkes, Blair & Wiles, 1998; Tonkes & Wiles, 1999; Rodriguez, Wiles & Elman, 1999; Rodriguez, 1999; Tabor & Tanenhaus, 1999),
- activation values plotted over time (e.g. Husbands, Harvey & Cliff, 1995; Meeden, 1996; Ziemke & Thieme, 2002),
- iterated maps (e.g. Wiles & Elman, 1995),

- vector flow fields (e.g. Rodriguez et al., 1999; Rodriguez, 1999),
- external descriptive behaviour analysis of RNN based autonomous robotic controllers (e.g. Husbands et al., 1995; Meeden, 1996),
- weight space analysis (e.g. Bodén, Wiles, Tonkes & Blair, 1999; Tonkes & Wiles, 1999),
- dynamic systems theory (e.g. Tonkes et al., 1998; Rodriguez et al., 1999; Rodriguez, 1999; Bodén, Jacobsson & Ziemke, 2000),
- and ordinary quantitative evaluations of RNN performance for different domains (basically every single paper where an RNN is applied).

Unlike previous surveys of rule extraction (Andrews, Diederich & Tickle, 1995; Tickle, Andrews, Golea & Diederich, 1997, 1998), this thesis deals exclusively with rule extraction from *recurrent* neural networks (resulting in quite different evaluation criteria than in previous RE surveys, as Chapter 4 illustrates). In fact, many of the RE approaches for non-recurrent networks could potentially be used on RNNs, or at least on non-recurrent networks in temporal domains (e.g. Craven & Shavlik, 1996; R. Sun, Peterson & Sessions, 2001). There are also other symbolic learning techniques for "training" finite automata on symbolic sequence domains directly, without taking the extra step of training a neural network, which could be mentioned (R. Sun & Giles, 2001; Cicchello & Kremer, 2003). While these techniques are certainly interesting in themselves and should also be compared to RNN-RE techniques experimentally, they are not further examined in this thesis.

To summarize, this part of the thesis (i.e. Chapters 2–7) is focused solely on RNN-RE techniques, but this field is closely related to the above mentioned areas[4]. It may also be worth mentioning that, as a review of techniques, the descriptions are not meant to be tutorials. Thus, for readers interested in implementing the algorithms, consulting the cited papers should be more helpful.

2.2 Overview of Part I

Firstly, Chapter 3 describes RNNs, finite state machines, and common characteristics of RNN-RE algorithms. The evaluation criteria underlying the construction

[4]As well as a number of areas that are related based on coincidental overlap rather than tradition (cf. Chapter 15).

of a taxonomy for appropriately classifying and describing RNN-RE algorithms are described in Chapter 4. The techniques are described in Chapter 5 and are consequently discussed in light of the evaluation criteria in Chapter 6. The open research issues are summarized in Chapter 7.

Chapter 3

Background

An RNN processes sequences of data (input) and generates responses (output) in a discrete time manner. The RNN processes information by using its internal continuous state space as an implicit, holistic memory of past input patterns (Elman, 1990). In the extraction of rules from an RNN, the continuous state space is approximated by a finite set of states and the dynamics of the RNN is mapped to transitions among this discrete set of states.

A brief definition of what constitutes a recurrent neural network in the scope of this thesis follows. In addition, a concise introduction to finite state machines (FSMs) will also be provided, since the extracted rules are typically represented as such. A more detailed description of what RNN-RE algorithms typically constitute will then follow.

3.1 Recurrent neural networks

To provide a detailed review of the achievements in RNN research and the vast variety of different RNN architectures is far beyond the scope of this thesis. Instead, a set of identified common features of most RNN architectures will be described at an abstract enough level to hopefully not only incorporate most networks to which the existing RNN-RE algorithms could be applied but also abstract enough to see the striking similarities of RNN computation with the computation in finite state machines (see Definition 3.2). Readers with no prior experience of RNNs can find more detailed descriptions and well developed classifications of RNNs in Kolen and

Kremer (2001), Kremer (2001) or Barreto et al. (2003).

Only a few of the many RNN architectures have been used at all in the context of rule extraction, e.g., simple recurrent networks (SRNs, Elman, 1990) and more commonly second-order networks (e.g., Sequential Cascaded Networks, SCNs, Pollack, 1987). These models differ somewhat in their functionality and how they are trained. But the *functional dependencies* are, at some level of abstraction, basically the same, which is exploited in the definition below.

Definition 3.1 A *Recurrent Neural Network* R is a 6-tuple $R = \langle I, O, S, \gamma_s, \gamma_o, \mathbf{s}^0 \rangle$ where, $I \subseteq \mathbb{R}^{n_i}$ is a *set of input vectors*, $S \subseteq \mathbb{R}^{n_s}$ is a *set of state vectors*, $O \subseteq \mathbb{R}^{n_o}$ is a *set of output vectors*, $\gamma_s : S \times I \rightarrow S$ is the *state transition function*, $\gamma_o : S \times I \rightarrow O$ is the *state interpretation function*, and $\mathbf{s}^0 \in S$ is the *initial state vector*. $n_i, n_s, n_o \in \mathbb{N}$ are the *dimensionalities* of the input, state and output spaces respectively. $\square$

Often the input, state and output are restricted to *hypercubes* with all elements limited to real numbers (or, of course, rational approximations of real numbers when simulated) between zero and one or minus one and one. When training the networks, the two functions γ_s and γ_o are typically adjusted to produce the desired output according to some training set. For a sequence of input vectors $(\mathbf{i}^1, \mathbf{i}^2, \ldots, \mathbf{i}^\ell)$ the state is updated according to $\mathbf{s}^t = \gamma_s(\mathbf{s}^{t-1}, \mathbf{i}^t)$ and the output according to $\mathbf{o}^t = \gamma_o(\mathbf{i}^t, \mathbf{s}^{t-1})$. The functional dependencies are depicted in Figure 3.1.

Note that the weights, biases, activation functions and other concepts typically associated with neural networks are all hidden in the state transition function γ_s and state interpretation function γ_o. This is because, as far as RNN-RE algorithms are concerned, the fact that the networks have adaptive weights and can be trained, is of less importance. An interesting consequence of the abstract nature of this RNN description, which is also all that is required to continue describing RNN-RE algorithms, is that it reveals something about the portability of the algorithms[1] (cf. Section 4.2). There are simply not many assumptions and requirements of the underlying RNNs, which means that they are portable to more RNN types than they would be otherwise. However, there are a few assumptions, e.g., that states

[1]To ensure high portability in my own suggested technique, a slightly more general definition will be given in Part II which does not include an initial state (Definition 9.1 on page 57).

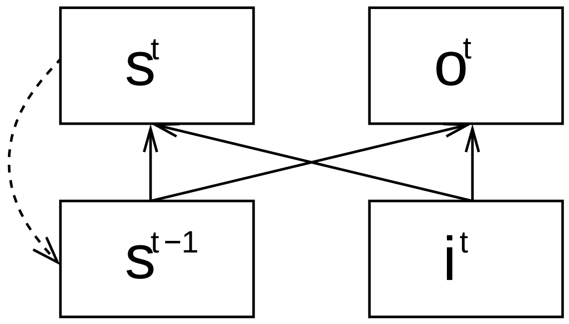


Figure 3.1: The functional dependencies of the input, state and output of an RNN. This is a Mealy-type RNN, i.e. where the output is determined by state and input together. For some RNNs it may be more appropriate to describe them as Moore machines where the output can be determined from the state alone. This can however be achieved in the Mealy machine by simply letting the input domain's influence over the output be possible in theory but non-existent in practice. Therefore the Mealy machines encompass also Moore machines.

should cluster in the state space as a result of the training (Cleeremans et al., 1989; Servan-Schreiber et al., 1989). Some, more implicit, assumptions are also the target of some of the criticisms of RNN-RE (Kolen, 1993, 1994a), which will be discussed in Section 6.7 (more implicit assumptions are also discussed in Section 6.5).

3.2 Finite state machines

The rules extracted from RNNs are almost exclusively represented as finite state machines (FSMs). The following description is kept brief. For a full discussion of what comprises a regular language and other classes of formal languages, interested readers are referred to Hopcroft and Ullman (1979).

Definition 3.2 A *Deterministic Mealy Machine* M is a 6-tuple $M = \langle X, Y, Q, \gamma_s, \gamma_o, q^0 \rangle$ where, X is the finite *input alphabet*, Y is the finite *output alphabet*, Q is a *finite set of states*, $\gamma_s : Q \times X \rightarrow Q$ is the *transition function*, $\gamma_o : Q \times X \rightarrow Y$ is the *output function*, and $q^0 \in Q$ is the *initial state* (note the similarities with the RNNs in Definition 3.1). $\square$

In cases where the output alphabet is binary the machine is often referred to as a *finite state automaton* (FSA). In an FSA, the output is interpreted as an *accept* or *reject* decision determining whether an input sequence is accepted as a grammatical string or not.

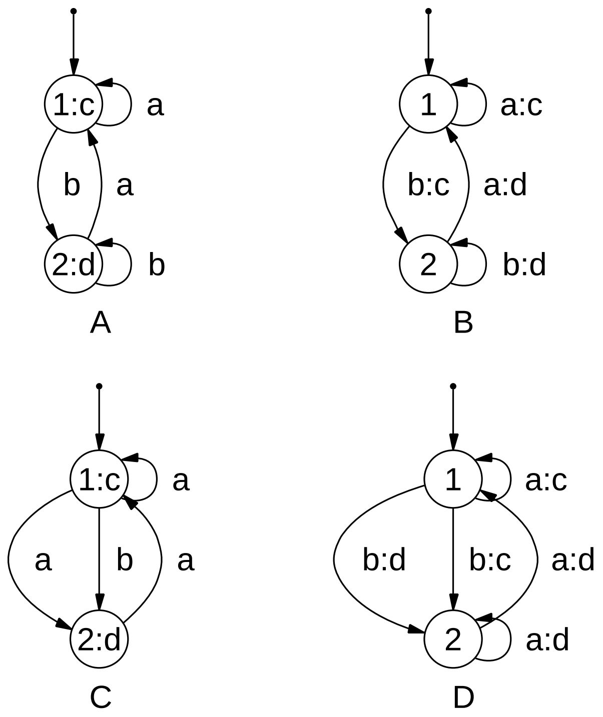


Figure 3.2: Examples of (non-equivalent) different FSM types with $X = \{a, b\}$, $Y = \{c, d\}$, $Q = \{1, 2\}$ and $q_o = 1$; (A) deterministic Moore machine, (B) deterministic Mealy machine, (C) nondeterministic Moore machine, and (D) nondeterministic Mealy machine.

There are actually two different models which can describe an FSM; *Mealy* (as above) or *Moore* machines that, although they are quite different from each other, are computationally equivalent (Hopcroft & Ullman, 1979). Moore machines generate outputs based only on the current state and Mealy machines on the transitions between states, i.e. the output function, γ_o, is for a Moore machine $\gamma_o : Q \rightarrow Y$ and for a Mealy machine $\gamma_o : Q \times X \rightarrow Y$.

In *deterministic* machines, an input symbol may only trigger a single transition from one state to *exactly one* state (as in the definition above). In a *nondeterministic machine*, however, a state may have zero, one or more outgoing transitions triggered by the same input, i.e. the transition function, γ_s, is $\gamma_s : Q \times X \rightarrow 2^Q$ (a function to the power set of Q) instead of $\gamma_s : Q \times X \rightarrow Q$. This means that in a nondeterministic machine, a symbol may trigger one or more transitions from a state, or even no transition at all (since $\emptyset \in 2^Q$). I will denote nondeterministic machines *incomplete* if there is at least one $q \in Q$ and $x \in X$ such that $\gamma_s(q, x) = \emptyset$. Deterministic and nondeterministic finite state machines are computationally equivalent, although nondeterministic machines can typically be much more compact (i.e. have less states) than their deterministic counterpart. Deterministic FSM and determin-

1. Quantization of the continuous state space of the RNN, resulting in a discrete set of states.
2. State and output generation (and output classification, if necessary) by feeding the RNN input patterns.
3. Rule construction based on the observed state transitions.
4. Rule set minimization.

Table 3.1: The common "ingredients" of RNN-RE algorithms.

istic FSA, will be abbreviated DFM and DFA respectively.

In summary, there are four types of FSMs: deterministic Moore machine, deterministic Mealy machine, nondeterministic Moore machine, and nondeterministic Mealy machine, see Figure 3.2 for examples. Moreover, the machines can be *stochastic*[2] as well if transition probabilities are also encoded in the machine.

For a more detailed description of deterministic and nondeterministic, Mealy and Moore machines, proofs of equivalence, and a "standard" minimization algorithm, see Hopcroft and Ullman (1979). For the corresponding theory on stochastic machines, see Paz (1971).

3.3 The basic recipe for RNN rule extraction

The algorithms described in this thesis have many features in common as listed in Table 3.1.

The continuous state space of the RNN needs to be mapped into a finite set of discrete states corresponding to the states of the resulting machine. We will refer to the states of the network as *microstates* and the finite set of quantized states of the network as *macrostates*. The macrostates are basically what the RE algorithm "sees" of the underlying RNN, whereas the actual state of the network, the microstates, are hidden. The act of transforming the microstates into macrostates is a critical part of RNN-RE algorithms (ingredient one in Table 3.1) and is called *quantization*. One macrostate corresponds to an uncountable set of possible mi-

[2]Cf. *stochastic sequential machines* (Paz, 1971), *probabilistic automata* (Rabin, 1963) or the *substochastic* sequential machines as I suggest in Part II.

crostates (only in theory; in practice the RNN is simulated on a computer with finite precision). Therefore deterministic sequences of events at the microstate-level may appear stochastic at the macrostate-level since information is lost in the quantization, e.g., if two microstates $a_1, a_2 \in A$ deterministically transit to microstates $b_1 \in B$ and $c_1 \in C$ respectively, then, at macrostate level, it cannot be determined from observing macrostate A whether the next macrostate will be B or C.

Another common ingredient of RNN-RE algorithms is *systematic testing* of the RNN with different inputs (from the domain or generated specifically for the extraction) and the (macro)states and outputs are stored and used to induce the finite state machine (ingredient two). The third ingredient is the machine construction, a process often conducted concurrently with the state and output generation.

Many times, the generated machine is then minimized using a standard minimization algorithm (Hopcroft & Ullman, 1979), which is the fourth common ingredient of RNN-RE algorithms. FSM minimization is however not part of all algorithms, and can also be considered an external feature, independent of the actual extraction.

Chapter 4

Evaluation Criteria and Taxonomy

Several evaluation criteria have been chosen in order to simplify comparisons and to structure the descriptions of the algorithms in the following chapters. The *rule type*, *quantization method* and *state generation method* can be considered to constitute the main distinguishing features of RNN-RE algorithms, and are therefore been used to structure this survey.

4.1 Main criteria

4.1.1 Rule type

As previously mentioned (in Section 3.2) the rules generated by RNN-RE algorithms are FSMs that are either *deterministic*, *nondeterministic* or *stochastic*. They can also be in a *Mealy* or *Moore* format. In my classification of rule types I have also chosen to distinguish whether the machine (and underlying RNN) is producing a binary *accept/reject* decision at the end of a string (i.e. like an FSA) or if the task is to produce an output sequence of symbols based on the input sequence (typically for prediction).

4.1.2 Quantization

One of the most varying elements of existing RNN-RE algorithms is the *state space quantization* method. Examples of methods used include: hierarchical clustering, vector quantization and self organizing maps (see Section 6.2 for a detailed discussion).

4.1.3 State generation

Another important criterion is the *state generation* procedure for which there are two basic methods: *searching* and *sampling*. These are further described in the descriptions of the algorithms.

4.1.4 Network type and domain

Although not a feature of the extraction algorithm *per se*, the *network type(s)* and in which *domain(s)* each RNN-RE algorithm is used, are explicitly listed for each presented technique.

4.2 Criteria from the ADT taxonomy

Andrews et al. (1995) introduced a taxonomy, the *ADT*[1] *taxonomy*, for RE algorithms which has since been an important framework when introducing new, or discussing existing, RE algorithms (e.g. Schellhammer, Diederich, Towsey & Brugman, 1998; Vahed & Omlin, 1999; Craven & Shavlik, 1999; Blanco, Delgado & Pegalajar, 2000). The five evaluation criteria in the ADT taxonomy are: expressive power, translucency, portability, rule quality and algorithmic complexity. However, for some of their classification aspects all RNN-RE algorithms would end up in the same class and those aspects are therefore not very informative. The ADT taxonomy does, however, provide us with some very useful viewpoints discussed in Chapter 6. Some of the terminology from the ADT taxonomy also appears in various sections of this survey, therefore a brief description of the ADT aspects follows.

4.2.1 Expressive power

The *expressive power* is basically the type of rules generated by the RE and hence subsumed by our rule type criteria. Taking Tickle et al. (1997) and Tickle et al. (1998) into account, ADT identifies four basic classes:

[1] "ADT" comes from the names of the authors, Andrews, Diederich and Tickle.

- propositional logic (i.e. *if...then...else*),
- nonconventional logic (e.g., fuzzy logic),
- first-order logic (i.e. rules with quantifiers and variables), and
- finite state machines.

Almost all rules from RNN-RE algorithms comply with the last category.

4.2.2 Translucency

One of the central aspects in the ADT taxonomy, *translucency*, described as the "degree to which the rule-extraction algorithm 'looks inside' the ANN" is less relevant in this survey since it is not a distinguishing feature of RNN-RE algorithms. ADT initially identified three types of RE algorithms, (i) *decompositional* algorithms where rules are built on the level of individual neurons and then combined, (ii) *pedagogical* approaches using a black-box model of the underlying network and (iii) *eclectic* algorithms with aspects from both previous types. Tickle et al. (1998) also introduced a fourth intermediate category, *compositional*, to accommodate for RNN-RE algorithms that are all (except for one pedagogical algorithm (Vahed & Omlin, 1999, 2004)) based on analysing ensembles of neurons (i.e. the hidden state space).

4.2.3 Portability

Portability denotes how well an RE technique covers the set of available ANN architectures. As for translucency, portability is probably much the same for all RNN-RE algorithms. It is also a quite complex aspect of RE techniques (tightly bound with translucency, and, in terms of feasibility, with algorithmic complexity) and therefore this survey does not distinguish RNN-RE algorithms by this criterion.

4.2.4 Quality

The *quality* of the extracted rules is a very important aspect of RE techniques, and perhaps the most interesting for evaluation of the quality of the algorithms. This aspect differs from the other ones because it evaluates RE algorithms at the level of the *rules* rather than at the level of the RE algorithms themselves.

Based on previous work, such as Towell and Shavlik (1993), four sub-aspects of rule quality are suggested in the ADT taxonomy;

- *rule accuracy*, i.e. the ability of the rules to generalize correctly to unseen examples,
- *rule fidelity*, i.e. how well the rules mimic the behaviour of the RNN,
- *rule consistency*, i.e. the extent to which equivalent rules are extracted from different networks trained on the same task, and
- *rule comprehensibility*, i.e. readability of rules and/or the size of the rule set.

4.2.5 Algorithmic complexity

The *algorithmic complexity* of RE algorithms is unfortunately also often an open question as authors seldom analyse this explicitly (Andrews et al., 1995). Although Golea (1996) demonstrated that RE can be an NP-hard problem, it is unclear how existing heuristics affect the actual *expected* time and space requirements. The complexity of RNN-RE has not received much attention and the issue itself is quite complex as the execution time can be affected by many factors, e.g., number of state nodes, number of input symbols, granularity of the quantization, RNN dynamics etc.

Chapter 5

RNN-RE Techniques

In spite of the identified common characteristics of RE algorithms (Table 3.1), dividing them into groups has been a painstaking task as there are innumerable ways to do so. The techniques are presented in a primarily chronological order and when a later technique is similar to an earlier one, it is presented in connection with its predecessor (although this relation may be constituted by coincidental similarities rather than a direct continuation of prior work).

Firstly, some early work that laid the foundation for the development of RE techniques is presented in the following section. The algorithms are subsequently described in more detail in Sections 5.2-5.7. However, for more comprehensive descriptions of the algorithms, interested readers should refer to the original papers.

5.1 Pre-RE approaches

To understand the roots of FSM extraction from recurrent networks, it is useful to recognize that in some early attempts to analyse RNNs, clustering techniques were used on the state space, and clusters corresponding to the states of the FSM generating the language were found (clustering, i.e. quantization, is still today one of the central issues of the research on RE from RNNs). Hierarchical Cluster Analysis (HCA) was used for analysing RNNs in a few early papers on RNNs (Cleeremans et al., 1989; Servan-Schreiber et al., 1989; Elman, 1990; Servan-Schreiber et al., 1991). The authors found that for a network trained on strings generated by a small finite-state machine, the HCA may find clusters in the state space which apparently correspond to the states of the grammar. The clusters of the HCA were labelled using the labels of the states of the underlying (known) state machine, making it easy to draw the connection between the RNN and the FSM.

The fact that much of the early research on RNNs was conducted on problem sets *explicitly based* on FSMs may have biased subsequent research to look for these FSMs inside the network. However, for some successful networks (e.g. Servan-Schreiber et al., 1991), no clusters corresponding directly to the states of the FSM, which generated the training set language, were found. This meant the network had an alternative, but apparently correct, representation of the problem, that differed from the one anticipated. This was probably due to the fact that the clusters of the internal state of the network did not necessarily have a straightforward one-to-one relation with the states of the corresponding *minimal* machine. It was later shown that non-minimal machines would typically be what is initially extracted by clustering the state space when RE was used on RNNs (Giles, Miller, Chen, Chen & Sun, 1992). Therefore FSM minimization is included in most RNN-RE algorithms.

The basic problem of using only clustering (and not recording the transitions) for analysing RNNs is that there is no reliable way of telling how the clusters relate to each other temporally[1]. If the exact same FSM is not found, the clusters may not be labelled using the original FSM as a source and the temporal ordering of the clusters is therefore lost. This problem was also observed by Elman (1990): "the temporal relationship between states is lost. One would like to know what the trajectories between states [...] look like.". The solution of this problem led to the development of FSM extraction from RNNs.

5.2 Search in equipartitioned state space

The algorithm of Giles et al. (Giles et al., 1991; Giles, Miller, Chen, Chen & Sun, 1992; Omlin & Giles, 1996b) partitioned the state space into equally sized hypercubes (i.e. macrostates) and conducted a breadth-first search by feeding the

[1] There are also other problems of an HCA-based analysis of ANNs in general, as adjacent states (i.e. hidden unit activations) may be interpreted differently by the output layer and remote states may have the same interpretation (N. E. Sharkey & Jackson, 1995). For RNNs this becomes even more problematic as the state is not only mapped into an output but also mapped recursively to all succeeding outputs through the state transitions. This is one of the issues that is dealt with in more detail in Part II.

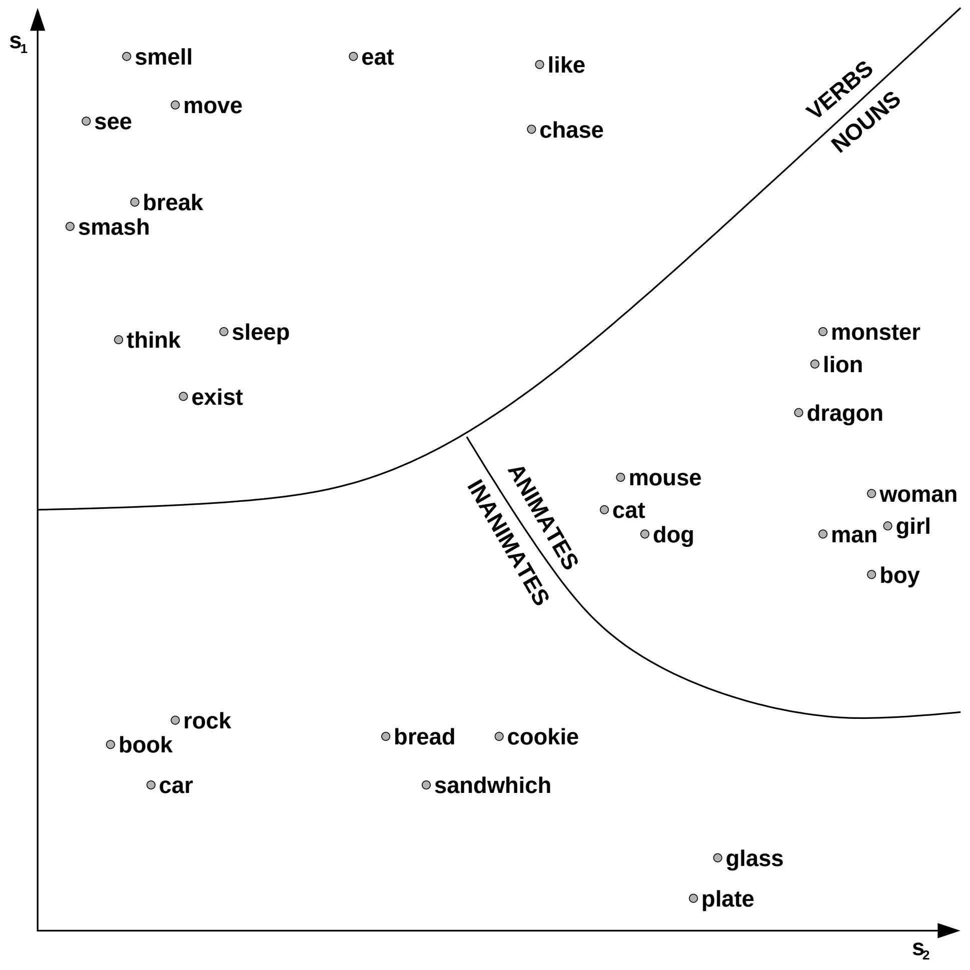


Figure 5.1: A schematic conceptual diagram of spatial representations in the state nodes s_1 and s_2 of an SRN as presented by Elman (1990) (Elman did, however, use an HCA rather than plotting values directly since his SRN had 150 state nodes). Elman's SRN was trained on predicting words in natural language sequences, and it separated the internal representations of words through their context in sentences. Word classes as well as semantical grouping were observed. By observing only recorded activations of the state space, however, there is no information how the temporal relationships of the words are processed dynamically by the SRN. These dynamics can be analysed using RNN-RE techniques, however.

DFA extraction, regular partitioning, breadth first search (Giles et al., 1991; Giles, Miller, Chen, Chen & Sun, 1992; Omlin & Giles, 1996b)	
Rule type:	Moore DFA with binary (accept/reject) output.
Quantization:	Regular partitioning by q intervals in each state dimension, generating q^N bins of which typically only a small subset is visited by the RNN.
State generation:	Breadth-first search.
Network(s):	Predominantly used on second-order RNNs
Domain(s):	Predominantly regular languages with relatively few symbols. Some applied domains, e.g., quantized financial data (Giles, Lawrence & Tsoi, 1997; Lawrence, Giles & Tsoi, 1998; Giles, Lawrence & Tsoi, 2001)

Table 5.1: Summary of algorithm extracting DFA through searching in an equipartitioned state space.

network input patterns until no new partitions were visited. The transitions among the macrostates (induced by input patterns) were the basis for the extracted machine. The search started with a predefined initial state of the network and tested all possible input patterns on this microstate, see Figure 5.2. The first encountered microstate of each macrostate was then used to induce new states. This guaranteed the extraction of a deterministic machine since any state *drift* (Das & Mozer, 1994, 1998) was avoided as the search was pruned when reentering already visited partitions. The extracted automaton was then minimized using a standard minimization algorithm for DFA (Hopcroft & Ullman, 1979). The algorithm is summarized in Table 5.1.

The central parameter of the algorithm is the quantization degree q of the equipartition. The authors suggested starting with $q = 2$ and increasing it until an automata consistent with the training set is extracted, i.e. the termination criteria is to have perfect accuracy of the rules. The choice of q is, however, usually not explicitly described as part of the RE algorithm (one exception is in the description by Omlin (2001) where the suggested incremental procedure is also part of the algorithm).

Giles, Miller, Chen, Chen and Sun (1992) found that the generalization ability of the extracted machines sometimes exceeded that of the underlying RNNs. Since the networks were trained on regular grammars, if the extraction result was a DFA equivalent with the original grammar that generated the training/test set, generalization would also be perfect. Giles, Miller, Chen, Sun et al. (1992) showed

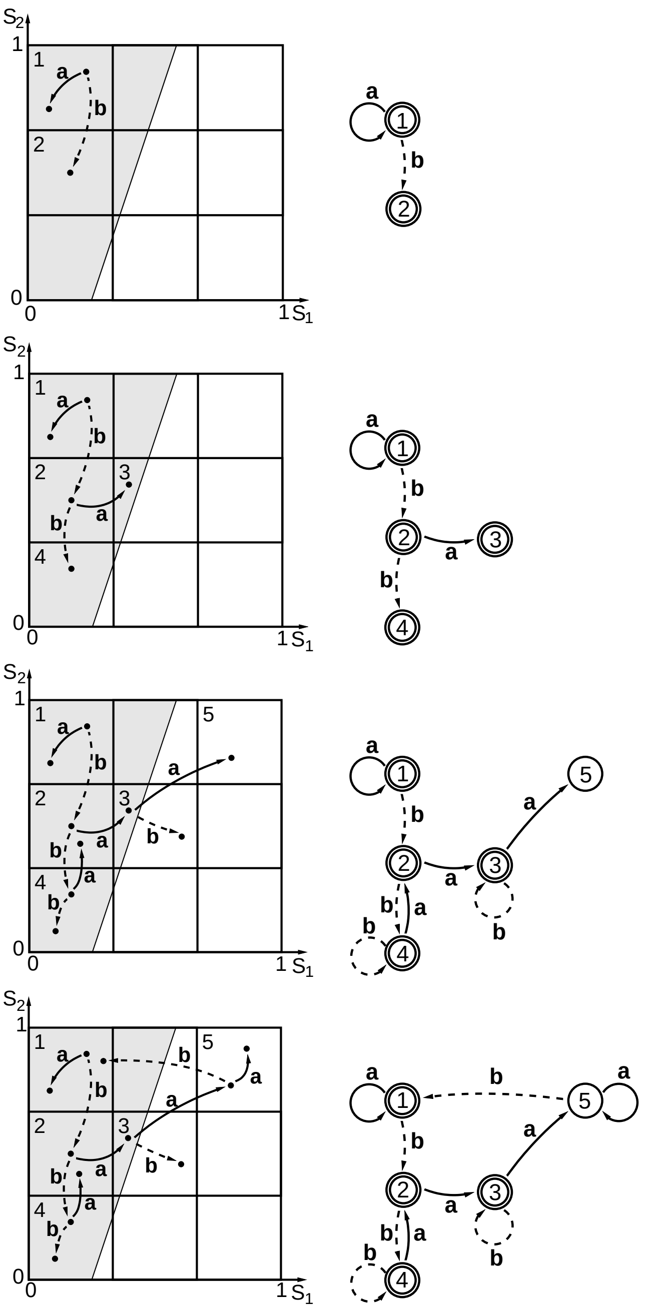


Figure 5.2: An example of the DFA extraction algorithm of Giles et al. (1991) used on an RNN with two state nodes trained on a binary language and the quantization parameter $q = 3$. The state space is divided into accept and reject regions (gray and white respectively). The algorithm expands the graph until all nodes have two outgoing arcs. Note that the macrostate corresponding to node 3 could actually be interpreted both as an accept *and* reject state depending on the microstate, but the algorithm used the interpretation of the first encountered microstate as the interpretation of the macrostate.

that during successful training of an RNN, the extracted DFA will eventually belong to the same *equivalence class* as the original DFA. The existence of equivalence classes over different degrees of quantization (i.e. different values of q) was used in Omlin, Giles and Miller (1992) as an indicator of the networks' generalization ability, i.e. if the extracted DFAs for increasing values of q collapsed into a single equivalence class, it was taken as a sign of good generalization ability without the need for explicitly testing this on a separate test set.

The same algorithm has been used in various other contexts: as part of rule refinement techniques (e.g. Omlin & Giles, 1992; Giles & Omlin, 1993; Das et al., 1993; Omlin & Giles, 1996c), as an indicator of an underlying language class (Blair & Pollack, 1997), as a method for complexity evaluation (e.g. Bakker & Jong, 2000), as part of a quantitative comparison of different RNN architectures (Miller & Giles, 1993), as a means for FSM acquisition[2] (e.g. Giles, Horne & Lin, 1995) or simply as an analysis tool of the RNN solutions[3] (e.g. Giles & Omlin, 1994; Goudreau & Giles, 1995; Giles et al., 1997; Lawrence et al., 1998; Lawrence, Giles & Fong, 2000; Giles et al., 2001; Bakker, 2004). The algorithm has also been used in the context of *recursive* networks (Maggini, 1998).

An apparent problem with this technique is that the worst-case number of clusters grows exponentially with the number of state nodes N (q^N). The time needed for the breadth-first search will also grow exponentially with the number of possible input symbols. In practice, however, the number of visited states is much smaller than the number of possible states.

This, the earliest of RNN-RE methods, is also the most widespread algorithm. Almost all subsequent papers where new RNN-RE techniques have been proposed cite Giles, Miller, Chen, Chen and Sun (1992). But often these papers do not contain citations to each other, implying that the field is less diverse than it actually is. Consequently there is a surprising variety of RE approaches, some of them seemingly developed independently of each other.

[2]Implicitly, however, more or less *all* papers using RE are in some way on FSM/language acquisition. This division into RNN-RE usage should be taken with a grain of salt since each paper has more than one contribution.

[3]This is also implicitly part of many other papers as well.

DFA extraction, vector quantifier, breadth first search (Zeng et al., 1993; Frasconi et al., 1996; Gori et al., 1998)	
Rule type:	Moore DFA with binary (accept/reject) output.
Quantization:	k-means.
State generation:	Breadth-first search.
Network(s):	Second-order RNNs (Zeng et al., 1993), Recurrent radial basis function network, (Frasconi et al., 1996; Gori et al., 1998), RNN with an external pushdown automaton (G. Z. Sun, Giles & Chen, 1998).
Domain(s):	Regular binary languages (Tomita, 1982), context free languages (G. Z. Sun et al., 1998).

Table 5.2: Summary of algorithms extracting DFA through searching in a state space partitioned by vector quantization.

5.3 Search in state space partitioned through vector quantization

An alternative to the simple equipartition quantization was already suggested by Zeng et al. (1993) where a k-means algorithm was used to cluster the microstates. The centres of the clusters, the *model vectors*, were used as the basis for the breadth-first search, i.e. the RNN was tested with all input symbols for each model vector state (cf. the equipartition algorithms where the first encountered RNN state is the basis for further search). See Figure 5.3 for an illustrative example of this algorithm. A similar approach, also using k-means, developed seemingly independently from Zeng et al. (1993) was presented in Frasconi, Gori, Maggini and Soda (1996) and Gori, Maggini, Martinelli and Soda (1998), and a similar SOM-based approach in Blanco et al. (2000). A summary of these approaches is provided in Table 5.2.

In order to support an appropriate clustering of states, Zeng et al. (1993) and Frasconi et al. (1996) induced a bias for the RNN to form clusters during training. Other studies have also followed this approach (Das & Das, 1991; Das & Mozer, 1994, 1998). RE-RNN algorithms developed on such specialized RNNs may, however, not work on other networks. RE techniques that can be used on already existing networks (i.e. typically not designed for easy analysis) are described by Tickle et al. (1998) as more attractive techniques.

In the presented search-based approaches, the reentering into partitions was the basis of pruning the search. A different pruning strategy was suggested by Alquézar

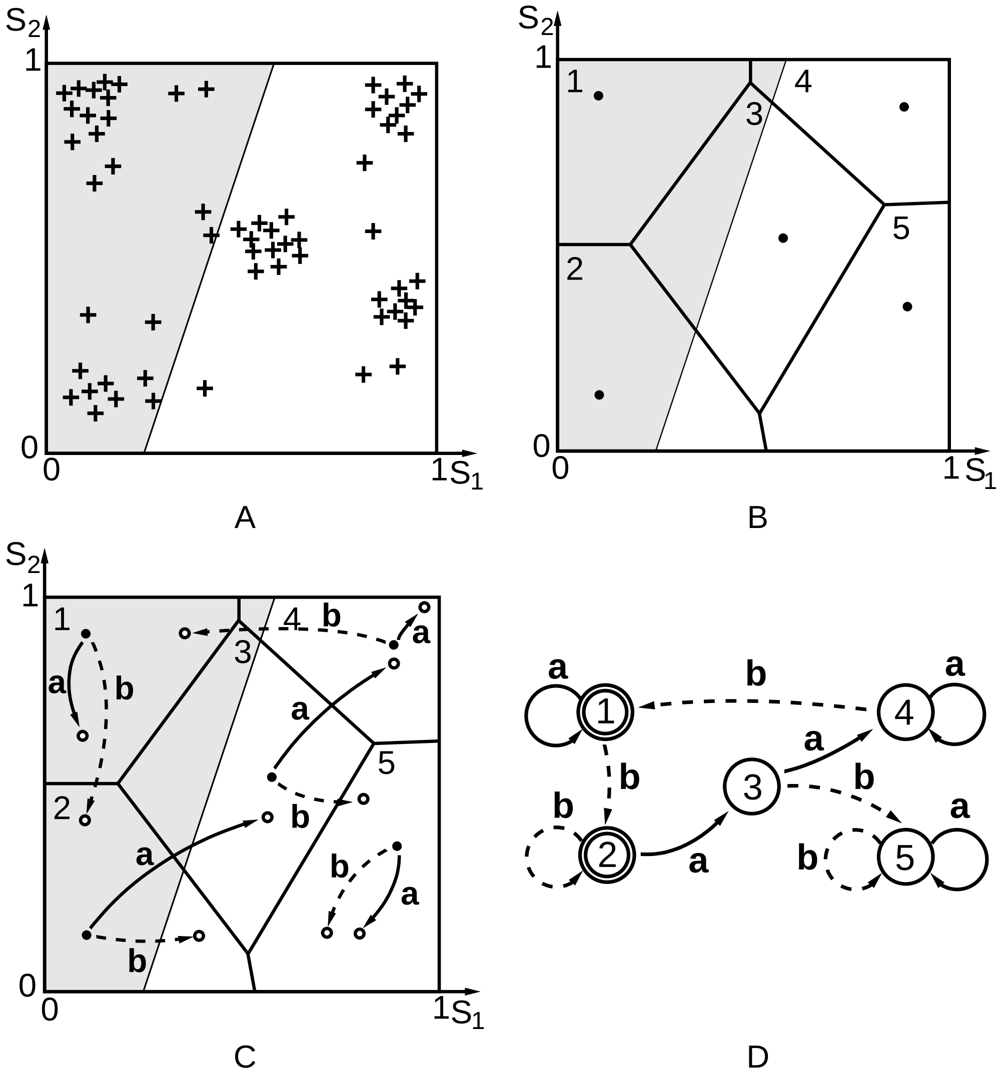


Figure 5.3: An illustrative example of rule extraction through breadth-first search in a state space clustered by k-means. (A) The states of the RNN are sampled during training, (B) these states are clustered into a predefined number of clusters, (C) a breadth-first search (cf. Figure 5.2)conducted based on the model vectors and, (D) the machine is constructed.

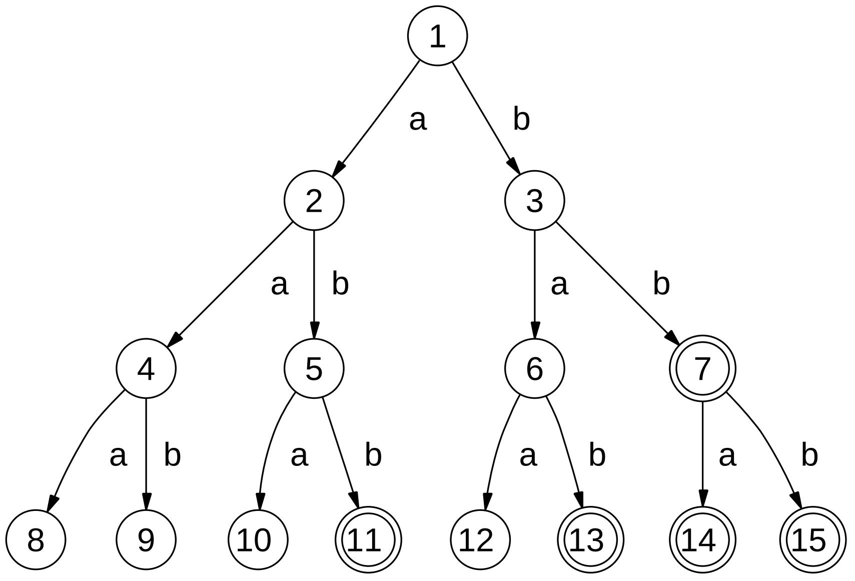


Figure 5.4: An example of a prefix tree of depth three, created from a language that only accepts strings containing at least two `b`'s.

and Sanfeliu (1994a) and Sanfeliu and Alquézar (1995) who chose to use the domain to determine search depth (the algorithm is summarized in Table 5.3). A *prefix tree* (see Figure 5.4) was built based on the occurrences of positive and negative strings in the training set, i.e. the prefix tree contained only strings present in the training set. The states of the RNN were generated using only the strings in the prefix tree. The authors used RE as part of their Active Grammatical Inference (AGI) learning methodology, an iterative rule refinement technique.

The states generated with the prefix tree were the basis of the initial machine. The spatially closest pair of these states was then merged iteratively until further clustering would result in an inconsistency. This RE technique was also used for a wide variety of regular grammars and two types of networks in Alquézar et al. (1997). The authors reported that the extracted machines on average performed significantly better than the original RNNs.

5.4 Sampling-based extraction of DFA

Instead of conducting a search in the quantized state space, the activity of the RNN in interaction with the data/environment can be recorded. In this way, the domain can be considered as heuristics confining the states of the RNN to only relevant states.

DFA extraction, hierarchical clustering, sampling on domain (Alquézar & Sanfeliu, 1994a), and (Sanfeliu & Alquézar, 1995)	
Rule type:	Unbiased Moore DFA. Unbiased means the output is trinary (accept, reject and unknown).
Quantization:	Hierarchical clustering.
State generation:	A prefix-tree is built based on the examples of the training set.
Network(s):	First-order RNN (not specified in Alquézar and Sanfeliu (1994a) but in Sanfeliu and Alquézar (1995)).
Domain(s):	At least 15 different regular binary languages (Alquézar et al., 1997).

Table 5.3: A summary of the search-based DFA extracting algorithm proposed by Alquezar and Sanfeliu for unbiased grammars.

DFA extraction, dynamic interval clustering, sampling on domain (Watrous & Kuhn, 1992)	
Rule type:	Moore DFA with binary (accept/reject) decision.
Quantization:	Dynamically updated intervals for each state unit. States are collapsed and split through updating the intervals.
State generation:	Sampling the RNN while processing the domain.
Network(s):	Second-order RNNs.
Domain(s):	Regular binary languages (Tomita, 1982).

Table 5.4: A summary of the sampling-based DFA extraction algorithm proposed by Watrous and Kuhn (1992).

Already before the development of RE techniques for RNNs, sampling of the state space using the domain, was the most natural way of conducting analysis of RNNs (Cleeremans et al., 1989; Servan-Schreiber et al., 1989; Elman, 1990). The first RE technique based on sampling the RNN was proposed by Watrous and Kuhn (1992) (see Table 5.4). The quantization of the state space was based on splitting individual state units' activations into intervals. They described that these intervals could be merged and split to help the extraction of minimal and deterministic rules. The procedure of state splitting, however, is somewhat vaguely described and may require intervention from the user.

Manolios and Fanelli (1994) chose to use a simple vector quantifier to discretize the state space. Training from different, randomly initiated, model vectors were repeatedly conducted until a deterministic machine was found. The termination of this procedure is, however, not guaranteed. The algorithm is summarized in Table 5.5.

DFA extraction, vector quantifier, sampling on domain (Manolios & Fanelli, 1994), originally in tech. rep. (Fanelli, 1993)	
Rule type:	Moore DFA with binary (accept/reject) decision.
Quantization:	A simple vector quantifier, details unclear.
State generation:	Sampling on a test set.
Network(s):	First-order RNNs.
Domain(s):	Regular binary languages (Tomita, 1982).

Table 5.5: The sampling-based DFA extractor originally proposed in Fanelli (1993).

DFM extraction, SOM, sampling on domain (Tiňo & Šajda, 1995)	
Rule type:	Mealy DFM with multiple output symbols.
Quantization:	Star topology SOM.
State generation:	Sampling on training set.
Network(s):	Second-order RNNs.
Domain(s):	Regular formal language domains with either two or three input symbols (not counting the end-of-string symbol) and two or three output symbols.

Table 5.6: Summary of the sampling-based DFM extractor of Tiňo and Šajda (1995).

A similar approach was suggested in Tiňo and Šajda (1995) where an algorithm for removing inconsistent transitions was introduced. This algorithm could, however, fail under certain circumstances so that the extraction of a DFA could not be guaranteed. A star topology self-organizing map (SOM, (Kohonen, 1995)) was used to quantize the state space. Tiňo and Šajda (1995) were the first to extract Mealy instead of Moore machines and also the first who did not confine the output to binary accept/reject decisions (not counting the unbiased DFA of Alquézar and Sanfeliu (1994a)). This algorithm is summarized in Table 5.6.

The breadth-first search will reliably find consistent DFMs since the search is pruned before inconsistencies leading to indeterminism are introduced. The DFM will also be complete since all symbols are tested on all states. In sampling the state space, determinism is no longer guaranteed, since two microstates of the same macrostate may result in transitions to different macrostates (even though these transitions are triggered by the same input symbol). Two state vectors in the same partition may also be mapped to different classes in the output. The extracted machines may also be *incomplete* since all symbols may not have been tested on

DFM extraction, vector quantizer, sampling on domain (Schellhammer et al., 1998)	
Rule type:	Mealy DFM with a "rescue state" used to make machine complete.
Quantization:	k-means.
State generation:	Sampling on training set. Inconsistencies solved by discarding the least frequent of inconsistent transitions.
Network(s):	SRN.
Domain(s):	Natural language prediction task.

Table 5.7: Summary of the only sampling-based DFM extractor, where inconsistencies and incompleteness are handled.

all states. Therefore, the DFM extraction, through sampling could fail as in the above cases of Watrous and Kuhn (1992), Manolios and Fanelli (1994) and Tiňo and Šajda (1995). It is unclear how incomplete machines were handled in the above described approaches. Perhaps the extracted machines were small enough and the domains simple enough that no such problems occurred.

One approach to solving the problem of indeterminism is the use of transition frequencies to discard the least frequent of inconsistent transitions. This heuristic should, in most cases, solve the inconsistency without deviating much from the operation of the underlying RNN in the majority of the transitions. This simple procedure was proposed by Schellhammer et al. (1998) (summarized in Table 5.7). They also dealt with the problem of incomplete machines by creating transitions to a predefined "rescue state" to complete the machine. These simplifications did not significantly reduce the performance of the DFM and the rescue state enabled the machines to make "guesses" about inputs that otherwise would not be possible to parse.

5.5 Stochastic machine extraction

As described in the previous section, the extraction of *deterministic* FSMs (DFMs) from RNNs through sampling is hampered by the fact that the quantization of the state space may lead to inconsistencies in the macrostate transitions. These inconsistent transitions (and potentially state interpretations) will, however, follow some patterns and if all such transitions are counted they can be transcribed into a *stochastic machine*, i.e. a machine with probabilities associated with the transitions. The inconsistencies that ruin a DFM extraction may in other words contain informative probabilities that more accurately describe the RNN.

An algorithm for the extraction of stochastic machines from RNNs was proposed by Tiňo and Vojtek (1998). These extracted machines were, however, not equivalent to the stochastic machines defined by Paz (1971) and Rabin (1963) since the the conditional probability of the output given the state transition was not included in the model, i.e. the extracted machines did not model the output of the RNN. The algorithm quantized the state space using a SOM (as did Tiňo and Šajda (1995)). The generation of states (and state transitions) was divided into two phases; the "pre-test" phase, where the RNN was domain-driven, and a "self-driven" phase, where the output of the RNN was used as input in the next time-step (this RNN was trained to predict a long sequence of symbols). In Tiňo and Köteles (1999) (further described in Tiňo, Dorffner and Schittenkopf (2000)) the SOM was replaced with a dynamic cell structure (DCS, Bruske and Sommer (1995)), but otherwise the algorithm was the same (see the summary in Table 5.8).

The stochastic machines can be analysed in new interesting ways. The authors (Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999), for example, used *entropy spectra* (K. Young & Crutchfield, 1993) to compare the probabilities of strings generated by the RNNs with the probabilities of the strings in the original source. While the results were interesting, there were no indications, in that paper, how well the extracted machines corresponded to the network (i.e. rule fidelity) or how well they generalized on any test set[4] (i.e. rule accuracy). The comprehensibility of the extracted rules complete also cannot be determined from these papers. The fact that the extracted machines did not model the output of the RNN also makes it difficult to evaluate this algorithm in the context of the other ones

Another related approach which may not be rule extraction *per se*, but can perhaps, at least, be termed a *partial* rule extraction algorithm, is the "neural prediction machine" (NPM) constructed in (Tiňo, Čerňanský & Beňušková, 2004). The NPM predicts the next symbol given the state of the network, i.e. the state dynamics are handled by the RNN and not extracted at all (see a summary of this

[4]Unless the entropy spectra analysis is considered a form of accuracy measurement. This is something it can be argued to be (P. Tiňo, personal communication, June 27, 2006)

Stochastic machine extraction, SOM, sampling on domain (Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999)	
Rule type:	Stochastic Mealy finite state machine.
Quantization:	SOM (unspecified topology) in Tiňo and Vojtek (1998) and DCS in Tiňo and Köteles (1999)
State generation:	Two phases: Sampling on training set and "self-driven" RNN.
Network(s):	Primarily second-order RNNs.
Domain(s):	Prediction of (four) symbols generated from continuous chaotic laser data and a chaotic series of binary symbols generated with iterated logistic map function.

Table 5.8: Summary of approaches of RNN-RE for extraction of stochastic machines.

Neural prediction machine, vector quantizer, sampling on domain (Tiňo et al., 2004)	
Rule type:	A "Neural Prediction Machine" (NPM) predicting the next output based on current state of the RNN. State transitions not modelled.
Quantization:	k-means.
State generation:	Sampling.
Network(s):	First-order RNN.
Domain(s):	Continuous chaotic laser data domain transformed to four symbols and recursive natural language domains.

Table 5.9: Neural Prediction Machines (NPMs) differ from the FSM ordinarily extracted from RNNs in that state transitions are not incorporated into the model.

approach in Table 5.9).

5.6 A pedagogical approach

All previously described algorithms comply with the category *compositional* in ADT's translucency classification (see Section 4.2). There is, to my knowledge, only one algorithm that uses a *pedagogical* approach instead. Vahed and Omlin (1999, 2004) used a machine learning method requiring only the input and the output to extract the machine, i.e. the internal state is ignored (see the summary in Table 5.10). The data used for extraction was based on all strings up to a given length. The input and output of the network was recorded and fed to the polynomial-time "Trakhtenbrot-Barzdin" algorithm (Trakhtenbrot & Barzdin, 1973).

It was also reported that this algorithm was more successful in returning correct

DFA extraction, black-box model (Vahed & Omlin, 1999, 2004)	
Rule type:	Moore DFA with binary (accept/reject) output.
Quantization:	N/A.
State generation:	All strings up to a certain length.
Network(s):	Second-order RNN.
Domain(s):	One randomly generated 10-state DFA.

Table 5.10: The only RNN-RE algorithm where the internal state of the RNN is not regarded during the extraction process.

DFAs than clustering-based algorithms (Giles, Miller, Chen, Sun et al., 1992). This paper actually seems to be the *only* one that describes an experimental comparison of different RE techniques at all.

The machine learning algorithm they used is indeed of polynomial time complexity, given that a prefix tree (see Figure 5.4) is available. But the size of the prefix tree up to a string length L is of complexity $O(n^L)$, where n is the number of symbols. As a consequence, this approach is likely to have some problems scaling up to more complex problems with more symbols.

5.7 RE-supporting RNN architectures

As previously mentioned, clusters can be induced during training to support RE in later stages (Zeng et al., 1993; Frasconi et al., 1996). This was originally suggested in Das and Das (1991) and further developed in Das and Mozer (1994) and Das and Mozer (1998). Training to induce clusters results, if successfully performed, in RNNs that are *trivially* transformed to finite machines. Since the focus of this survey is on the details of the *extraction procedure*, more particulars about these approaches are not be included in this thesis.

Chapter 6

Discussion

This section summarizes and evaluates the described techniques from the perspectives of the evaluation criteria: rule type, quantization method, state generation, network type and domain. The two criteria portability and quality of the ADT taxonomy (described in Section 4.2), are also discussed.

6.1 Rule types

It is quite clear that most of the research described in Chapter 5 focuses on extracting "traditional" DFA for classification of binary strings as grammatical/ungrammatical (Giles et al., 1991; Giles, Miller, Chen, Chen & Sun, 1992; Watrous & Kuhn, 1992; Zeng et al., 1993; Alquézar & Sanfeliu, 1994a; Manolios & Fanelli, 1994; Sanfeliu & Alquézar, 1995; Omlin & Giles, 1996b; Frasconi et al., 1996; Gori et al., 1998; Vahed & Omlin, 1999, 2004). Only a few DFA extraction algorithms are used on domains with more than two output symbols (Tiňo & Šajda, 1995; Schellhammer et al., 1998). It is also interesting that only three papers (Schellhammer et al., 1998; Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999) have studied DFA RNN-RE in a prediction domain while prediction of sequences is quite commonly studied in RNN research in general (e.g. Elman, 1990; Alquézar & Sanfeliu, 1994b; Jacobsson, 1999; Gers & Schmidhuber, 2001; Jacobsson & Ziemke, 2003a).

The crisp DFA do not model probabilistic properties of macrostate transitions and macrostate interpretations; that kind of information is lost in the rules, independently of whether search or sampling is used to generate states. Hence, a more expressive set of rules may be represented in stochastic FSM (Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999). Furthermore, the fidelity, i.e. the coherence of the rules with the RNN, of stochastic rules should in principle be higher (given the same premises, e.g., quantization) than for their deterministic counterparts. The fidelity can, however, be measured in various ways (since the term is not clearly defined) and may possibly lead to ambiguous results. If stochastic machines are to be used for RNN analysis, however, it is important that the extracted machines also model the output of the RNN. Such stochastic machines (Paz, 1971; Rabin, 1963) have yet to be extracted and further work is required to realize this (which this thesis does actually in Part II).

A way of combining "the best of both worlds" may be to advance the method chosen by chosen by Schellhammer et al. (1998) in which probabilities were calculated and then used as heuristics for transforming the incomplete and nondeterministic machine into a deterministic and complete machine. Thus the information loss from transforming the RNN to a deterministic machine could possibly be tracked. Of course, this will depend on if the RNN robustly emulates a FSM (Casey, 1996). Whether this would work or not remains an open issue, but the suggested algorithm in Part II may alleviate some of the problem by extracting stochastic machines until a deterministic machine possibly will be found (which is not guaranteed since the RNN may for example be chaotic).

A last, "exotic", form of rules is the Neural Prediction Machine. The NPM only predicts the output of the network given the state, and is not concerned with the internal mappings of states in the RNN (Tiňo et al., 2004).

6.2 State space quantization

Clearly, there is no consensus about how to quantize the state space. Methods that have been used are (see Chapter 5 for more complete reference lists):

- Regular (grid) partition (Giles et al., 1991),
- k-means (Zeng et al., 1993; Frasconi et al., 1996; Schellhammer et al., 1998; Tiňo et al., 2004; Cechin, Pechmann Simon & Stertz, 2003),

- SOM (Tiňo & Šajda, 1995; Tiňo & Vojtek, 1998; Blanco et al., 2000), dynamical cell structures (Tiňo & Köteles, 1999),
- "other" vector quantifiers (Manolios & Fanelli, 1994),
- hierarchical clustering (Alquézar & Sanfeliu, 1994a),
- dynamically updated intervals (Watrous & Kuhn, 1992) and,
- fuzzy clustering (Cechin et al., 2003).

This totals eight different techniques, not counting small variations in implementations. Although only a fraction of the existing clustering techniques have been tested at all (Mirkin, 1996; Jain, Murty & Flynn, 1999) it is clear that many of them has been used to approach the quantization problem.

However, the most striking aspect about the use of this multitude of various techniques is not that there are so many, but that there are *no studies* comparing different quantization techniques to each other in the context of RNN-RE.

The two main families of clustering techniques used are vector-quantization (VQ, e.g., k-means and SOM) and equipartition. The main difference between these, apart from VQ-partitions not being of equal sizes and shapes, is that the VQ-clusters are not fixed *prior* to the extraction but are instead *adapted* to fit the actually occurring state activations in the RNN. In principle, vector quantization should be able to scale up to more state nodes than the equipartition method since the number of partitions can be arbitrarily selected independent from state space dimensionality.

6.3 State generation

There are two basic strategies for generating the states in the RNN (see Chapter 5 for more complete reference lists):

- *Searching* (Giles et al., 1991; Zeng et al., 1993; Frasconi et al., 1996) and,
- *sampling* (Watrous & Kuhn, 1992; Manolios & Fanelli, 1994; Alquézar & Sanfeliu, 1994a; Tiňo & Šajda, 1995; Schellhammer et al., 1998; Tiňo & Vojtek, 1998; Tiňo et al., 2004).

There are almost no studies experimentally comparing searching- and sampling-based RNN-RE, apart from one preliminary study (Jacobsson & Ziemke, 2003b) (cf. Appendix D). In contrast to the situation with clustering techniques however, it is quite easy to see at least a few of the consequences of the choice of state generation method.

Firstly, breadth-first search will obviously have problems with scaling up to larger problems and is especially sensitive to the number of input symbols. There are also reasons to believe that for prediction networks in domains that are not completely random, many of the transitions and states generated with breadth-first search would not be relevant or ever occur in the domain (Jacobsson & Ziemke, 2003b). Machines extracted with search are, however, guaranteed to be deterministic, which may very well be desired (see discussion in the previous section). The extraction is also guaranteed to result in a *complete* machine where all possible inputs are tested on all encountered states.

RE through sampling on the domain is not guaranteed to result in deterministic and complete machines. If this is required, there is no guarantee that a certain state space quantization will result in a solution since inconsistencies might occur. A heuristic solution to this problem has only been proposed in one paper (Schellhammer et al., 1998). In summary, sampling-based RE techniques may be a preferable strategy for extraction of stochastic rather than deterministic machines.

6.4 Network types and domains

The networks that have been studied using RNN-RE are in most cases relatively small ones with few state nodes. This may be due to the fact that most domains used were simple enough to allow small networks to be trained.

There are also significantly more second-order than first-order networks. This is probably an effect of the focus on formal language domains where second-order RNNs are more commonly used than first-order networks (Goudreau, Giles, Chakradhar & Cheng 1994).

As mentioned in Section 6.1, the investigated domains mostly require only binary string classification. More complex domains with many symbols, deep syntactical structures or chaotic behaviour, etc., have not been tested using RNN-RE (before this thesis, cf. Chapter 12 of Part II). Therefore, the applicability of these techniques is largely an open issue. The importance of rule extraction as described by Andrews et al. (1995), e.g., explanation capability, verification of ANN components, etc., is therefore less than it would have been if the techniques had been demonstrated to work on the state-of-the-art RNNs operating on the most challenging domains.

6.5 Portability

Even though most RNN-RE algorithms are compositional, (i.e. under the ADT translucency criteria) and have, in principle, the same requirements on the underlying RNN, there are some *implicit* requirements that could be useful to identify, especially if the existing RNN-RE algorithms are to be applied on previously unencountered RNN architectures (or other dynamic systems) and domains. Current RE techniques are preferably used on RNNs that:

1. operate in discrete time since continuous-time RNN can not be described as finite state machines. There is, however, no known study on continuous time RNNs in the domain of FSM generated languages (Forcada & Carrasco, 2001),
2. have clearly defined input, state and output nodes, i.e. randomly structured RNNs may be problematic,
3. have a fully observable state, otherwise unobserved state nodes or noise in the observation process would disturb the extraction process since the state space would not be reliably quantized,
4. have state nodes that can be set explicitly (for search-based techniques),
5. are deterministic, otherwise the same problem would occur as when the state is not fully observable,
6. are fixed during RE, i.e. no training of the RNN can be allowed during the RE process,
7. operate on a preferably discrete domain (or be transformed to a discrete representation prior to RE (e.g. Giles et al., 1997)) since there are no means of representing continuous input/output data in the current types of extracted rules since the transitions of the extracted FSMs must be labeled using input

and output symbols.

The problem with making a list of implicit requirements is just that they are *implicit* (i.e. not readily apparent). Hence, there may be other essential requirements that I have not managed to consider at this stage. Furthermore, the strengths of these requirements are unclear as well, some of them may actually be quite easily alleviated with some enhancements of current RNN-RE techniques (this possibility will be discussed further in Section 17.9).

6.6 Rule quality

As previously mentioned (cf. Section 4.2.4), the quality of RNN-RE techniques is (or should be) evaluated at the level of the actual rules, rather than at the level of the algorithms. Extracted rules depend not only on the algorithm but also on the underlying domain and network. Evaluation of the rule quality therefore requires extensive studies comparing different RE techniques under similar conditions. Unfortunately, such studies have not yet been conducted for most RNN-RE algorithms.

There are, in the existing corpus of papers on RNN-RE, a few *indirect* results that provide some indications for some of the rule quality sub-categories: accuracy, fidelity, consistency and comprehensibility.

A number of studies indicate that the extracted machines indeed have high *accuracy* since they may even be generalizing better than the underlying RNN (Giles, Miller, Chen, Chen & Sun, 1992, 1992; Giles & Omlin, 1993; Omlin & Giles, 1996b). There are, however, unfortunately no studies in which the *fidelity* of the extracted rules has been tested separately from the *accuracy*[1]. The studies tend to focus on networks that are quite successful in their domain and under such circumstances the difference between fidelity and accuracy is very small. For networks performing badly in their domain, high fidelity would, however, imply low accuracy since the errors of the network would then be replicated by the machine.

Rule consistency has not been extensively studied, although some papers touch the subject. Rules extracted from a network during training were found to fall

[1]There is an interesting discussion about the fidelity-accuracy dilemma in Zhou (2004).

under a sequence of equivalence classes during training (Giles, Miller, Chen, Sun et al., 1992). This can be seen as an example of consistency since the extracted rules after a certain period of training eventually stabilized in the same equivalence class, i.e. the set of quite similar networks at the later stage of the training resulted in equivalent rules. The consistency over different parameter settings (of the quantization parameter q in the equipartitioned RNN-RE algorithm) has also been proposed as an indicator of regularity in the underlying network (Blair & Pollack, 1997). These results on consistency are, however, more or less indirect.

Rule comprehensibility is typically considered an important issue to ensure further progress for RNN-RE research. After all, if the goal of extracting rules is to understand the underlying incomprehensible network, the rules should preferably be comprehensible themselves. The comprehensibility of extracted rules has not been directly evaluated. It is, however, clear that in some cases the RNN-RE-analysis has been informative in qualitative ways. I will, however, argue in Part III of this thesis that the incomprehensibility of the rules does not render them useless.

6.7 RNN-RE, fool's gold?

Kolen (1993) showed with some simple examples that some dynamic systems with real-valued state space (e.g., an RNN) cannot be described discretely without introducing peculiar results (cf. Kolen and Pollack (1995)). If you want to approximate the behaviour of a physical system with a real-valued state space as a discrete machine you will not only risk that the approximation might not be exact. A more profound effect of the approximation is that induced machines, from the same physical system, may belong to completely different classes of computational models, depending only on how the transformation from the real-valued space to a discrete approximation is conducted[2].

This critique strikes at the very heart of RNN-RE, since the quantization of the state space is a crucial element of these algorithms and RNN-RE was actually

[2]A potential flaw in Kolen's argument is, however, that he is not only considering discretization of the state space, but also the discretization of the time of a continuous time dynamic system (P. Tiňo, personal communication, June 27, 2006). The underlying problem is, however, there nevertheless; that discretizing a continuous space will obviously result in something different than the continuous space in question and that depending on the exact nature of the discretization, different results may be obtained.

termed “fool’s gold” by Kolen (1993). He pointed out that RNNs should be analysed as dynamic systems or more specifically iterated function systems (IFSs) rather than state machines.

Nevertheless, there are some replies to this critique. One simple approach is to avoid the problem by not modelling transitions at all (Tiňo et al., 2004), or not even quantizing the state space (Vahed & Omlin, 1999, 2004). Another response to Kolen’s critique is that extraction of a state machine from an RNN has been proven to work if the underlying RNN robustly model a finite state machine (Casey, 1996). However, this does not alleviate the fact that the language class for unknown RNNs cannot be reliably recognized. But at least there is a theoretical “guarantee” that *if* there is an FSM at “the bottom” of an RNN, it can always be extracted in principle.

Failure of rule extraction from an RNN could therefore be an indicator that the underlying RNN is not implementing a finite state machine. One first step in this direction has been proposed by Blair and Pollack (1997). They used unbounded growth of the macrostate set under increased resolution of the equipartition quantization method as an indicator of a nonregular underlying RNN.

If we limit ourselves to real world domains, RE will necessarily be operating on finite domains, making FSM interpretations theoretically possible at all times (although they may not be the *minimal* description of a domain-RNN interaction). In fact, since the focus of RNN-RE research is on FSM extraction, the question should not be whether a language class is misjudged by an RE algorithm or not (since extraction at the level of the class of regular languages is one of the premises), but rather how well the extracted finite machine approximates the network, as proposed by Blair and Pollack (1997). How to evaluate the fidelity of an FSM and whether this evaluation may distinguish between errors stemming from a poorly quantized state space or from a higher language class in the RNN/domain remains an open issue.

In summary, although Kolen’s critique is justified, there are still reasons for further research on RE from RNNs: a lack of sophisticated analysis tools that can handle the complexity of RNNs hampers RNN research. Although there are theoretical possibilities that RE may result in ambiguous answers about an RNN, this also holds for many other analysis techniques. For any analysis tool to be trustworthy, the disadvantages must be known and taken into account when examining the results. This is precisely what makes Kolen's observations valuable for RNN-RE usage; the exposure of some of the limitations of RNN-RE techniques.

Chapter 7

Open Issues and Summary

7.1 Goals of RNN-RE

It is clear that the development of the RNN-RE algorithms has been driven by different goals in the different papers. But what are the possible goals of RNN-RE? Conceivable answers could be (partly overlapping with Andrews et al. (1995) in the context of RE in general):

1. to acquire a generic model of the domain, i.e. the RNN is used merely as a tool in the acquisition process (data mining),
2. to provide an explanation of the RNN,
3. to allow verification/validation of the RNN with respect to some requirements (cf. software testing) and thus make new, potentially safety critical, domains possible for RNNs,
4. to improve on current RNN architectures by identifying errors.

The appropriate measure to evaluate the success (or rule quality) of a specific instance of an RNN-RE algorithm being applied on an RNN (and domain) depends highly on which of these (or other) goals are desired (see Figure 7.1). For the first goal, for example, the maximization of accuracy is the prime target. In many of the papers it is clear that the accuracy is the most important aspect of rule quality. Accuracy is a satisfactory means for evaluating rule quality as long as the goal for rule extraction is to find rules that are "as good as possible" in the domain. For the other goals the maximization of fidelity is likely to be more important. After all, if the network is tested with an accuracy-maximizing RE method, the result

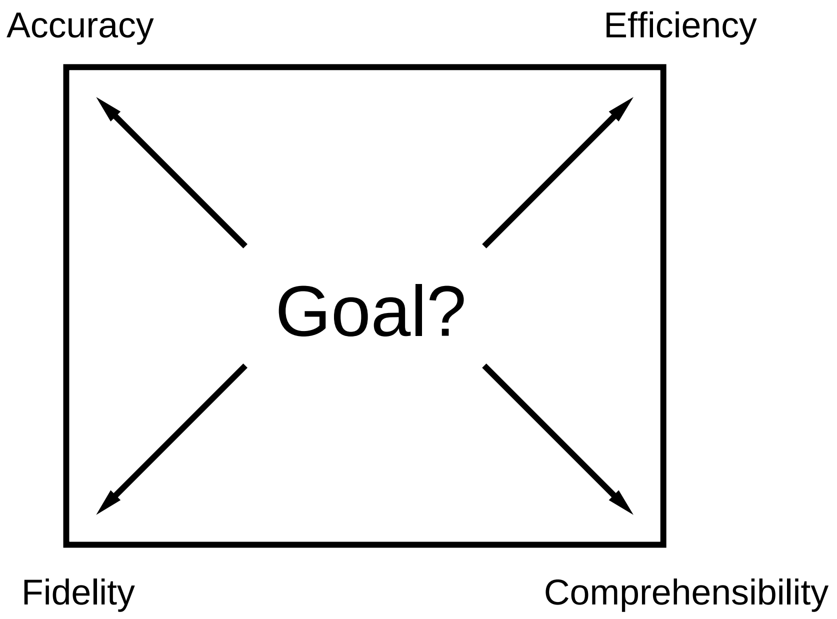


Figure 7.1: Four, possibly opposing, goals of RNN-RE that in an ideal algorithm would simply be chosen by the setting of a few user-defined parameters.

may be rules with a performance better than the network (a result confirmed by many studies). Therefore, for the purposes of RNN analysis, *fidelity* should be the preferred quality evaluation criterion. In some cases, however, comprehensibility may be more crucial than fidelity and accuracy. In others, it is imaginable that the efficiency of the algorithm (in terms of execution time or required memory storage) is the primary objective. There may also be other, more domain specific measures to evaluate the degree of goal achievement. The details of the resulting RNN-RE algorithm may depend on which of these goals is the aim. Preferably, however, the one and same algorithm should be generic enough to allow the user to choose among the goals.

7.2 New challenges

What should we expect from future RNN-RE algorithms? There are some challenging applications and requirements for RNN-RE algorithms that are worth suggesting.

7.2.1 Tailor-made quantization algorithms

The quantization algorithm is perhaps the most critical part of extracting rules from RNNs. But what characterizes a good quantization in the context of RNN-RE? It is not necessarily spatial requirements (N. E. Sharkey & Jackson, 1995), as is usually the case for evaluation of clustering techniques (Jain et al., 1999), but rather requirements based on properties of the extracted rule set. To have clusters that are *spatially* coherent and well separated is of less importance than the *fidelity* of the resulting rules. One would prefer a clustering technique in which the clusters are *functionally* coherent and well separated rather than spatially[1].

7.2.2 Goal oriented gradually refining rule extraction

Methods for controlling the "comprehensibility/fidelity tradeoff" are identified as an important line of research by Craven and Shavlik (1999). This "tradeoff" issue may be expanded to include techniques in which the user may, through the setting of a few parameters, not only have the ability to choose between fidelity and comprehensibility, but also fidelity and accuracy, fidelity and computation time etc. In an ideal RNN-RE algorithm the relation between execution time, fidelity and comprehensibility may be as illustrated in Figure 7.2. Rules should be refined gradually over time and the more time available, the higher the possibility of acquiring rules of high fidelity and/or comprehensibility ("anytime rule extraction" (Craven & Shavlik, 1999)).

One method for gradually refining rules may be to do "re-extraction" of uncertain/infrequent but possibly important rules by querying the network (Craven & Shavlik, 1994). This can, for example, be achieved by directly setting states in the network to be in the vicinity of the model vector (or something equivalent) of the macrostate of interest and then testing the effect of feeding the RNN various possible inputs. In Part III of this thesis, such reextraction is discussed in more detail due to of the results of Part II.

[1]Which is precisely what the RNN-RE method presented in Part II does using the Crystalline Vector Quantizer (cf. Chapter 10).

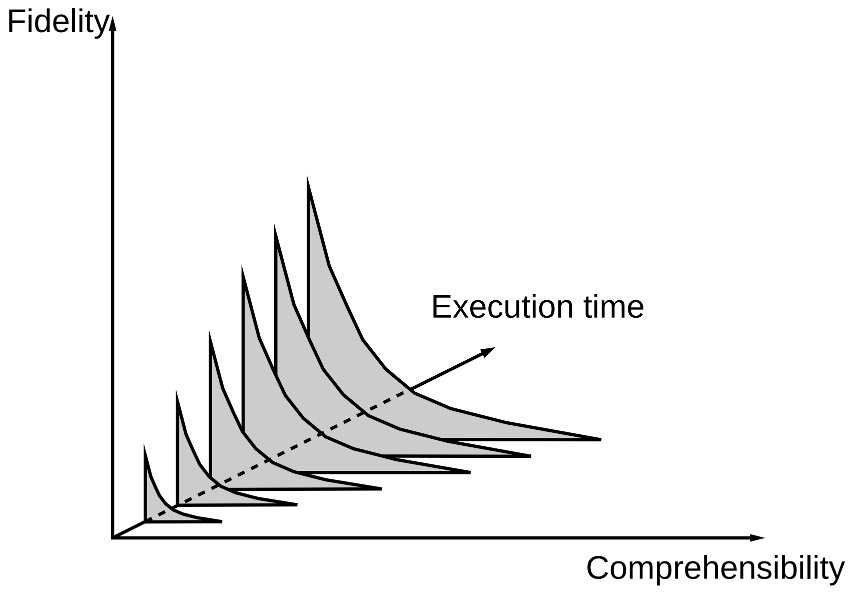


Figure 7.2: The relation between execution time, fidelity and comprehensibility for ideal RNN-RE algorithms with possible gradual refinement of the rules. The more time available, the more the degree of freedom in choosing between high fidelity and comprehensibility.

7.2.3 RNN comparisons and evaluations

A distance metric between RNNs could be defined by comparing rules extracted by RNN-RE. RNNs are otherwise difficult to compare directly since completely different weights can yield equivalent behaviour and small differences in weights may result in very different behaviours. This sort of distance metric could possibly be favourable if constructing heterogeneous RNN ensembles (Krogh & Vedelsby, 1995; A. J. C. Sharkey, 1996).

The concept that RNN-RE can be used as an indication of the complexity of the underlying RNN (or some other dynamic system) and domain could be further developed. Previous studies seem to show promising results (Crutchfield & Young, 1990; Blair & Pollack, 1997; Bakker & Jong, 2000) with regard to complexity estimations that go beyond Shannon entropy (Cover & Thomas, 1990) and minimum algorithmic description length complexity (Chaitin, 1987) (a.k.a. Kolmogorov complexity).

7.2.4 RNN debugging

The underlying task of the RNN (e.g., prediction) can be integrated with the rules in order to identify more exactly when and how erroneous behaviour occurs in the network. This can be achieved simply by marking which of the states in the extracted machines are involved in the errors. These errors can then perhaps be further retraced, in the rules, to the actual erroneous behaviour. This can possibly be further integrated with the training of the network by updating the weights only in situations identified by the rules as being part of an erroneous behaviour (cf. Schmidhuber (1992)).

7.3 Some practical recommendations

Since this thesis is, in part, aimed at attracting more researchers to the field it is perhaps beneficial to not only identify open issues, but also to provide some practical recommendations about *how* things should be done. These recommendations are partly a repetition of Craven and Shavlik (1999), but they are nevertheless important enough to be repeated.

Firstly, when developing a new RNN-RE algorithm, strive for generality (i.e. high portability). The usefulness of the algorithm developed will directly correlate with how easily it can be used on existing RNNs, originally implemented and tested without the intention of making them suitable for RE. Craven and Shavlik (1999) even suggest that the RE algorithms should be so general that not even the assumption that the underlying system is a neural network is necessary. Actually, there are indications that this is already a fact for most RNN-RE algorithms, considering the very limited assumptions of the underlying RNN (cf. Definition 3.1 or Definition 9.1).

Another good piece of advice is to seek out collaborators who already have RNNs they want to analyse (Craven & Shavlik, 1999). It is highly unlikely that there will be enough time to develop both state-of-the-art RNNs and state-of-the-art RNN-RE algorithms at the same time. Finding willing collaborators should not be too difficult since researchers applying novel RNNs on new domains will most likely benefit from the knowledge acquired through rule extraction.

Another important ingredient for enhancing the attraction of a technique is to make its implementation publicly available[2] (Craven & Shavlik, 1999). After all, accessibility of use by researchers who are quite busy pursuing their own line of work is the aim of the techniques.

7.4 Conclusions of Part I

Ideally, if the RNN-RE techniques developed so far had been really successful, they would have been among the first analysis techniques used when new RNN architectures were developed or a new domain was conquered. No other analysis tools seem to promise a more detailed and profound understanding of RNNs. But we are not there yet.

Despite numerous achievements, there seems to be no apparent common direction (or well defined goals) in previous RNN-RE research[3]. In most cases, developed algorithms are seemingly not built on the basis of previous results and there seems to be very slow (if any) progress towards handling more complex RNNs and domains with RNN-RE algorithms. In fact, only one algorithm has been used to any wide extent in the follow-up work, and moreover, it is the *first* RNN-RE algorithm developed (Giles, Miller, Chen, Chen & Sun, 1992). Surprisingly, it has not been replaced by anything significantly better in the years since then. Actually, more recent algorithms may very well be better, but they are still not used as frequently as the first one, and there are almost no comparative studies.

In the following part of this thesis, a novel RNN-RE algorithm is presented. The key to this algorithm was precisely the development of a tailor-made quantizer as suggested above (Section 7.2.1). Many of the practical recommendations mentioned above were, of course, central in the development of the algorithm, to make it portable and anytime extracting. In the final part of the thesis, more ambitious and speculative goals are stated for the field as RNN-RE is reinterpreted as a computational scientific process applied to simulated dynamic systems.

[2]An open source distribution of my own algorithm, presented in Part II, is under preparation on `cryssmex.sourceforge.net`.

[3]One of the goals with this thesis and with Jacobsson (2005) is to suggest such goals.

Part II

The Crystallizing Substochastic Sequential Machine Extractor

Chapter 8

Introduction to Part II

A number of techniques for extracting rules from RNNs were described in the previous part of the thesis. The common ingredients of most of these were identified (in Table 3.1 on page 17) as (1) the quantization of state space, (2) generation of data, (3) rule construction and (4) rule minimization. Although the presented techniques constituted a wide variety, their one common aspect is the separability of these ingredients from each other. These four constituents have not been integrated in these approaches and the quantization of the state space of the RNN has been treated as any Euclidean space with clusters of data, without accounting for the fact that the points in state space are part of a dynamic system in interaction with a domain.

This part of the thesis[1] presents a novel RNN-RE algorithm; `CrySSMEx`[2] (Crystallizing Substochastic Sequential Machine Extractor), which is parameter free, handles missing data, generates approximative rules if the underlying system is chaotic, and returns results at "any-time" (Craven & Shavlik, 1999), i.e. coarse models are initially created and then iteratively refined. The underlying concept of `CrySSMEx` is to observe the state and output of an RNN, quantize the state space, and refine the quantization of the state space such that the resulting machine typically is minimal, deterministic, and equivalent to the RNN. `CrySSMEx` is able to extract rules from RNNs in domains where other techniques cannot. At least, earlier techniques seem not to be feasible in these domains.

[1]Which is largely based on Jacobsson (2006).

[2]To be pronounced somewhat like "Christmas". An open source distribution of the algorithm is also under preparation at the time of preparation of this thesis (`cryssmex.sourceforge.net`).

8.1 Aim

CrySSMEx differs from many earlier approaches in that it strives for fidelity rather than accuracy of the rules. Fidelity is the degree to which the rules mimic the network, whereas accuracy is related to how well the rules generalize when applied to unseen examples (Andrews et al., 1995). When fidelity is the goal and the underlying network makes errors, the machine extracted from the network should also replicate those mistakes. Some earlier approaches have also focused on fidelity (e.g. Vahed & Omlin, 2004), but most work has had accuracy as the prime goal for the rules (e.g. Giles, Miller, Chen, Chen & Sun, 1992; Zeng et al., 1993). This is logical if the network is used as an intermediate step for acquiring symbolic knowledge from data, e.g., for grammar induction. In some cases this approach has been very successful when the extracted rules were equivalent to the symbolic data generator (e.g. Giles, Miller, Chen, Chen & Sun, 1992; Giles, Miller, Chen, Sun et al., 1992).

The strive for fidelity is beneficial because it makes the rules useful for analysing erroneous RNNs. One could compare an erroneous RNN to a sick patient and an RNN-RE algorithm to an instrument a doctor uses to diagnose the patient. The doctor would gain little from an accuracy-seeking instrument that describes the condition of the patient if completely healthy, which is basically what accuracy-optimizing methods strive for. Instead, the analysis tool should generate an analysis that reflects the *actual* condition of the patient.

Another difference between accuracy and fidelity is that the latter does not presuppose the existence of any task in which errors can be defined. Instead, the quality of extraction is measured on how well the extracted model mimics the underlying system. This allows for the analysis of simulated systems other than just RNNs. Therefore, in this thesis, the extraction of rules from RNNs is treated as an interesting special case of extraction from a broad range of dynamic systems (defined in Section 9.1).

8.2 What is new in `CrySSMEx`?

The three main criteria in the taxonomy of RNN-RE methods in Part I (Jacobsson, 2005) are: (1) the means of state observation, (2) the type of rules extracted, and (3) the state space quantization method.

The observation of states in `CrySSMEx`, as in many other approaches (e.g. Watrous & Kuhn, 1992; Manolios & Fanelli, 1994; Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999; Tiňo et al., 2004), is solely based on sampling the system as it behaves in its domain. The novel components of `CrySSMEx` are: the rule type (Section 9.2), and the quantization method (Chapter 10). But what really distinguishes `CrySSMEx` from all earlier methods, is the integration of the four basic elements found in previous methods (Jacobsson (2005), p. 1230):

- quantization of state,
- observation of the underlying system,
- rule construction and
- rule minimization.

These four subprocedures have typically been quite separable in RNN-RE algorithms. In earlier approaches, the quantization of the state space was achieved by traditional clustering techniques with no sensitivity to, nor any integration with, the dynamics of the RNN. Also, the minimization of the rules (when conducted at all) was just a postprocessing of the rules. In `CrySSMEx`, all four constituents are tightly integrated into one system resulting in an empirical loop of model refinement through model based data selection (cf. Chapter 11).

8.3 Overview

This part of the thesis is structured to enhance the understanding of the main loop of the algorithm (in Section 11.2). The algorithm should, however, be understandable, at an abstract level, without knowing all the details of the constituents. Therefore readers are recommended to briefly look at Algorithm 11.2 (on page 87) of Section 11.2, the point of convergence of this part of the thesis, before continuing to read Part II. To further aid readers, important abbreviations are listed in Table B.1 of Appendix B.

The remaining chapters of this thesis part is otherwise organized as follows: in Chapter 9 the specific class of dynamic systems that are analysable with `CrySSMEx` is defined together with a discrete stochastic model of these systems. A novel vector quantizer is described in Chapter 10. As mentioned, Chapter 11 connects the constituents of `CrySSMEx` into one coherent algorithm. While remaining chapters contain experiments, discussion and conclusions, future work suggestions and discussion are covered mainly in Part III.

Chapter 9

Modelling Dynamic Systems

This chapter introduces a class of dynamic systems (Section 9.1), a finite stochastic model of these systems (Section 9.2) and a means of transforming the dynamic system into the stochastic model through system observation (Section 9.2.3). The translation process of the system into a model is refined by other parts of `CrySSMEx` (Chapters 10–11) so that more precise translations can be made.

9.1 Situated Discrete Time Dynamic Systems

The target domain for `CrySSMEx` is a general class of dynamic systems which includes RNNs. Therefore, only properties of RNNs that are of importance for rule extraction are included. Other properties typically associated with neural networks, such as weights, activation functions and learning, are simply omitted.

The targeted class of systems is referred to in this thesis as *situated* discrete time dynamic system, incorporating state, input, output and dynamics of the system. The system is situated in the sense that it has a defined interface with a domain with which it interacts. Henceforth in the thesis, the extraction of rules from such dynamic systems rather than only from RNNs will be considered, but the underlying problems are precisely the same (cf. Definition 3.1).

9.1.1 Definition

Definition 9.1 A *situated discrete time dynamic system* (SDTDS), is 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*, $\gamma : S \times I \rightarrow S \times O$ is the *transition function*, and $n_s, n_i, n_o \in \mathbb{N}$ are the *dimensionalities* of the state, input and output spaces respectively. $\square$

Interpretation: If the system, at time t, occupies a state $\vec{s}(t)$ and is fed an input $\vec{\imath}(t)$, then the resulting next state and produced output is determined by $[\vec{s}(t+1), \vec{o}(t+1)] = \gamma(\vec{s}(t), \vec{\imath}(t))$. The current and initial state of the system are not included in the SDTDS model since it is something imposed on the system (the SDTDS specifies the framework and behaviour for any arbitrary initial state just as a function specifies the image of any arbitrary member of the domain of the function). To simplify descriptions, the transition function, γ, can be subdivided into two functions $\gamma_s : S \times I \rightarrow S$ and $\gamma_o : S \times I \rightarrow O$.

It should be noted that the functional dependencies are those of a Mealy system rather than a Moore system in that the output is determined by state *and* input rather than a function of state alone (Hopcroft & Ullman, 1979). The reason for this choice is that a Mealy model can subsume a Moore model but not necessarily vice versa[1] (if we only consider finite state machines, however, they are completely equivalent (Hopcroft & Ullman, 1979)).

In its current implementation, `CrySSMEx` also requires the set of input vectors to be finite, which for example is the case for any symbol processing RNN. This restriction is not included in the definition since it applies more to what is used as input to the SDTDS, rather than a restriction of the system itself. Other than that, there are no theoretical restrictions on the SDTDS as defined above for `CrySSMEx` to analyse it.

There are also some other "implicit requirements" (cf. Section 6.5) , made by a rule extraction algorithm of the underlying SDTDS, that cause some systems of general interest not to comply with the above definition (Jacobsson, 2005). For example, the state, input and output must be distinctly separable as well as fully and unintrusively observable. Moreover, γ must be a noise-free function, i.e. the observed system is assumed to be completely deterministic.

[1]A Moore model, and a Moore machine extraction version of `CrySSMEx` has also been implemented, but is not presented here since it involves small changes in many different parts of the descriptions.

9.1.2 Collection of data from an SDTDS

An RNN-RE algorithm should transform an RNN into a discrete model mimicking the RNN to a satisfactory degree. To do this, a *compositional* approach has typically been adopted where data is gathered from the internal activations of the RNN and a model is subsequently built from this (Tickle et al., 1998). Within the RNN-RE field, two sub-types of the compositional approach exist; one where the RNN-RE algorithm interacts directly with the RNN while performing a breadth first search, and another where the data is collected from the RNN during interaction with the domain in which it was trained (cf. Section 6.3). In `CrySSMEx`, the latter is chosen for three reasons: (1) the data (and hence the extracted model) will only contain aspects of the RNN relevant for the domain, (2) it is far more efficient since, in effect, the domain is used as a heuristic when searching among all the possible models that describe the behaviour of the system (Jacobsson & Ziemke, 2003b) (cf. Appendix D), and (3) it is possible to do the extraction off-line, i.e. pregenerated data can be used in `CrySSMEx` since no direct interaction between extractor and underlying system is needed.

When the SDTDS is set to hold a certain initial state and is then fed a sequence of input vectors from a domain it will generate a sequence of states and outputs as a result. This domain interaction is the basis for the data collection and the result is recorded as a sequence of *transition events*.

Definition 9.2 An *SDTDS transition event* at a time t, $\omega(t)$, is a quadruple $\langle \vec{s}(t), \vec{\imath}(t), \vec{o}(t+1), \vec{s}(t+1) \rangle \in \mathbb{R}^{n_s} \times \mathbb{R}^{n_i} \times \mathbb{R}^{n_o} \times \mathbb{R}^{n_s}$ where $\vec{s}(t+1)$ is the state vector reached after the SDTDS received input $\vec{\imath}(t)$ while occupying state $\vec{s}(t)$, and $\vec{o}(t+1)$ is the output generated in the transition. □

Definition 9.3 A *transition event set*, Ω, consists of selected transition events recorded from the SDTDS with a given set of input sequences. □

The reason that Ω is defined to consist of *selected* events is that it is quite possible that some events are not wanted in the model, e.g., when the user has made an explicit reset of the state with no wish to model the transition caused by this. The user may also want to let the system "settle in" before starting data collection. The fact that the user is allowed to choose a subset of available events to work with, may give the notion that the biases of the user may affect the results. However, such selection requirements are always part of any empirical process and the selection of data will necessarily affect the results. The initial state are also omitted from the definition SDTDS for this reason; possibly several initial states should be selected by the user to be appropriate for the aspects of the SDTDS the user wants to model.

9.1.3 Building a stochastic dynamic model from a quantized SDTDS

The most essential part of `CrySSMEx`, and all earlier RNN-RE algorithms, is the quantization of the state space. The set of possible states in the state space of the SDTDS is uncountable and must be transformed to a finite domain to make the extraction of a finite machine possible.

Definition 9.4 A *quantizer* $\Lambda : \mathbb{R}^n \rightarrow \{1, 2, \ldots m\}$ is a function that separates an n-dimensional real space into m uniquely labelled disjoint subspaces. The maximum number of subspaces, m (i.e. the cardinality of the codomain of function Λ), will, for pragmatic reasons, be denoted $|\Lambda|$. $\square$

Although not explicitly stated in most RNN-RE papers, all three spaces of RNNs (input, state and output) are actually labelled using some form of quantization function. The quantization of the state space is, of course, a central concern, but also the input and output need to be labelled into a finite set of symbols to produce the extracted finite machine. The state, input and output quantizers will be denoted Λ_s, Λ_i and Λ_o respectively.

The SDTDS is in itself, of course, capable of reacting according to any of the possible input vectors (since the SDTDS definition includes the whole vector spaces in the domains of the transition function), but in its current implementation `CrySSMEx` requires the input domain to be finite (and Λ_i must be invertible).

The frequencies of quantized transitions in the transition event set, Ω, are transformed into a joint probability distribution that will later be used to build a dynamic model which mimics the SDTDS (Section 9.2.3):

Definition 9.5 A *stochastic dynamic model* of an SDTDS is a joint probability mass function induced from a transition event set Ω and quantizers Λ_o, Λ_i and Λ_s

is defined as a function $p_\Omega : [1, |\Lambda_s|] \times [1, |\Lambda_i|] \times [1, |\Lambda_o|] \times [1, |\Lambda_s|] \to [0, 1]$ where $p_\Omega(i, k, l, j)$ denotes the probability, that if one picks a random transition event from Ω, it would be a transition from a state enumerated i by Λ_s, over an input vector enumerated k by Λ_i, which generated an output enumerated l by Λ_o, and a new state enumerated j by Λ_s[2]. □

9.2 Substochastic Sequential Machines

Stochastic machines have been extracted earlier (Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999), but without modelling the output of the system explicitly. In `CrySSMEx`, however, the output of the system will be modelled as well.

The stochastic dynamic model (p_Ω in Definition 9.5) collected from the SDTDS in interaction with its domain provides information about the estimated probabilities of the effect and outcome of transitions in the system as "viewed" through the quantizers. These probabilities are used to build a finite stochastic machine model of the SDTDS. This type of machine resembles *stochastic sequential machines* (Paz, 1971) or *probabilistic automata* (Rabin, 1963) but has some distinguishing features since there is a realistic possibility of model "incompleteness" due to a finite observed set of transition events. This is due to the fact that the sample of input sequences in Ω will not necessarily provide examples of *all* possible input symbols in *all* possible enumerations of the quantized SDTDS space. The choice here is to make a "closed world assumption", and consequentially, only what is observed in Ω will be included in the model.

Missing data must therefore be handled when the model is built from Ω. This causes the probabilistic model to become a *substochastic sequential machine* (SSM) rather than the *stochastic* sequential machines of Paz (1971). As a consequence, this incompleteness of the model implies that probability can "leak" out from the state of the machine during parsing of input sequences, causing the probability distributions to become substochastic (see Appendix A). The details of what this entails are clarified in the following sections. First, however, some additional definitions and notational conventions are introduced, followed by the full SSM definition.

[2]The awkward order of i, j, k, and l is due to other contexts of the variables of p_Ω later in this thesis.

9.2.1 Notation of probability distributions as vectors

Sometimes a probability distribution over a finite domain is preferably denoted as a vector (cf. Paz, 1971). The probability mass function over a discrete finite stochastic variable X, is denoted $p(X = x_i)$, or $p(x_i)$ for short. $p(x_i)$ is interpreted as the probability of X having the value x_i. If we want to express this probability as a vector it is convenient to just write $p(x_i)$ as $\vec{x}_i$. The full vector, representing the full distribution over X is denoted $\vec{x}$, i.e. with no index. The vector and probability notation of distributions will be used interchangeably since they are more conveniently expressed as one or the other depending on context. Important types of substochastic vectors and operations on them are defined in Appendix A.

9.2.2 SSM definition

Definition 9.6 A substochastic sequential machine (SSM) is a quadruple $\langle Q, X, Y, \mathcal{P} = \{p(q_j, y_l|q_i, x_k)\}\rangle$ where Q is a finite set of state elements (SEs), X is a finite set of input symbols, Y is a finite set of output symbols, and $\mathcal{P}$ is a finite set of conditional probabilities (cf. explanation of Equation 9.3) where $q_i, q_j \in Q$, $x_k \in X$ and $y_l \in Y$. $\square$

The terminology is here somewhat different from that of conventional finite state machines. The input and output domains of the SSM will still be considered alphabets of *symbols*, whereas the Q of the SSM will instead be denoted *state elements* or SEs[3] to not confuse them with the state of the SDTDS. Also, the actual state of the SSM is more properly described as a (sub)stochastic distribution over these elements. The interpretation of $p(q_j, y_l|q_i, x_k)$ is that it is the probability of the machine entering the SE q_j and in this transition producing symbol y_l given that it occupied only SE q_i and was fed input symbol x_k. A more detailed description of the SSM interpretation is given in Section 9.2.4 where the use of an SSM as a parser of input symbols is described. But first, the construction of an SSM from a model of the SDTDS will be described.

[3]See Table B.1 for a list of abbreviations.

9.2.3 Translation of an SDTDS into an SSM

It is quite straightforward to see the similarities of the SDTDS and the SSM (cf. definitions 9.1 and 9.6). The difference lies mainly in the discreteness of the input, state and output domains of the SSM versus the uncountable domains in the SDTDS. In practice, however, the SSM can be seen as a subclass of the set of SDTDSs since a substochastic SE distribution can be subsumed as an SDTDS state and correspondingly for input and output.

When transforming an SDTDS into an SSM-model, the uncountable domains S, I and O of the SDTDS are reduced to the finite domains, Q, X and Y respectively. The SSM is created from a quantized SDTDS so that the domains of the SSM are isomorphic to the codomains of the respective quantizers. In other words, Q of the SSM is isomorphic to $[1, |\Lambda_s|]$ and correspondingly for the input and output symbols. In the following text, an SE denoted $q_i \in Q$ corresponds to the portion of the state space of the SDTDS enumerated i by the Λ_s-quantizer.

The joint probabilities of observed and quantized SDTDS transitions (p_Ω), are translated into joint probabilities of *SSM transitions* according to:

$$\begin{gathered} p(q_i, x_k, y_l, q_j) = \\ p_\Omega\Big(\Lambda_s\big(\vec{s}(t)\big) = i, \Lambda_i\big(\vec{\imath}(t)\big) = k, \Lambda_o\big(\vec{o}(t+1)\big) = l, \Lambda_s\big(\vec{s}(t+1)\big) = j\Big) \end{gathered} \tag{9.1}$$

i.e. the joint probability of SSM transitions are defined so that they correspond to the observed frequency of transitions in the SDTDS. The conditional probability of the SSM, $p(q_j, y_l|q_i, x_k)$, is calculated from the joint probability according to equations 9.2 and 9.3.

$$p(q_i, x_k) = \sum_{j=1}^{|Q|} \sum_{l=1}^{|Y|} p(q_i, x_k, y_l, q_j) \tag{9.2}$$

$$p(q_j, y_l|q_i, x_k) = \begin{cases} \dfrac{p(q_i, x_k, y_l, q_j)}{p(q_i, x_k)} & \text{if } p(q_i, x_k) > 0 \\ 0 & \text{if } p(q_i, x_k) = 0 \end{cases} \tag{9.3}$$

Although conceptually appealing, the distribution $\mathcal{P} = \{p(q_j, y_l|q_i, x_k)\}$, is perhaps a bit haphazardly termed *conditional probabilities* since a conditional probability $p(a|b)$ traditionally is undefined if $p(b) = 0$. But in the SSM these need to be defined since there might actually be cases where there is *no* transition from a SE q_i over a specific symbol x_k, simply because there are no observations in Ω of any such event.

Definition 9.7 If $p(q_j, y_l|q_i, x_k) = 0$ for all $q_j \in Q$ and $y_l \in Y$, then the transition from q_i over input x_k will be referred to as a *dead* transition. $\square$

Definition 9.8 The procedure of transforming an SDTDS from Ω, through the stochastic dynamic model, p_Ω of Definition 9.5, into an SSM as defined above in equations 9.1–9.3 will in pseudo-code be denoted as
$ssm = \texttt{create\_machine}(\Omega, \Lambda_s, \Lambda_i, \Lambda_o)$ where Λ_s, Λ_i and Λ_o are the state, input and output quantizer respectively, and *ssm* the resulting SSM. $\square$

When an SSM is created with `create_machine`, the SDTDS from which Ω was sampled is referred to as *the underlying system* of the SSM. Next, the exact calculations of state and output of the SSM are described. The SSM processes input such that its distributions over Q and Y correspond to the *degree of belief* of the occupied state and output enumeration of the underlying system.

9.2.4 Parsing an input sequence using an SSM

Unlike a "standard" discrete Mealy machine where exactly one state is occupied at a time (Hopcroft & Ullman, 1979), the complete description of the state occupied by an SSM is the substochastic distribution over zero, one, or more SEs. Likewise, the transitions generate substochastic distributions of output symbols rather than individual symbols.

The exact calculations of distributions are as follows: Let $\vec{q}(t) = (\vec{q}_1(t), \vec{q}_2(t), \ldots, \vec{q}_n(t))$ be a substochastic vector denoting the distribution over Q at time t and $x_k(t) \in X$ be the input symbol fed to the machine in that time step. The resulting distribution vector over Q, $\vec{q}(t+1)$, is calculated by[4]

$$\vec{q}(t+1) = \mathcal{P}_q(\vec{q}(t), x_k(t)) \tag{9.4}$$

[4]Note that this is a case where the notational choice of letting $p(q_i) = \vec{q}_i$ comes into play (cf. Section 9.2.1), i.e. it is implicit that $\vec{q}_i(t+1)$ and $p(q_i(t+1))$ refer to the probability $p(Q = q_i)$ at time $t+1$.

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

$$\vec{q}_j(t+1) = \sum_{i=1}^{|Q|} \left(\vec{q}_i(t) \cdot \sum_{l=1}^{|Y|} p\big(q_j(t+1), y_l \big| q_i(t), x_k(t)\big) \right) \tag{9.5}$$

and concurrently, the distribution of output symbols $\vec{y}(t+1)$ over Y is generated in the transition by

$$\vec{y}(t+1) = \mathcal{P}_y(\vec{q}(t), x_k(t)) \tag{9.6}$$

where each element $\vec{y}_l(t+1)$ of $\vec{y}(t+1)$ (corresponding to a probability of an output symbol) is calculated by

$$\vec{y}_l(t+1) = \sum_{i=1}^{|Q|} \left(\vec{q}_i(t) \cdot \sum_{j=1}^{|Q|} p\big(q_j, y_l(t+1) \big| q_i(t), x_k(t)\big) \right) \tag{9.7}$$

Note that if the transition from $q_i(t)$ over $x_k(t)$ is dead and $\vec{q}_i(t) > 0$, then the respective sum of the probabilities of distribution $\vec{q}_j(t+1)$ and $\vec{y}_l(t+1)$, will be less than 1. In such cases, distributions of the machine will become substochastic (cf. Appendix A).

Another possibility of parsing is to, when possible, divide the probabilities with the sum of the probabilities after each symbol. This mode of parsing is referred to as *normalized* parsing.

$$\hat{\mathcal{P}}_*(\vec{q}(t), x_k(t)) = \texttt{normalize}(\mathcal{P}_*(\vec{q}(t), x_k(t))) \tag{9.8}$$

where the '$*$' is either q or y (`normalize` is defined in Appendix A).

One may argue that instead of the notion of substochastic probabilities and state "leaking" from the machine, it would be better to add an additional state element q_{dead} to which all dead transitions are then made (producing an additional "dead" output symbol, y_{dead})[5]. This can work, and it would also, as far as I can judge, create a machine equivalent to the machines of Paz (1971). It would, however, destroy the otherwise complete semantic connection between underlying system and SSM since there will be no corresponding elements in S and Y of the SDTDS

[5]This can be compared with the "rescue state" of Schellhammer et al. (1998) (cf. Table 5.7).

for q_{dead} and y_{dead} respectively.

To illustrate the parsing of symbol sequences with SSMs, some examples are examined in Section 9.2.7. But first some important types of SEs must be introduced. There are a number of properties of SSMs and SEs that can be used for a deeper analysis of the machines. In this thesis only the ones that are crucial for `CrySSMEx` are mentioned: deterministic and equivalent SSM SEs.

9.2.5 SSM determinism

An SSM will always be deterministic in the sense that the state element and output symbol distributions are always deterministically calculated. Therefore, the determinism of an SE is instead defined to reflect the degree to which the SSM determines the succeeding occupied state enumerations and output symbols of the underlying dynamic system. For this purpose entropy and, especially, conditional entropy (Cover & Thomas, 1990) are suitable (see Definition A.5 in Appendix A).

A conditional entropy $H(Y|X = x)$ can be interpreted as the remaining uncertainty of variable Y given that variable X would be known to have the value x. Here, the conditional SSM-based entropy of the output given an SE q_i and input x_k in an SSM ssm will be denoted $H_{ssm}(Y|Q = q_i, X = x_k)$ and is defined by

$$H_{ssm}(Y|Q = q_i, X = x_k) = H(\mathcal{P}_y(\vec{q}, x_k)) \tag{9.9}$$

where $\vec{q}$ is here the degenerate (see Appendix A) SE distribution vector with $\vec{q}_i = 1.0$. The conditional entropy of the SE given the previous SE and input symbol is likewisely denoted $H_{ssm}(Q|Q = q_i, X = x_k)$ and is here defined by

$$H_{ssm}(Q|Q = q_i, X = x_k) = H(\mathcal{P}_q(\vec{q}, x_k)) \tag{9.10}$$

with $\vec{q}$ degenerate as in equation 9.9.

The interpretations of the entropies in equations 9.9 and 9.10 are that given a distribution over Q, concentrated to only q_i, and the input then is x_k, they return the degree of uncertainty of the SSM regarding the succeeding output symbol and occupied state enumeration of the underlying SDTDS, respectively. This is an idealized interpretation due to the substochastic nature of the model. The conditional entropy will be zero also when the SSM has another type of uncertainty when transition from q_i over x_k is dead. In this sense, the conditional entropies of equations 9.9 and 9.10 are not entirely compatible with how the concept of entropy is typically used (cf. discussion after Definition A.5 and after Equation 9.3 in relation to the conditional probabilities).

Definition 9.9 An SE $q_i \in Q$ of an SSM *ssm* is *deterministic* iff $H_{ssm}(Y|Q = q_i, X = x_k) = 0$ and $H_{ssm}(Q|Q = q_i, X = x_k) = 0$ for $\forall x_k \in X$. □

A deterministic SE has exactly zero or one outgoing transition for each input symbol.

Definition 9.10 An SSM is deterministic iff all SEs $q_i \in Q$ are deterministic. □

If a machine is deterministic, and its initial SE distribution is degenerate, then all subsequent SE and output distributions will both be either degenerate or exhausted (cf. Appendix A). This definition of a deterministic machine differs somewhat from that of traditional deterministic finite automata (Hopcroft & Ullman, 1979), in which states (corresponding to the state elements of the SSM) must have transitions to exactly one state for all input symbols.

It is quite straightforward to see that a deterministic SSM, in which there are no dead transitions, is equivalent to the nonstochastic standard Mealy machines as defined in Hopcroft and Ullman (1979), if a degenerate distribution over Q is defined as initial state. Such a machine must always occupy only one SE at a time and generate one single output symbol at a time.

SSM determinism is used as a termination criterion in `CrySSMEx` (see Algorithm 11.2). The conditional entropies are also used as a basis for selection of the most informative state vectors of Ω in order to perform optimization of the SDTDS state quantizer (see Algorithm 11.1 on page 85).

9.2.6 Equivalence and nonequivalence of SEs

The second important property of SEs is equivalence. In automata theory, two states q_i and q_j of a machine are equivalent if, and only if, the output of the automata would be the same for all possible future input sequences independent of which of the two possible states that are occupied initially. This can be tested quite efficiently in traditional nonstochastic automata (Hopcroft & Ullman, 1979), but for stochastic machines it is somewhat more difficult. In fact, it is even impossible in general for substochastic machines since the model would not "know" what the outcome of dead transition would be in the underlying system. It would, for example, be impossible to determine what other state elements an SE with *no* outgoing transition is or is not equivalent to since the outcome of any possible future input sequence is undefined in the model. The only way to determine equivalence of such an SE, to other SEs, is to return to the underlying SDTDS to record the missing transitions, and thereby make it part of the SSM model. However, since this would break the closed world assumption, it is not considered. Rerecording of Ω is however considered an option in the future work sections of the thesis (cf. Section 18.4).

It is, however, possible to determine that two SEs are *not* equivalent if they, in their outgoing transitions, share some input symbols and transitions over these lead to discrepancies in the future output of the SSM. Therefore, an algorithm that returns true if and only if two SEs are *not decisively inequivalent* (NDI-equivalent for short) is provided[6]. For example, an SE which has no outgoing transitions will be NDI-equivalent with *all* other SEs since there will be no decisive evidence of the opposite. Two SEs with no input symbols in common in their outgoing transitions will also always be NDI-equivalent.

To determine the NDI-equivalence of SEs q_i and q_j, the recursive function `NDI_equivalent`$(ssm, \vec{u}, \vec{v}, \emptyset)$ (described in Algorithm 9.1) is called, where $\vec{u}$ and $\vec{v}$ are the corresponding degenerate SE distributions for q_i and q_j respectively (the need for the empty set is clarified in Algorithm 9.1), and the result is true or false depending on whether the SE distributions $\vec{u}$ and $\vec{v}$ are NDI-equivalent or not. The algorithm is highly recursive and uses a "trick" based on the support sets (see Appendix A) of the SE distributions to avoid infinite recursions that otherwise could occur. If we allow ourselves to jump ahead to a later example, consider the testing of equivalence between SEs q_5 and q_6 in Figure 9.2 (on page 74). When starting in

[6]It is also possible to test if SEs are decisively equivalent as well, i.e. when all subsequent SEs have the same symbols for outgoing transitions. But preliminary studies have shown that more interesting results are achieved using NDI-equivalence simply because dead transitions are quite common.

SE q_5 and the SSM is fed symbol b, the SE distribution is gradually approaching a pure SE q_6, but it will never quite reach it. The algorithm would not stop if it were not for the use of reencounters of SE support sets as a termination criteria. Since there is just a finite set of possible support sets (2^Q), this guarantees that the algorithm will terminate.

What is currently lacking, is formal proof that `NDI_equivalent` functions as intended for all possible SSMs under all possible conditions. Formal proof of a method for equivalence testing of states in *stochastic* sequential machines, however, does exist (Paz, 1971). In that proof, strong similarities with this algorithm do occur, but a formal 1:1 connection is yet to be achieved. For now, the experiments of Chapter 12 are the only indication that the algorithm as a whole functions for the presented cases. In addition to these experiments, the algorithm has been successfully tested in a number of hand-made SSMs, with properties that make them interesting to analyse with respect to SE equivalence, e.g., the SSM of Figure 9.2.

For three SEs q_i, q_j and q_k of an SSM it may very well hold that q_i and q_j are NDI-equivalent and likewise for q_j and q_k while q_i and q_k are not. In other words, the relation is not transitive. It is required, by other parts of `CrySSMEx` (see Algorithm 11.2) that states can be grouped into disjoint equivalence sets, which is not possible if the equivalence relation is not transitive (and symmetric and reflexive as well).

Definition 9.11 Let $\pi(q_i)$ denote the set of SEs with which q_i is NDI-equivalent. Two SEs q_i and q_j are defined as *universally NDI-equivalent* (UNDI-equivalence, for short) if $\pi(q_i) = \pi(q_j)$. $\square$

UNDI-equivalence is a transitive relation (symmetry and reflexiveness is inherited from the NDI-equivalence) and therefore can be used to define non-overlapping equivalence sets. There is, however, more than one way of translating the NDI-equivalence into a transitive relation and this issue is again mentioned in Section 16.1.

Definition 9.12 A *set of UNDI-equivalence sets*, E, consists of disjoint sets of SEs, $e \in E$ where $e \subseteq Q$ (with all es together covering all SEs in the SSM) and for all $q_i, q_j \in e$, q_i and q_j are UNDI-equivalent. In the pseudo-code notation, the

```
NDI_equivalent($ssm, \vec{u}, \vec{v}, H$)
Input: an SSM $ssm$, 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
        possible future input sequences.
begin
1   if $\exists x_k \in X : (\hat{\mathcal{P}}_y(\vec{u}, x_k) \neq \hat{\mathcal{P}}_y(\vec{v}, x_k) \wedge \sup(\mathcal{P}_q(\vec{u}, x_k)) \neq \emptyset \wedge$
            $\sup(\mathcal{P}_q(\vec{v}, x_k)) \neq \emptyset)$ then return false;
    /*i.e. the output must be the same for both SE distributions for all
      possible input symbols.                                              */
2   else if $\vec{u} = \vec{v}$ then return true;
    /*i.e. if the distributions are identical, they are equivalent.        */
3   else if $\langle \sup(\vec{u}), \sup(\vec{v}) \rangle \in H$ then return true;
    /*i.e. a loop has been encountered. Eventual inequivalence will be
      encountered in another branch of the recursion tree.                 */
4   else
        /*If the equivalence/inequivalence cannot be asserted, then subsequent
          inputs must be tested.                                           */
        $R := true$;
        $k := 1$;
        while $R = true \wedge x_k \in X$ do
            /*As long as no inequivalence has been shown ...               */
            $\vec{u}\,' = \hat{\mathcal{P}}_q(\vec{u}, x_k)$;
            $\vec{v}\,' = \hat{\mathcal{P}}_q(\vec{v}, x_k)$;
5           if $\sup(\vec{u}') \neq \emptyset \wedge \sup(\vec{v}') \neq \emptyset$ then
                /*... continue testing recursively.                        */
                $H' = H \cup \langle \sup(\vec{u}), \sup(\vec{v}) \rangle$;
                $R :=$ NDI_equivalent($ssm, \vec{u}\,', \vec{v}\,', H'$);
            end
            $k := k + 1$;
        end
        return $R$;
    end
end
```

Algorithm 9.1: The recursive function `NDI_equivalent`$(ssm, \vec{u}, \vec{v}, \emptyset)$ returns true if and only if there is no evidence that the future *ssm* output could differ depending on which of SE distributions $\vec{u}$ or $\vec{v}$ are occupied in the SSM. The if-statement on line 2 can actually be logically omitted, since line 3 will catch the equivalence in subsequent levels of recursion (line 4), but it makes the algorithm considerably more efficient in most realistic cases. The empty support set tests on lines 1&5 together with the normalized parsing ($\hat{\mathcal{P}}_q$ and $\hat{\mathcal{P}}_y$) cause the algorithm to return true when assessment of inequivalence cannot be performed.

function call $E =$ `generate_UNDI_equivalence_sets`(ssm) is be used to denote the generation of a set of UNDI-equivalence sets E from an SSM ssm. □

A machine with equivalent SEs can be collapsed to a smaller machine by collapsing all equivalent sets into new, individual SEs. This collapsing, or merging, is, however, part of another subalgorithm (the `merge_cvq`-function of Definition 10.7).

9.2.7 SSM examples and interpretations

Example 9.1 Consider an SSM with $Q = \{q_1, q_2\}$, $X = \{\mathbf{a}, \mathbf{b}\}$, $Y = \{\mathbf{c}, \mathbf{d}\}$ and transition probabilities $\mathcal{P} = \{p(q_2, \mathbf{c}|q_1, \mathbf{a}) = 1.0, p(q_2, \mathbf{c}|q_1, \mathbf{b}) = 0.1, p(q_1, \mathbf{c}|q_1, \mathbf{b}) = 0.9, p(q_1, \mathbf{c}|q_2, \mathbf{a}) = 0.8, p(q_1, \mathbf{d}|q_2, \mathbf{a}) = 0.2, p(q_2, \mathbf{d}|q_2, \mathbf{b}) = 1.0\}$ (SSM A in Figure 9.1). All zero probabilities are omitted from the descriptions, e.g., that $p(q_1, \mathbf{a}|q_1, \mathbf{a}) = 0.0$. If we let the initial SE vector be $\vec{q}(0) = (1.0, 0.0)$ (i.e. that $p(Q = q_1) = 1.0$ at time $t = 0$) and then parse the string **aabbba** with the machine, the sequence of SE and output symbol distribution vectors (where the two elements of vector $\vec{y}$ correspond to probabilities of symbol **c** and **d** respectively) would be as follows:
(**a**) $\vec{q}(1) = (0.0, 1.0)$, $\vec{y}(1) = (1.0, 0.0)$,
(**a**) $\vec{q}(2) = (1.0, 0.0)$, $\vec{y}(2) = (0.8, 0.2)$,
(**b**) $\vec{q}(3) = (0.9, 0.1)$, $\vec{y}(3) = (1.0, 0.0)$,
(**b**) $\vec{q}(4) = (0.81, 0.19)$, $\vec{y}(4) = (0.9, 0.1)$,
(**b**) $\vec{q}(5) = (0.729, 0.271)$, $\vec{y}(5) = (0.81, 0.19)$,
(**a**) $\vec{q}(6) = (0.271, 0.729)$, $\vec{y}(6) = (0.9458, 0.0542)$. □

Note that, since the SSM of Example 9.1 has no dead transitions, the sums of the SE and output probabilities are always one, respectively. In the next example an SSM that has some dead transitions is shown.

Example 9.2 Consider an SSM with $Q = \{q_1, q_2\}$, $X = \{\mathbf{a}, \mathbf{b}\}$, $Y = \{\mathbf{c}, \mathbf{d}\}$ and transition probabilities $\mathcal{P} = \{p(q_2, \mathbf{c}|q_1, \mathbf{a}) = 1.0, p(q_1, \mathbf{c}|q_1, \mathbf{b}) = 0.9, p(q_2, \mathbf{c}|q_1, \mathbf{b}) = 0.1, p(q_1, \mathbf{d}|q_2, \mathbf{a}) = 1.0\}$ (SSM B in Figure 9.1). Note that the machine has dead transitions since q_2 has no outgoing transition over symbol **b**. SE q_2 is also an example of a deterministic SE. If $\vec{q}(0) = (0.0, 1.0)$ and the SSM is fed symbol **b** as input the probabilities of all SEs and outputs would therefore immediately reach

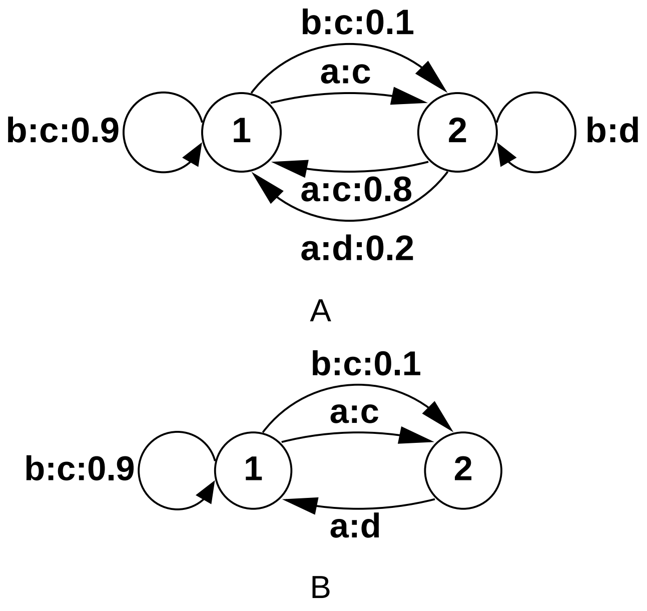


Figure 9.1: The two SSMs of examples 9.1 (A) and 9.2 (B) (with q_i denoted by i). A transition label $\mathbf{x}$:$\mathbf{y}$:p should be read as a transition with $\mathbf{x}$ as input and $\mathbf{y}$ as output and p as the probability of this transition. For example, the transition label "$\mathbf{b}$:$\mathbf{c}$:0.1" from q_1 to q_2 corresponds to the conditional probability $p(q_2, \mathbf{c}|q_1, \mathbf{b}) = 0.1$. If the p is 1.0, then the probability is omitted from the label.

zero. In other words, the possibility of being in SE q_2 is eliminated by the symbol $\mathbf{b}$, and as a consequence of the SSM "observing" $\mathbf{b}$, the probability of this impossibility vanishes from the machine. If we instead let $\vec{q}(0) = (1.0, 0.0)$ and then parse a sequence of t $\mathbf{b}$s, the sum of SE probabilities would be $0.9^{(t-1)}$ when $t \geq 1$. $\Box$

As the example illustrates, the the SSM acts as an observer of inputs, from which it derives a modelled degree of belief of what the actual enumeration of the state and output of the underlying system would be, given the same input sequence. Typically, if an SSM is given a uniform initial SE distribution, the SE distribution will, for each input symbol, gradually become more and more focused towards a small number of possible SEs (and output symbols). In a way, the SSM can be seen to "condense", or "crystallize" to a minimal hypothesis of the factual (knowable) state of the underlying system.

An SSM can be quite different and counter-intuitive compared to state machines typically encountered in the literature which is illustrated in the next example.

Example 9.3 The SSM of Figure 9.2 represents a more complex SSM. This machine is not fully connected and also contains a “dead SE” (cf. Definition 9.7) from which there are no transitions (q_{10}). This is a perfectly correct form of SSM and, if provided with an initial SE distribution, this machine can process input sequences just as the SSMs of the previous examples. In this machine, many properties of the UNDI-equivalence become clear. The set of equivalence sets returned by `generate_UNDI_equivalence_sets` is $\{\{q_1\}, \{q_2, q_3, q_4\}, \{q_5, q_6, q_7\}, \{q_8\}, \{q_9\}, \{q_{10}\}\}$. State element q_{10} will, since it has no outgoing transition, be NDI-equivalent with all other SEs. However, since it is the only element with this property, it is not UNDI-equivalent with anything but itself. q_8 is, on the other hand, the only one NDI-equivalent with only itself. One can easily see that the SEs of the equivalence sets $\{q_2, q_3, q_4\}$ and $\{q_5, q_6, q_7\}$ have the output symbols in common, respectively. q_3 is special since it has no outgoing transition over symbol **b**, whereas q_2 and q_4 have. q_3 is, however, NDI-equivalent with SEs q_2 and q_4 since it cannot be decided that symbol **b** should result in any different output given any of these three SEs. Then, for the same reason, why is not q_9 UNDI-equivalent to q_5, q_6 and q_7, although it too, is constantly giving **c** as output? The reason is that SE q_9 is also NDI-equivalent with q_1, which none of q_5, q_6 and q_7 are, therefore it is not UNDI-equivalent with them the way q_3 is with q_2 and q_4. If one added q_{11}, NDI-equivalent with q_3, but not with q_2 and q_4, then this situation would change (even though q_{11} may seem completely unrelated to q_3). Another aspect to notice is that the transition from the equivalence set $\{q_2, q_3, q_4\}$, from q_2 to q_{10}, makes no difference for the assessment of the equivalence of q_2 with q_3 and q_4 since the transitions is to a dead SE from which no decisive inequivalences can be derived. $\square$

The format of the extracted rules of `CrySSMEx` has now been described. The next step is to define a vector quantization function which is later orchestrated to work in conjunction with these rules.

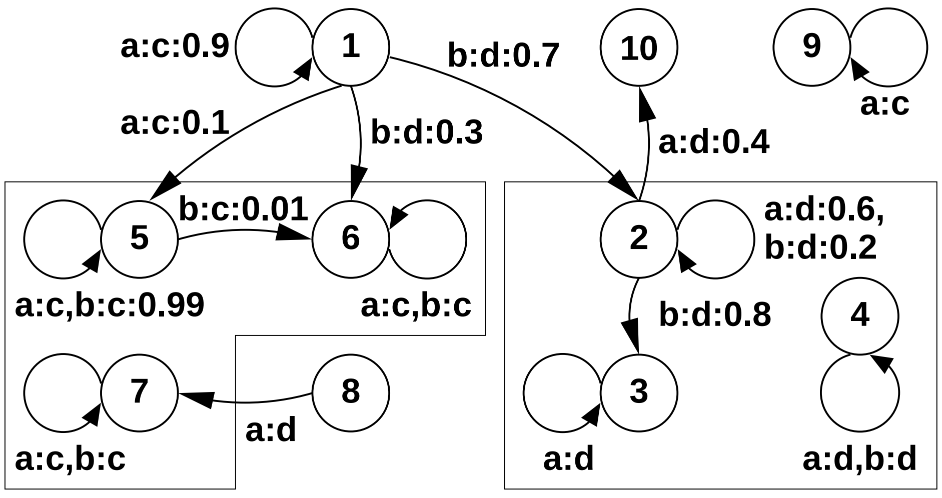


Figure 9.2: A more complex SSM example where UNDI-equivalence sets have been grouped together. See discussion in text of Example 9.3.

Chapter 10

The Crystalline Vector Quantizer

The observation and quantization of the state space of the underlying SDTDS is perhaps the most signifying constituent of RNN-RE algorithms. In previous work, quite traditional clustering algorithms have been used to partition the state space of the RNN (Jacobsson, 2005), e.g., self-organizing maps and *k*-means clustering (cf. Section 6.2). The problem with these clustering algorithms is that they partition the state space solely according to spatial properties, e.g., so that datapoints have low intracluster distances and high intercluster distances (Everitt, Landau & Leese, 2001). In the case of RNNs and other dynamic systems, however, the spatial requirements should give way to *functional requirements*. The spatial (e.g., Euclidean) proximity of two states of the SDTDS is of less importance for deciding if they belong to the same cluster, than the invariance of the apparent behaviour of the SDTDS with respect to these states. Similar problems also exist when clustering internal activations of feedforward networks (N. E. Sharkey & Jackson, 1995). This means, among other things, that the quantizer may need to have varying granularity in different regions of the state space.

A partitioning that is completely guided only by the dynamics of the SDTDS is, however, an idealization (Casey, 1996; Blair & Pollack, 1997; Jacobsson & Ziemke, 2003b). Instead we will have to be content with partitions that are equivalent for a specific and finite set of input sequences (in the finite Ω of Definition 9.3).

To satisfy the functional requirements, a quantizer that allows generation of a division of the state space based on spatial properties is needed, as well as splitting and merging regions into new ones when the functional requirements are not satisfied (details of when exactly when it is appropriate to split or merge are covered in Chapter 11). For this purpose, a novel quantizer is suggested, the *Crystalline Vector Quantizer* (CVQ). The CVQ has some resemblance to the hierarchical decision tree representation extracted from feed-forward networks by Craven and Shavlik (1996), but differs in the details.

The CVQ is built upon a graph which is defined below. How the information of this graph is used to quantize a vector space is described in Section 10.2 and CVQ training in Section 10.3.

10.1 Definition of CVQ graph

Definition 10.1 A *CVQ graph* is a quadruple
$CVQ = \langle N_{Leaf}, N_{VQ}, N_{Merged}, n_{root} \rangle$ where $n_{root} \in N_{Leaf} \cup N_{VQ}$ is the root node of the CVQ graph, in which the constituents are defined as in definitions 10.2–10.4. $\square$

The CVQ graph is a directed graph and could thus be described as a set of vertices and edges, but for notational reasons it is easier to omit the edges from the description and instead of edges let nodes have explicit references to other nodes. The first node type, however, has no explicit references to any other nodes.

Definition 10.2 A *leaf node* in a CVQ graph $n \in N_{Leaf}$ has only one constituent, $n = \langle ID \rangle$, where $ID \in \mathbb{N}$ is a unique enumeration of the leaf nodes within the CVQ and $1 \leq ID \leq |N_{Leaf}|$. $\square$

Definition 10.3 A *Vector Quantizer (VQ) node* in a CVQ graph, $n \in N_{VQ}$ is a tuple $n = \langle M, C \rangle$ where M is a list of K *model vectors*, $[\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_K]$ where $[\vec{m}]_i \in \mathbb{R}^d$, and C is a (nonrepetative) list of child nodes $[c_1, c_2, \ldots, c_K]$ where $c_i \in N_{Leaf} \cup N_{Merged} \cup N_{VQ}$. $d \in \mathbb{N}$ is the dimensionality of the vector space which the CVQ will be used to quantize. $\square$

Definition 10.4 A *merged node* in a CVQ graph, $n \in N_{Merged}$, contains only a "link", $n = \langle n_{group} \rangle$, where $n_{group} \in N_{Leaf} \cup N_{Merged} \cup N_{VQ}$. $\square$

The interpretation is clarified in the next section where the use of a CVQ as quantization function is described in which all CVQ node constituents are of relevance. The example of Figure 10.1 is also beneficial for understanding the interpretation of the CVQ nodes.

The constituents of a CVQ are simply as defined above, but there are, of course, a number of constraints for how the CVQ graph can be constructed, e.g., that there may be no cycles in the graph. These constraints cannot be simply formalized, but are quite intuitive. Therefore, instead of a lengthy formal description, an example illustrates a typical CVQ topology in Figure 10.1. Also, the way the `CrySSMEx` algorithm builds the CVQ defines the constraints in exact detail (Chapter 11).

Firstly, however, some useful implicit properties of CVQ nodes should be defined.

Definition 10.5 The *parent set* of a node n is denoted[1] $n.Parent$ and refers to the set of all nodes where $N \in N_{Merged}$ or $N \in N_{VQ}$ where $N.n_{group} = n$ or $n \in N.C$ for merged and VQ node parents respectively ($n_{root}.Parent = \emptyset$). $\square$

A node, n, can have two types of parent sets; either it has a set of merged nodes that are linked to n, or it has just one VQ node (as illustrated in the example of Figure 10.1).

Definition 10.6 The level $L \in \mathbb{N}$ of a node n is denoted $n.L$. The level of the root node is 0 and of all the other nodes $n.L = \max\limits_{n_p \in n.Parent}(n_p.L)$ if $n.Parent \subset N_{Merged}$ and $n.L = n_p.L + 1$ if $n.Parent = \{n_p\}$ and $n_p \in N_{VQ}$. $\square$

The level of a node reflects how many VQ nodes, or *splits*, are maximally required to reach the node from n_{root} (see Sections 10.2 and 10.3 for more details on CVQ graph interpretation and splits). The example in Figure 10.1 illustrates the structure of a typical CVQ graph.

10.2 Quantizing with a CVQ

When a CVQ is used as a quantizer (Definition 9.4) the corresponding quantization function is denoted Λ_{cvq} and is in turn defined by the recursive function `winner` : $N_{Leaf} \cup N_{Merged} \cup N_{VQ} \times \mathbb{R}^d \rightarrow \{1, 2, \ldots m\}$ as defined in equation:

[1]An object orientation like notation is adopted here, where $X.Y$ means "The Y of X".

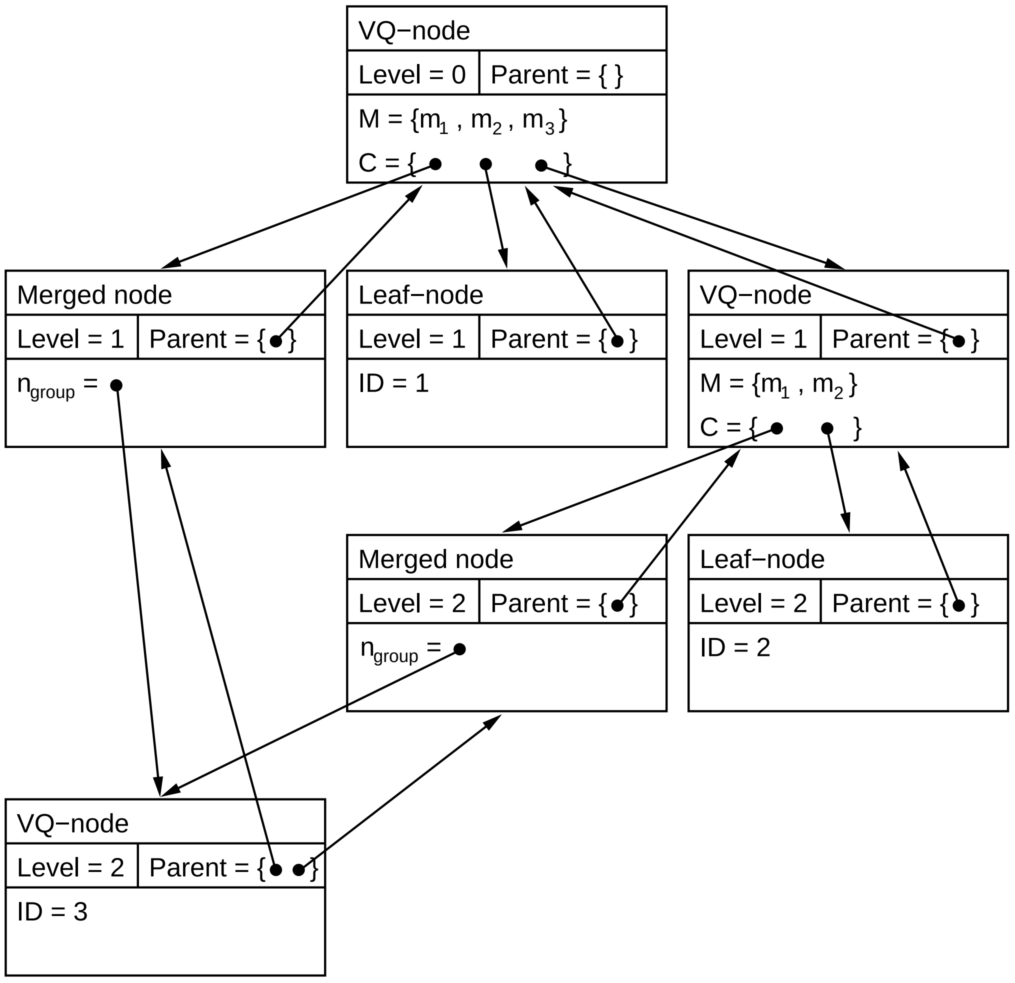


Figure 10.1: Example of a CVQ with $N_{Leaf} = \{n_2, n_5, n_6\}$, $N_{Merged} = \{n_1, n_4\}$, $N_{VQ} = \{n_0, n_3\}$ and $n_{root} = n_0$.

$$\Lambda_{cvq}(\vec{v}) = \texttt{winner}(n_{root}, \vec{v}) \tag{10.1}$$

where `winner` is recursively defined as

$$\texttt{winner}(n, \vec{v}) = \begin{cases} n.ID & \text{if } n \in N_{Leaf} \\ \texttt{winner}(n.n_{group}, \vec{v}) & \text{if } n \in N_{Merged} \\ \texttt{winner}(n.c_w, \vec{v}) & \text{if } n \in N_{VQ} \end{cases} \tag{10.2}$$

where w, the index of the winning child of a VQ-node, is determined according to

$$w = \underset{1 \le i \le |n.C|}{\operatorname{argmin}} \|\vec{v} - n.[\vec{m}]_i\| \tag{10.3}$$

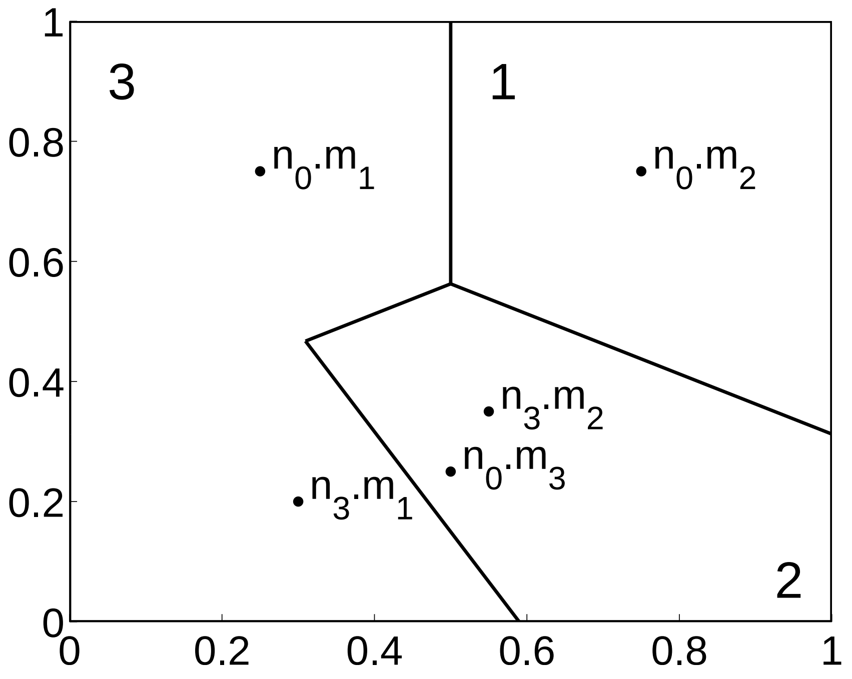


Figure 10.2: How a two-dimensional space would be quantized if the example CVQ in Figure 10.1 had model vectors $n_0.M = [(0.25, 0.75), (0.75, 0.75), (0.5, 0.25)]$ and $n_3.M = [(0.30, 0.20), (0.55, 0.35)]$.

where $\|\vec{v} - n.[\vec{m}]_i\|$ denotes the Euclidean distance between the vector to be quantized and the ith model vector of the VQ-node. If two model vectors have equal distance to the data vector, the smaller of the indices will be returned.

Example 10.1 If a vector $\vec{v}$ is classified by the CVQ of Figure 10.1, the classification starts with the root, which is a VQ node. $\vec{v}$ is compared to the model vectors and the closest such is chosen as a winner. If $\vec{m}_1$ is the closest, then the merged node (n_1) is entered from which the leaf node (n_6) with $ID = 3$ is immediately entered and 3 is returned. In briefer terms:
$\Lambda_{cvq}(\vec{v}) = \texttt{winner}(n_0, \vec{v}) = \texttt{winner}(n_1, \vec{v}) = \texttt{winner}(n_6, \vec{v}) = n_6.ID = 3.$ The division of a two-dimensional state space using the CVQ of Figure 10.1 with exemplified instantiated model vectors is illustrated in Figure 10.2. □

10.3 CVQ training

The training of the CVQ in `CrySSMEx` is tightly connected with operations of the SSM and sampling of the SDTDS (as discussed in Section 8.2). Here, however, the operations that are later used to refine the CVQ are defined as independent of their role in `CrySSMEx` (see Chapter 11 for this context).

The training consists of replacing leaf nodes with either merged nodes or VQ nodes, add new leaf nodes, and then reenumerate the IDs appropriately. Replacement of a leaf node with a VQ-node results in a larger number of leaf nodes and is referred to as *CVQ splitting*. Replacement of several leaf nodes with merged nodes results in a smaller number of leaf nodes and is referred to as *CVQ merging*. After completion of each of these operations, leaf nodes will be reenumerated.

Firstly, merging is described, followed by basic splitting, then an operation called *complete* splitting. "The user" which is mentioned in the following descriptions, is another part of the `CrySSMEx` algorithm, but it should of course be possible to use CVQ in other contexts.

10.3.1 The initial CVQ

The initial CVQ, denoted cvq^0, is the simplest possible CVQ consisting of only one leaf node (n_{root}) with $ID = 1$. All vectors will thereby be quantized as $n_{root}.ID = 1$ by the initial CVQ.

10.3.2 Merging

The merging of nodes in a CVQ corresponds to merging regions in the quantized space. This is conveniently described with an example:

Example 10.2 In the example of Figure 10.1, nodes n_1 and n_4 have been merged into n_6. Before this merge, n_1 and n_4 were two separate leaf nodes, but then the "user" discovered that the corresponding regions should not be separated, for some reason. The merge was subsequently conducted by creating a new leaf node, n_6, and then replacing the leaf nodes n_1 and n_4 with merged nodes connected to n_6. $\square$

In principle, any number of leaf nodes can be merged simultaneously. The merge is an operation on the CVQ graph, not necessarily related to any spatial properties of the quantized space, i.e. disconnected regions can be merged. The decicion of which subset of leaf nodes to merge is also entirely independent from their position in the CVQ graph.

Definition 10.7 The merging of one or more groups of leaf nodes will be denoted $cvq' := \texttt{merge\_cvq}(cvq, E)$ where E is a set of disjoint sets of IDs covering all leafs of the CVQ. The result, cvq', is the CVQ where leaves have been merged into one new leaf node per set in E (trivial sets in E, with only one member, are simply ignored). The leaf nodes are also re-enumerated before returning the resulting CVQ. □

E is later (in Algorithm 11.2) connected to the set of equivalence sets generated from SSMs by the function `generate_UNDI_equivalence_sets` (described in Definition 9.12).

Example 10.3 If $cvq' = \texttt{merge\_cvq}(cvq, \{\{1, 3, 5\}, \{2, 4\}, \{6\}\})$ is called, it will replace leaf nodes with IDs 1, 3 and 5 with merged nodes connected to a new leaf node, and correspondingly for 2 and 4. The leaf node with $ID = 6$ will be left unaltered. CVQ-based quantization function $\Lambda_{cvq'}$ will then quantize vectors into the range $[1, 3]$ whereas Λ_{cvq} quantized into the range $[1, 6]$. □

10.3.3 Basic splitting

When a CVQ leaf node is split, this amounts to splitting the corresponding region enumerated by that leaf. It is simpler to also describe this mechanism with an example:

Example 10.4 A set of two-dimensional data vectors V are quantized as in Figure 10.2. Now, the user has discovered that there are actually two types of vectors quantized as 1 (let us call the set of these vectors V_1). The user wants the two classes to be correctly separated by the CVQ. To do this the user collects all data vectors $V_1 = \{\vec{v}_i : \Lambda_{cvq}(\vec{v}_i) = 1\}$ and separates this set into two sets V_1^+ and V_1^- corresponding to the two classes. The node with $ID = 1$ (i.e. n_2 in Figure 10.1) would then be replaced with a VQ node with two model vectors and two new leaf nodes as children. The model vectors of the VQ node is then set to be the average of the vectors in sets V_1^+ and V_1^- respectively. □

In the above example, a leaf was only split into two new leaves, but in general, a leaf's corresponding region in the quantized space can be split into any number of regions.

10.3.4 Complete splitting

It is possible that the model vectors will not perfectly separate the data vectors after a basic split, since they might be linearly inseparable or the average vectors of the data sets may not be the perfect model vectors for separating the data[2]. In such cases, `CrySSMEx` would typically re-split the resulting region again automatically (see Chapter 11 where `CrySSMEx` is described in detail). It is however possible that an imperfect split may cause a non-minimal machine to be extracted and also that `CrySSMEx` will not terminate due to spurious SEs. Therefore a "perfect", or complete, split mechanism has also been devised.

To perform a complete split, the splitting is first conducted as in Example 10.4. But if the enumerated data vectors are still not separated, then the new leafs will be re-split using the corresponding subsets of the data vectors until the data vector class can be uniquely inferred from the identity of the involved leaf nodes. After this, all involved leaf nodes "belonging" to the same class are merged.

Definition 10.8 The complete split of several VQ nodes using a number of data sets at once will be denoted $cvq' := \texttt{split\_cvq}(cvq, D)$ where cvq is the CVQ to be split and $D = [D_1, D_2, \ldots D_{|\Lambda_{cvq}|}]$ is a list of data sets where D_i is the data set for splitting the leaf node with $ID = i$ (if the node should not be split, then $D_i = \emptyset$). The elements of a data set are pairs $\langle \vec{v}, \ell \rangle$ where $\vec{v} \in \mathbb{R}^n$ is the data vector and $\ell \in \mathbb{N}$ is label, or class, of the data vector. The leaf nodes of cvq' are also re-enumerated immediately after the completion of all splits. $\square$

There is also a possibility that the averages of two or more classes are exactly the same, in which case the splitting will fail completely. It is very unlikely this will occur by chance and no fixes are included in the definition of the algorithm.

[2] In fact, it can be quite inefficient to use the average as model vectors. The reason that model vectors are chosen as such is basically that it is simple, deterministic and does not require any parameters. Other, more sophisticated methods, such as resource allocating learning vector quantization (Everitt et al., 2001), have also been tested, but it does not really make a big difference apart from longer computation times and somewhat smaller CVQ graphs.

It has not occurred in any of the experiments (Chapter 12), and if it did, the implemented algorithm (`cryssmex.sourceforge.net`) would abort execution and generate a warning. It is of course also very important that there is no pair of differently classified but identical data vectors. This should not happen in the context of `CrySSMEx` due to the way data is collected from a deterministic machine, but it should be kept in mind if the CVQ is to be used in another context.

Chapter 11

The Crystallizing Substochastic Sequential Machine Extractor

11.1 Data selection from Ω

Perhaps the most important point of convergence of the various constituents described so far, in this thesis, is where subsets of Ω are selected and classified based on properties of the extracted SSM. The goal of `CrySSMEx` is to generate a deterministic SSM from the underlying deterministic SDTDS by dividing the state space into a minimal set of enumerated regions that can be used to describe the SDTDS perfectly in the context of Ω. To do this, indeterministic SEs of the SSM are targeted for splitting in the corresponding CVQ using selected state vectors from Ω. The basis for the selection of state vectors is to choose the set which should convey the most information, primarily of the output of the SSM and secondarily, the next state element of the SSM. The entropies $H_{ssm}(Y|Q = q_i, X = x_k)$ and $H_{ssm}(Q|Q = q_i, X = x_k)$ (definitions 9.9 and 9.10 respectively) are used for this selection. This basis for selection is not the only one possible, however, and this is mentioned again in Section 16.1. The entire selection procedure is contained in the function `collect_split_data`, described in Algorithm 11.1.

11.2 `CrySSMEx` main loop

The ingredients for `CrySSMEx` have now been presented:

```
collect_split_data($\Omega, ssm, \Lambda_i, \Lambda_s, \Lambda_o$)
Input: A transition event set, $\Omega$, an SSM, $ssm$, an input quantizer, $\Lambda_i$, a
       state quantizer, $\Lambda_s$, an output quantizer, $\Lambda_o$.
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 $\exists x_m : H_{ssm}(Y|Q = q_i, X = x_m) > 0$ then
            $x_{max} := \operatorname{argmax}_{x_m \in X} H_{ssm}(Y|Q = q_i, X = x_m)$;
            if $x_k = x_{max}$ then
                /*If output indeterministic with respect to $q_i$ and $x_k$, label the
                   state vector with the output symbol, $y_l$.                */
                $D_i := D_i \cup \langle \vec{s}(t), y_l \rangle$;
            end
        else if $\exists x_m : H_{ssm}(Q|Q = q_i, X = x_m) > 0$ then
            /*If output is uniquely determined from $q_i$, but next state is not,
               label state vector using next SE, $q_j$.                        */
            $x_{max} := \operatorname{argmax}_{x_m \in X} H_{ssm}(Q|Q = q_i, X = x_m)$;
            if $x_k = x_{max}$ then
                $D_i := D_i \cup \langle \vec{s}(t), q_j \rangle$;
            end
        endif
    end
    return $D$;
end
```

Algorithm 11.1: `collect_split_data` selects state vectors from Ω and labels them according to either Λ_o or Λ_s such that they are suitable for use in splitting CVQ nodes. The resulting list of data sets, D consists of one data set of labelled state vectors for each SSM SE, i.e. D_i corresponds to the data set for splitting state q_i.

- the SDTDS which represents the class of specimens for `CrySSMEx` to analyse (Definition 9.1),
- the data set, i.e. the SDTDS transition event set Ω (Definition 9.3),
- SSMs, which can be viewed as a subtype of SDTDSs (Definition 9.6),
- quantizers, e.g., Λ_i, Λ_s and Λ_o (Definition 9.4),
- SDTDS translation into SSM through quantization of input, output and state (Definition 9.8),
- the SSM transition functions $\mathcal{P}_q$ and $\mathcal{P}_y$ (Equations 9.4 and 9.6) and $\hat{\mathcal{P}}_*$ (Equation 9.8),
- the generation of UNDI-equivalence sets of SEs in SSMs (Definition 9.12),
- the CVQ (Definitions 10.1 to 10.3), used as a quantizer of vectors through the function Λ_{cvq} (Equation 10.1),
- merging and splitting of CVQ leaf nodes (Definitions 10.7 and 10.8),
- a mechanism for selecting and labelling state vectors of Ω based on properties of the SSM (Algorithm 11.1).

These constituents are integrated into the `CrySSMEx`-algorithm as described in Algorithm 11.2. The principle behind the algorithm is that the SSM should be kept as small as possible through the merging of SEs that are UNDI-equivalent while at the same time splitting indeterministic SEs. It is important to decide before an SE is deemed to be indeterministic, that it is not so because it, over one input symbol, transits to two or more SEs which are really equivalent (or at least UNDI-equivalent). If the machine was not minimized through the merging of equivalent state elements, it would risk resulting in an explosion of SEs due to unjustified splits.

If the algorithm does not converge in due time, additional termination criteria could be added. For example, one may want to limit the number of possible iterations, or put a limit on $|Q|$. The extracted, then possibly indeterministic, SSM will still be a model of the underlying SDTDS, and moreover, the more computational resources dedicated to iterate `CrySSMEx`, the better a model the SSM will be of the SDTDS, in terms of fidelity.

```
CrySSMEx($\Omega, \Lambda_o$)
Input: An SDTDS transition event set, $\Omega$, and an output quantization
       function, $\Lambda_o$.
Output: A deterministic SSM mimicking the SDTDS within the domain $\Omega$
        as described by $\Lambda_o$.
begin
    let $\Lambda_i$ be an invertible quantizer for all $I$ in $\Omega$;
    $i := 0$;
    $ssm^0 :=$ 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
```

Algorithm 11.2: The main loop of CrySSMEx.

Chapter 12

Experiments

The main purpose of the experiments in this thesis is to show that `CrySSMEx` manages to extract machines in contexts previously unsolved using RNN-RE algorithms. Another goal is to identify weaknesses of `CrySSMEx` by running it on notoriously challenging SDTDSs. A deeper analysis of how and why `CrySSMEx` behaves as it does, as well as of the resulting SSMs and CVQs will, however, have to be postponed for future work (discussed more in Chapter 16). Other relevant experiments are also presented in Jacobsson and Ziemke (2003a) and Jacobsson and Ziemke (2003b), which are both included verbatim in appendices C and D respectively. Jacobsson and Ziemke (2003a)[1] demonstrated how seemingly minor differences in testing procedure had significant effects on estimated results in the $\mathbf{a}^n\mathbf{b}^n$-domain used in Section 12.2. Jacobsson and Ziemke (2003b) compared breadth first search RNN-RE with a sampling based RNN-RE.

12.1 An illustrative example

Most previous work on RNN-RE algorithms has been experimentally tested on regular language domains (cf. Section 6.4). The aim of this experiment is to demonstrate that this kind of domain is trivial and at the same time illustrate the extraction process. It has already been proven that if an RNN is robustly performing a regular language recognition task, then this model can always be extracted (Casey, 1996). But no technique can warrant that such an extraction is possible in practice:

[1]Which in turn was an extension of the work in Jacobsson (1999).

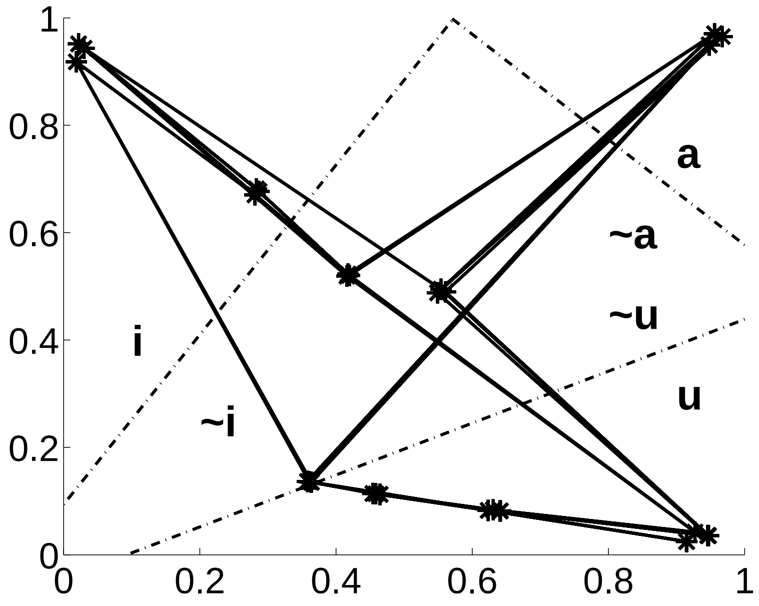


Figure 12.1: The state space of the RNN in the "**badiiguuu**"-domain. The states ($*$) and transitions between states are shown and the decision hyperplanes (N. E. Sharkey & Jackson, 1995) for each output unit are plotted (only three are visible).

there is no guarantee for `CrySSMEx` either, of course, but it does seem to be quite a straightforward process.

Elman (1990) used a simple regular language to train a simple recurrent network (SRN). The domain consisted of three subsequences **ba**, **dii** and **guuu** repeated in random order, e.g., **babadiibaguuudiiguuu...**. The task for the RNN was to do next-symbol prediction. In essence, only the vowels were at all predictable. In this thesis, an SRN with two hidden nodes was trained on this domain with the symbols represented as six dimensional vectors with one node active for respective symbol.

To generate Ω, a string of 10^5 randomly ordered substrings was used on the trained RNN. The state space of the RNN is shown in Figure 12.1. Three iterations completed the extraction and the sequence of state space divisions, the CVQ graphs and the SSMs are illustrated in Figure 12.2. The breadth first technique of Giles, Miller, Chen, Chen and Sun (1992) has also been tested on this domain, and it resulted in a large number of states never visited by the RNN when predicting the actual strings (Jacobsson & Ziemke, 2003b) (cf. Appendix D).

Most essential features of `CrySSMEx` are exemplified in this extraction. In the initial SSM, ssm^0, it can be seen that input symbol **i** generates the maximum amount of uncertainty regarding the output symbol (the output symbol **C** here

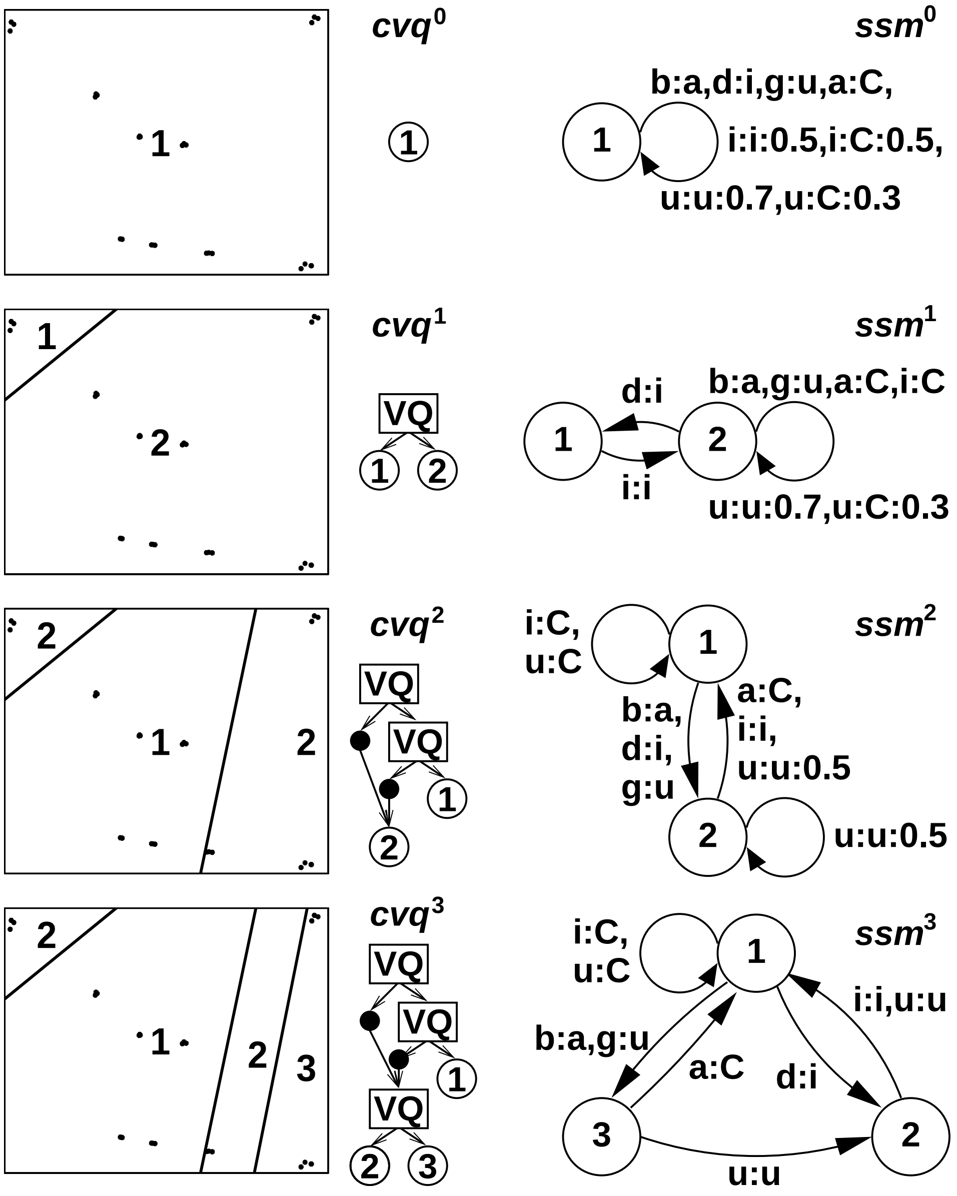


Figure 12.2: Extraction from RNN predicting in the "**badiiguuu**"-domain. The state space divisions (cf. Figure 12.1), CVQ graphs and SSMs are shown for the initial model and all subsequent iterations of `CrySSMEx`.

corresponds to the non-symbol generated by the RNN when it predicts a consonant, with no possibility of predicting the exact symbol due to the random ordering of substrings). For that reason, `collect_split_data` will select state vectors which the RNN occupied when it received **i** as input and label them according to the output label as determined by Λ_o. The CVQ is then split according to the selected data, resulting in cvq^1. The same procedure is repeated again with ssm^1, and an SSM of three SEs, of which two are UNDI-equivalent, is generated (not shown) This results in two merged nodes in cvq^2. As can be seen in the state space division, cvq^2 merges two (locally) disconnected subspaces. From both SEs of ssm^2, the output can now be deterministically predicted, but q_2 is still indeterministic since transitions from it over symbol **u** is ambiguously mapped to q_1 and q_2. Therefore, `collect_split_data` selects those RNN state vectors in Ω enumerated 2 by Λ_{cvq^2} from which a transition induced by symbol **u** was made, and labels them according to the Λ_{cvq^2}-enumeration of the subsequent state vector. After the split of cvq^2, `CrySSMEx` terminates since the resulting SSM is deterministic and will fully mimic the underlying RNN within Ω.

Note that there are some dead transitions in ssm^3, e.g., for symbol **g** in q_3. If the underlying RNN is fed a **g** while occupying a state in the corresponding subspace, it will certainly react in some manner, but since that event was not recorded in Ω, the resulting SSM does not model it. Also, the resulting SSM is not a model of the input source; for example, although not supported in Ω, ssm^3 models the outcome of infinite sequence of symbols **i** and **u**. This is due to the fact that `CrySSMEx` does not build a model of the domain, it builds a model of how the underlying system interacts with its domain without guarantees of generalization to situations outside Ω (again, a consequence of the closed world assumption).

12.2 An RNN trained on a context free language

The prediction of symbols in the context free language $\mathbf{a}^n\mathbf{b}^n$ is a challenging domain for RNNs that has been studied quite intensely (e.g. Wiles & Elman, 1995; Bodén & Wiles, 2000; Gers & Schmidhuber, 2001). In my study, `CrySSMEx` was used to analyse 100 successfully trained[2] SRNs (of one input node, two state nodes and one output node) to predict the predictable symbols of randomly ordered $\mathbf{a}^n\mathbf{b}^n$-strings $(1 \leq n \leq 10)$. Ω was here generated by exposing the RNN to exactly 200 $\mathbf{a}^n\mathbf{b}^n$-strings of each length $(1 \leq n \leq 10)$ in random order. For all 100 RNNs, extraction was successful, resulting in SSMs of eleven SEs. An example of an extracted machine, together with the CVQ-quantized state space, is shown in Figure 12.3.

The regular grid lattice quantizer of Giles, Miller, Chen, Chen and Sun (1992) was also tested, and it typically never found any SSM with the same behaviour as the RNN until the state space was divided into at least 40×40 grids. The number of SEs was then between 25 and 70. If the breadth first search of that paper is employed, the number of states becomes even higher (Jacobsson & Ziemke, 2003b) because then many states which would not have been visited when processing $\mathbf{a}^n\mathbf{b}^n$-strings are also included (cf. Appendix D).

In this domain, some problems for `CrySSMEx` are exposed. Firstly, although all extracted SSMs had the same $|Q|$, and all SSMs generated exactly the same outputs as the RNNs (within the sampled domain), actually two types of SSMs were extracted: 90 SSMs of one type and 10 of the other. This is probably due to different forms of dynamics of the underlying RNNs (Tonkes et al., 1998). If the deviance of the extracted SSMs from the RNN is plotted as error curves, they follow two exact and distinctly separate curves (see Figure 12.4). The extraction also took either nine or ten iterations depending on the underlying RNN. Clearly, `CrySSMEx` is sensitive to the internal properties of the RNN that give rise to these differences[3]. This may be a problem, but it may also be a key to a window of analysis of the dynamics of the underlying RNN.

A more serious problem arose when Ω was too small, e.g., with just ten occurrences of each string length, `CrySSMEx` could get stuck in loops where an ssm^i would be exactly equal to ssm^{i-n}, where $n \geq 1$. This was due to the merging and splitting of SEs cancelling each other over one or more iteration. The mechanisms behind these loops are not entirely clear and the issue definitely requires more targeted experiments. It seems, however, to be linked with some kind of data starvation

[2]Using a genetic algorithm, see Jacobsson and Ziemke (2003a) (or Appendix C) for further details and a more comprehensive list of references.

[3]Why exactly these results were obtained remains an open issue.

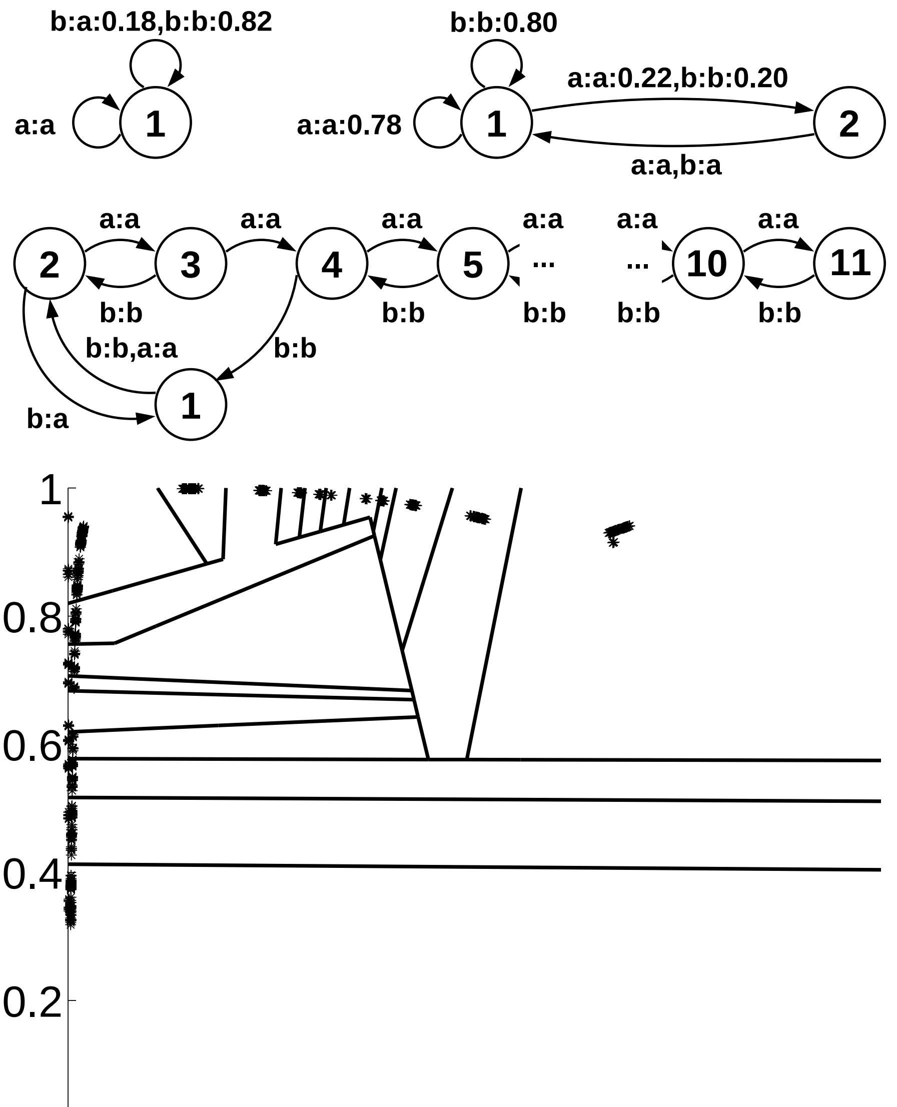


Figure 12.3: The first two machines (ssm^0 and ssm^1) and the last (ssm^{10}) in the sequence of machines extracted by `CrySSMEx` in the $\mathbf{a}^n\mathbf{b}^n$-domain. The state space of the RNN and its cvq^{10}-division of the state space is also shown below the machines. Note that some distant states belong to the same region, while, at the same time, some nearby states are divided due to the functionally driven quantization strategy employed in `CrySSMEx`. Some of the disjoint regions are also actually merged in the CVQ (which cannot be seen in the diagram).

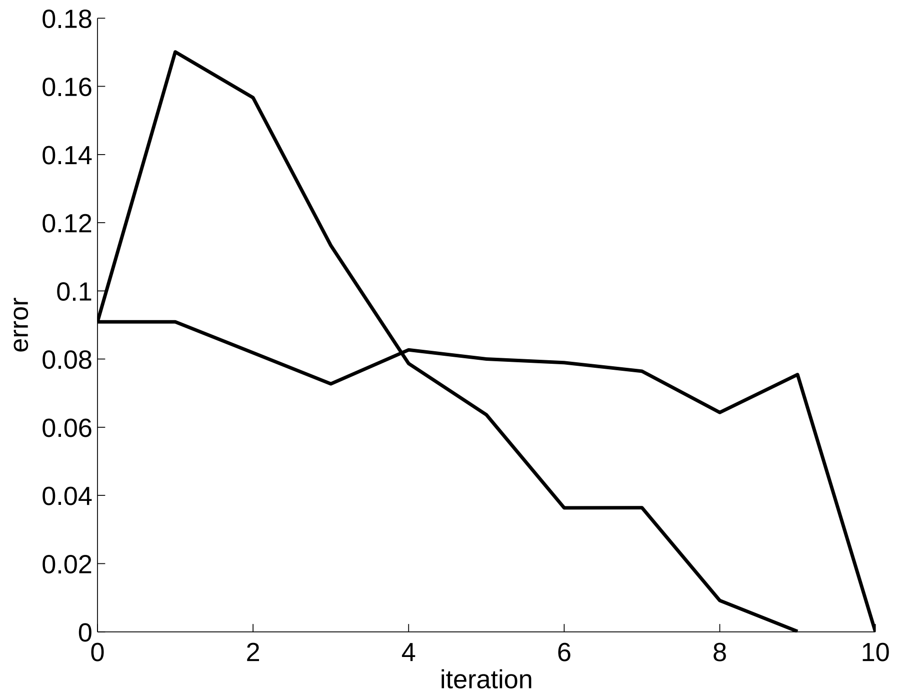


Figure 12.4: The error curves (where error here corresponds to the ratio of output symbols of the SSM not congruent with the RNN) of the extracted SSMs from 100 SRNs evolved to predict in the $\mathbf{a}^n\mathbf{b}^n$-domain. For 90 of the networks, the error curve followed the flatter line, and for the others it followed the steeper line. All 100 error curves are overlaid in the diagram and there are no deviances from the error curves since `CrySSMEx` is deterministic.

since it has only occurred for smaller Ωs. To circumvent this problem, `CrySSMEx` is now implemented to abort execution, by default, if a loop is encountered. Another, also implemented, option is to skip the merge completely if it should result in a loop. This approach is successful in that `CrySSMEx` terminates with a deterministic SSM equivalent with the RNN, but unfortunately with more SEs than the eleven otherwise extracted.

A third problem sometimes occurred when Ω was generated with longer $\mathbf{a}^n\mathbf{b}^n$-strings. In some cases, when the RNN generalized to longer strings perfectly, this posed no problem. In others, an erroneous prediction of the RNN was successfully modelled in the SSM, e.g., that it predicted an **a** prematurely if $n = 11$. However, in other instances, the temporal dependencies of the errors are quite complex, and the SSMs seem to grow indefinitely (without ever exceeding the size of Ω, of course). It is known that RNNs with weights in the vicinity of the correct solution have a seemingly chaotic error gradient distribuition which makes training using gradient descent difficult (e.g. Bodén & Wiles, 2000). Perhaps a chaotic RNN may also explain the difficulty for `CrySSMEx` (cf. Section 12.4).

12.3 A large RNN

An SRN of one input node, one output node and 10^3 state nodes (i.e. more than 10^6 weights) was created to test the feasibility of extracting rules from SDTDSs of high dimensionality. The weights were initiated in the interval $[-0.01, 0.01]$ and the network was then exposed to a sequence of 10^4 randomly ordered inputs ($I = \{(0), (1)\}$). The output quantizer used in this case gave three symbols, $+$, $-$ and $\mathbf{0}$, corresponding to whether activation of the output node increased, decreased or remained the same. The input symbols **a** and **b** corresponded to the binary activation of the single input unit. The continuous activation function of all nodes, $1/(1 + exp(-net))$, makes it typically impossible for the output to stabilize completely, i.e. there should be no need for symbol $\mathbf{0}$, but limits in machine precision made it necessary. This kind of network, with small random weights has been theoretically studied earlier and has been proven to implement definite memory machines (Hammer & Tiňo, 2003). This is, however, the first time a large scale network of this type has been studied using RNN-RE.

The extracted machine, with $|Q| = 19$, is illustrated in Figure 12.5. The machine emulated perfectly the behaviour of the SRN within Ω as "viewed" through Λ_o. It may be of interest to mention that the generated data required over 230MB of storage, yet `CrySSMEx` required only six iterations in the main loop to extract a machine of 19 states with the same apparent behaviour as the significantly larger RNN. The ease to extract from these networks is not surprising since the small weights force the network to have contracting transition maps, essentially causing the network to "forget" long term dependencies. It is, however, interesting to note that the CVQ does not seem to have any problems scaling up with respect to state space dimensionality.

Some more preliminary experiments were carried out with the same RNN architecture but with larger random weights. For smaller weights, $|Q|$ decreased, and for larger ones, the extraction may become impossible in the sense that the SSM

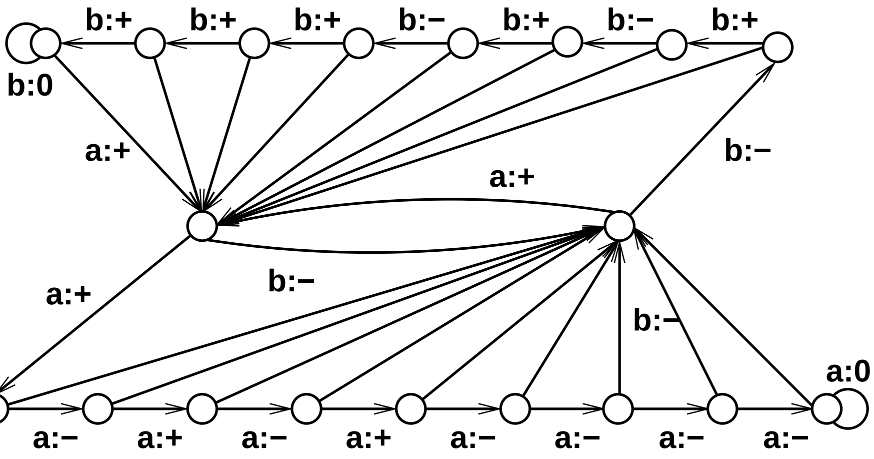


Figure 12.5: An SSM extracted from an RNN with 10^3 state nodes and random weights. To save space, numbering is omitted and repetitive transition labels are bundled.

size seemed to grow indefinitely. SSMs were of course still extracted, but no deterministic SSM were found within reasonable time. A high dimensional state space is not needed to make the extraction of deterministic SSMs impossible, however, as the following experiment demonstrates.

12.4 A chaotic system

To do rule extraction from a chaotic system may be considered unreasonable. If a system is chaotic it means that it will never repeat its trajectory in state space and that infinitesimal differences of two states will grow over time (Devaney, 1992). These properties make the system impossible to describe deterministically with a finite set of states. Any attempt to group two distinct SDTDS states into the same subspace will fail since their future trajectories will inevitably diverge if the system is chaotic.

It is, however, possible to use `CrySSMEx` to extract indeterministic SSMs from chaotic systems. An iterated quadratic map is used to demonstrate this :

$$s(t+1) = a \cdot s(t) \cdot \big(1 - s(t)\big) \tag{12.1}$$

The constant a, in the interval $[0, 4]$, determines whether the attractor of the system is a fixed point, cyclic or chaotic (Devaney, 1992). This system conforms with the SDTDS definition, but with $I = \emptyset$ and $O = \emptyset$. A similar experiment, in the same domain, was conducted by Crutchfield and Young (1990), but their approach was quite different from `CrySSMEx` because a fixed (unknown) translation from state space into a discrete set of observations was assumed. In `CrySSMEx`, it is precisely this translation that is the target for refinement.

The data was generated by running the system for 10^5 time steps, after an initial 10^5 unmonitored steps to let the system "settle in" on its attractor. The output symbols were, as in the last example, $+$, $-$ and $\mathbf{0}$ corresponding to whether the state increased, decreased or remained unchanged respectively[4]. This choice of output symbols is just one of many possible Λ_o, which is why it is part of the input parameters of `CrySSMEx`. Some readers might protest that this contradicts earlier claims in this thesis that `CrySSMEx` is parameter free. The subtle difference here is that although `CrySSMEx` requires the output quantization as a parameter, this quantization is for RNN applications typically defined *a priori* as a direct consequence of the symbolic domain of the RNN. In the above case, however, a number of output quantizations are conceivable.

The resulting machines are trivial when a is set so that the attractor is fixed or cyclic. If it is fixed, an SSM with one SE, and a transition generating symbol $\mathbf{0}$ are enough to describe the dynamics. If the system is cyclic, a finite set of SEs suffice to describe the system deterministically, e.g., if $a = 3.5$ (having a period four cycle) two SEs is enough, since the system, as "viewed" through Λ_0, generates the output sequence $\cdots + - + - \ldots$. If $a = 3.839$, the system has a 3-cycle attractor (Devaney, 1992) and generates the sequence $\cdots + + - + + - \ldots$ and consequently, the SSM had three SEs.

If a is chosen so that the system is chaotic, `CrySSMEx` will not terminate (at least not until the finite set of Ω is fully accounted for). But the extracted SSMs can nevertheless be argued to account for some of the dynamics of the system. To test the fidelity of the SSM, the extracted machines were initialized with the Λ_s-enumeration of an initial state (chosen within the attractor) of the underlying system, and both the SSM and the system were run in parallel until the SSM failed to predict the output symbol of the system. This tests also the generalization of

[4]A more common way to discretize this state space is to split the space into two parts with 0.5 as a delimiter (e.g. Crutchfield & Young, 1990). These experiments should therefore be repeated with a more standard approach to ensure comparability.

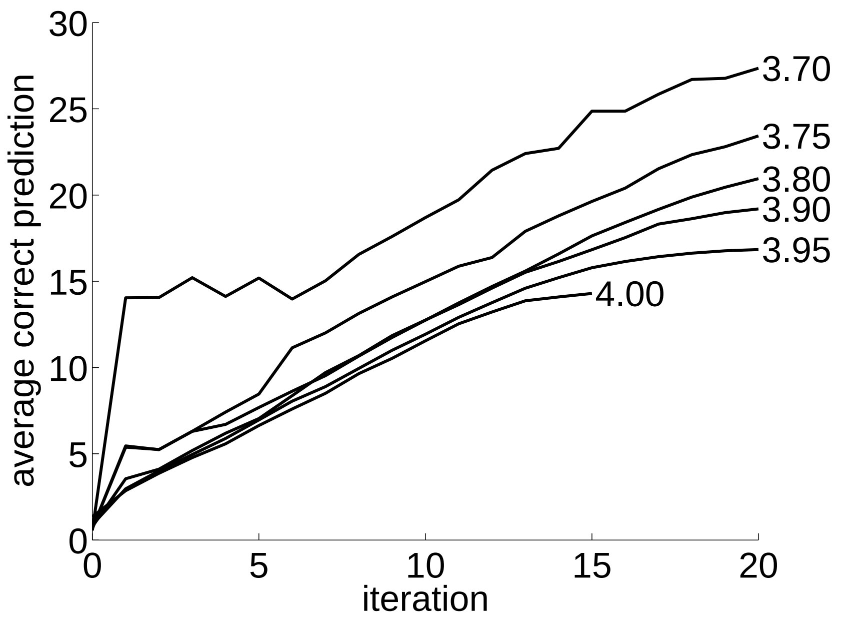


Figure 12.6: Some results of `CrySSMEx` modelling chaotic systems from extracted ssm^0 to ssm^{20}. The diagram illustrates the average number of correctly predicted symbols before the SSM failed to predict the output symbol of the system. When $a = 4.0$, the extraction was aborted at iteration 15 due to limited memory resources.

the SSM since previously unseen sequences are used in the test.

Six quadratic map systems, with $a = 3.7$, $a = 3.75$, $a = 3.8$, $a = 3.9$, $a = 3.95$ and $a = 4.0$ respectively, were analysed. The quality of extracted SSMs, in terms of the average time until SSM output deviates from the underlying system, typically increased for higher iterations of `CrySSMEx` (see Figure 12.6). The number of SEs grew exponentially for all systems, and grew faster for higher values of a (see Figure 12.7). The number of SEs in relation to the number of correctly predicted symbols reveals that the "cost", in terms of SSM size, for each correctly predicted symbol also increases exponentially (see Figure 12.8). It is however interesting to note that invested computational time clearly gives a gain in terms of SSM fidelity even when the underlying system is chaotic.

Other values of a were also tested, but if $a = 3.85$, for example, only three SEs were needed to predict the system indefinitely. Therefore, the seemingly monotonic relation between a and the number of SEs and prediction difficulty is merely an illusion.

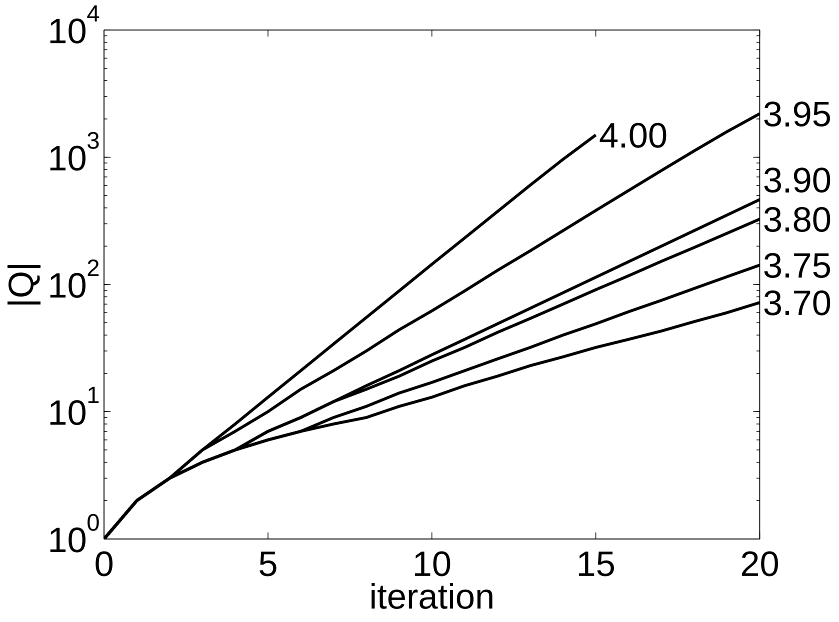


Figure 12.7: The the number of SEs of the SSMs extracted from the chaotic systems (cf. Figure 12.6).

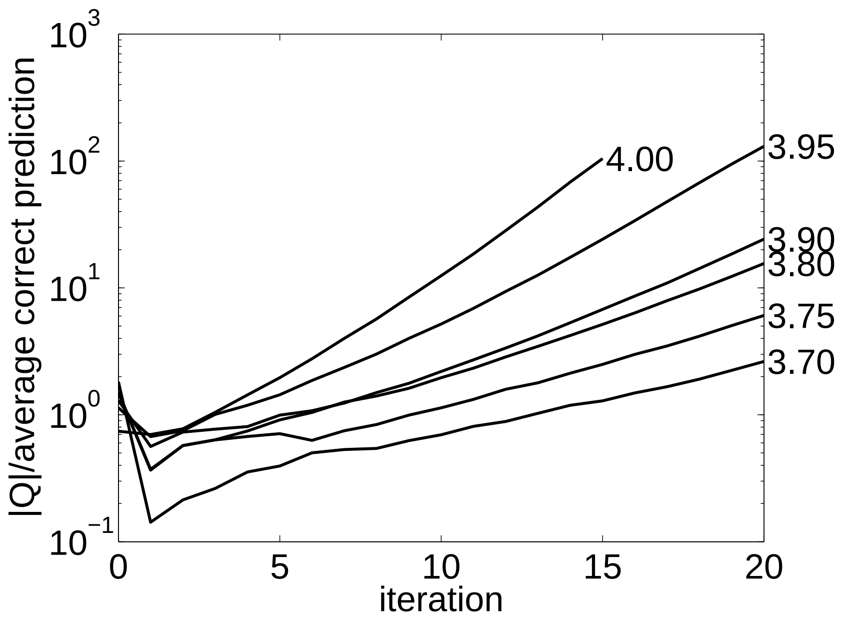


Figure 12.8: The diagram shows the number of states divided by the average number of correctly predicted symbols in the chaotic domain, thereby indicating the “cost” of predicting the system in terms of how many states each prediction needs (cf. Figures 12.6 and 12.7).

Chapter 13

Summary of Part II

13.1 New problem domains handled

The extraction of deterministic SSMs using `CrySSMEx` has now been shown to be possible for a number of challenging domains. The possibility of extracting stochastic SSMs from chaotic systems was also demonstrated. The domains on which earlier approaches have been tested were, almost exclusively, relatively simple binary regular grammars (cf. Part I or Jacobsson (2005)). Arguably, the context free domain, the high dimensional SRN and the chaotic system tested in this thesis, all constitute significantly more difficult problems.

13.2 New in `CrySSMEx`

Compared to the RNN-RE techniques presented in Part I, `CrySSMEx` introduces a number of novel features.

13.2.1 Integration of RNN-RE ingredients

As discussed in Chapter 8, there are three main differences between `CrySSMEx` and earlier approaches: the SSM, the CVQ and the integration of quantization, observation and minimization. These three ingredients make `CrySSMEx` more efficient than earlier algorithms simply because `CrySSMEx` performs a directed and deterministic search for a minimal quantization of the state space. Earlier approaches have relied on quantizers to find this minimal quantization without any information about the underlying dynamic system context of the state space.

13.2.2 Parameter freedom

Another difference to most earlier approaches is that `CrySSMEx` is parameter free[1] This is quite an advantage, since in the use of the algorithm as an analysis tool, the results are guaranteed not to be affected by the choice of parameters.

13.2.3 Deterministic extraction

`CrySSMEx` is deterministic and will result in the exact same extraction every time a data set is presented to it. Consequently, there is no need to run it more than once on the same data. The determinism stems from the determinism of the quantizer. This is not entirely novel, the regular lattice quantizer used by Giles et al. (cf. Section 5.2) is also deterministic, but the technique has problems scaling up and cannot be as precisely refined as the CVQ. Determinism is essentially the same as having no random seed parameter as input to the algorithm.

The determinism and parameter freedom should be essential if `CrySSMEx` is to be used as part of a larger system (as suggested in Part III of this thesis). As the component of a system, a parameter infested and indeterministic rule extractor would pass on these properties, in an amplified form, to the system of which it is part. If, for example, a rule extractor can give 10 possible separate rules as a result, a system of n such rule extractors would provide 10^n possible results.

13.2.4 Gradual anytime extraction

Another main feature is that the algorithm quickly creates an initial coarse stochastic model which it then gradually refines until a deterministic model is found (if possible). This "anytime rule extraction" possibility was considered by Craven and Shavlik (1999) to be an important aspect with respect to the scalability of the algorithms.

The advantage is that the more computational resources invested in running the

[1]Given that the output quantizer, Λ_o, is seen as derivable from the domain, which it always has been in the symbolic domains studied in the field of RNN-RE. It is not necessarily the case for SDTDSs in general, though.

algorithm, the more accurate the result will be. At the same time it will generate results on which can be based the decision of whether or not you want it to have more resources. For example, when very large SSMs were extracted from chaotic systems (Section 12.4), the diminishing performance gain of SSMs as predictor of the underlying system could be used as a termination criteria.

13.2.5 The handling of missing data

The algorithm can also handle missing data due to the substochastic nature of the extracted model. This is important since it uses the observation of a system to build models[2]. Observations that may, or may not, include all relevant aspects of the underlying system. For example, dead transitions are allowed, i.e. when the effect of an input symbol for a specific macrostate of the SDTDS is unknown, the corresponding transition in the SSM is undefined.

In effect, it means that the SSM partly models the ignorance that follows from Ω being a finite sample of the system. Furthermore, the entropies used to refine the CVQ are chosen so that the ignorance stemming from an imperfect state quantization can be redressed with no regards to dead transitions. A deeper discussion of this topic is found in Section 18.4.

13.2.6 An SSM is an SDTDS

Another distinguishing feature of `CrySSMEx` in comparison to most other RE algorithms is that the hypothesis space includes the system space, i.e. that the set of SSMs is a subset of the SDTDSs. I believe this is something quite unusual for RE algorithms (Andrews et al., 1995; Jacobsson, 2005). A finite state machine typically does not fit well into the framework of RNNs.

This relationship could be very fruitful. I have already used this feature for some verification of `CrySSMEx`; two SSMs, one SSM extracted from another deterministic SSM should generally be equivalent to each other. Further utilization of this relationship is suggested in Section 16.2.4, where it is proposed that the extraction of

[2]It remains to conduct experiments in different domains to evaluate the relative importance of this and to which extent dead transitions occur in extracted SSMs. Although not reported explicitly in the experiments of this thesis, however, dead transitions seem to be very common, which is especially obvious in the **badiiguuu**-domain (cf. Section 12.1).

SSMs from SSMs be automated within the rule extraction framework.

13.2.7 Hierarchically structured state space quantization

Last, but not least, the extraction results not only in a machine, but also in a hierarchically structured geometrical organization of the state space of the underlying system. This should be contrasted with the pure black box model of Vahed and Omlin (2004) where none of the internal dynamic state space is used in the extraction. Intuitively, the relation between the structure of the CVQ graph, the topology of SDTDS state space, and the SSM should contain important seeds for the further development of `CrySSMEx` and deeper analysis of the underlying system. Some such future prospects are discussed in Chapter 17.

Part III

From Rule Extraction to Machine Epistemology

Chapter 14

Introduction to Part III

In Parts I and II of this thesis the field of RNN-RE is surveyed in detail and a novel technique, `CrySSMEx`, is suggested and tested. In this final part of the thesis, potential future work is discussed in detail and new ambitious goals are suggested. Some of these goals may be immediately realizable and others may be considered more speculative. The suggestion of these goals is meant to serve as a catalyzer for the field, rather than suggesting a concrete agenda.

RNN-RE techniques previous to `CrySSMEx` are of a wide variety in terms of the *format* of the rules as well as *how* the rules are extracted. Common to all, however, is that their constituent parts (quantization, observation, generation and minimization) are not integrated to work in conjunction to solve the problem of rule extraction. The development of these techniques appears to have been focused on one or two of these constituents at a time. In `CrySSMEx`, however, the constituents are brought together for the first time and a number of novel features, discussed in Section 13.2, is the consequence of this. Furthermore, during the development of `CrySSMEx`, it has been assumed RNNs should not be considered the only system susceptible to the kind of automated analysis that RNN-RE constitutes. In this part of the thesis (Chapters 14–19), the specific assumptions underlying the development of `CrySSMEx`, the novel features of `CrySSMEx` and its possible improvements are further discussed, extrapolated and motivated. Hopefully, many of the issues are relevant also for techniques other than `CrySSMEx`, current or future ones.

The fields of RNN-RE and of RE in general, are for natural reasons traditionally associated with connectionism and neural computation. If, however, the underlying system is not restricted to neural networks only, then associations with other fields will become more obvious. This issue is referred to in the next chapter, where some of these related fields are identified and briefly surveyed.

In Chapter 16, some flaws of `CrySSMEx` are identified and possible enhancements suggested. These enhancements are primarily based on the novel SSM and CVQ, and should be fairly straightforward to implement and test. Such possible improvements lay the basis for Chapter 17, in which future challenges are suggested; challenges stemming from the novelties introduced by `CrySSMEx`, but which are also intended for the field as a whole. Several of these challenges are also connected to and potentially partially solved in some of the related fields referred to in Chapter 15.

Since the fidelity of extracted rules is the primary objective for `CrySSMEx`, rules may precisely describe the underlying system at the expense of comprehensibility. The virtue of rule extraction, however, does not have to be solely in terms of immediate comprehensibility of the rules. The extracted rules are models of simulated systems that may be employed just as scientists employ mathematical models to describe physical systems. In Chapter 18, this connection is further extrapolated and future goals are suggested for RNN-RE in lines of active learning and computational scientific discovery (Chapter 15).

Chapter 15

Future and Related Areas of Research

In this thesis I have followed the argument of Craven and Shavlik (1999), who claim that when developing a rule extractor one should not assume the underlying system is necessarily a neural network. The consequence is that the potential class of underlying systems will then encompass systems which are typically under study in other fields of research. This chapter briefly reviews a number of such fields and discuss how the RNN-RE-field may potentially benefit from studying some them more closely.

In some cases, techniques have been developed that could be used in RNN-RE contexts, and in others well developed fields could help by introducing a richer and more detailed terminology. As the title of this chapter implies, these related fields should be incorporated into RNN-RE research in the future.

15.1 Fields similar to RNN-RE

If we accept that RNN-RE techniques can be used on more types of systems than just RNNs, then I would argue RNN-RE should *not* be considered a field of neural computation, but rather a field of machine learning *applied to* models of neural computation (cf. Craven and Shavlik (1994)). The consequence is that related algorithms are not primarily found in the literature of neural computation (apart from the "classical" RNN-RE algorithms).

One important field, not immediately associated with machine learning, is control theory. Especially with regard to the system identification aspect of control theory, one suggested definition could as well apply to RE: "System identification deals with the problem of building mathematical models of dynamic systems based on observed data from the system" (Ljung (1999) p. 1). In control theory(e.g. S. Young & Garg, 1995; Marculescu, Marculescu & Pedram, 1996; Garg, Kumar & Marcus, 1999; Kumar & Garg, 2001), and especially for *discrete event systems*, similar problems as those for RNN-RE-algorithms have been dealt with for a long time. There are, however, some distinguishing features that separate `CrySSMEx` from algorithms of control theory, for example, the assumed full observability, discrete time and determinism of the underlying system.

In order to mature, however, the RNN-RE field needs to take into account the well developed theory of this related field. But once the connection to control theory is made, there is an abundance of other (some partly overlapping) fields that also needs to be taken into account:

- inductive logic programming (e.g. Muggleton & Raedt, 1994),
- grammar induction (e.g. Moore, 1956; Gold, 1967; Lang, 1992; de la Higuera, 2005),
- computational learning theory (e.g. Valiant, 1984; Angluin, 1987, 2004),
- symbolic dynamics (Crutchfield, 1994),
- computational scientific/mathematical discovery (e.g. Simon, 1995/96; Langley, 1998, 2000; Colton, Bundy & Walsh, 2000; Langley, Shrager & Saito, 2002; Langley, 2002),
- closed loop discovery (a.k.a. active learning) (e.g. MacKay, 1992; Cohn, Atlas & Ladner, 1994; Bryant, Muggleton, Page & Sternberg, 1999),
- belief revision (Friedman & Halpern, 1999),
- software testing (Bergadano & Gunetti, 1996),
- data mining, etc.

Taken together, these fields form an almost insurmountable abundance of literature (only very few examples are cited here). There are probably also other fields that are important to consider (cf. next section).

Some of the goals of these fields differ widely. For example, the goal for system identification is to better facilitate control of the underlying system whereas for software testing, it is to find errors. The terminology is also very diversified; the underlying systems may be called plants, machines and dynamic systems in control theory, an abstract teacher in computational learning theory, or an interactive user in inductive logic programming. The process is about system identification, model induction, scientific discovery or data mining, etc. The hypothesis space of the induced models also varies from differential equations, finite state machines, statements about systems and ad hoc representations of engineering problems.

After a brief review of the leading papers and books of the field, it becomes obvious from the lack of cross-referencing that the potential connections are not fully exploited. Yet all these fields have one thing in common with rule extraction; one of their central goals is to automatically induce models, conjectures, concepts and predictions based on observations.

The exact nature of the similarities and differences of these fields to each other and to RNN-RE is out of the scope of this thesis, however. But since the goals of these fields overlap with science in general, I would suggest that a natural way of bringing these fields closer together could be to build an encompassing theory by taking advantage of the deep insights philosophers of science have already provided us with (e.g. Simon, 1973; Williamson, 2004).

15.2 Theoretical connections

There are, at this point, no mathematical proofs that `CrySSMEx` will always provide the expected results, and clearly, some of the experiments demonstrate that it will not. A proof should distinguish the set of problems that can be solved by `CrySSMEx` from the ones that cannot. Therefore, a proof, or at least a deep theoretical analysis of the algorithm, is important. But such an analysis will arguably, since the parts are tightly integrated, require a merge of theories surrounding all `CrySSMEx`-constituents, combines ideas from areas such as:

- automata theory (Hopcroft & Ullman, 1979),
- stochastic machines (Paz, 1971),
- information theory (Cover & Thomas, 1990), and

- cluster analysis (Everitt et al., 2001).

There are, of course, also strong connections with the highly developed mathematical field of dynamic systems theory (Devaney, 1992), especially within the context of symbolic dynamics (Crutchfield, 1994). And the whole procedure of generating minimal algorithms (i.e. in this case SSMs) to explain a source of data (i.e. Ω) is of course related to algorithmic information theory (Chaitin, 1987).

The algorithmic complexity also remains an open issue. The experiments clearly show how evasive this issue is. For example, the analysis of an RNN of 10^3-dimensional state space resulted in a very modest SSM (simply because the weights were small enough, cf. Section 12.3) whereas chaotic one-dimensional autonomous systems generated enormous SSMs (Section 12.4). The SSM size for modelling chaotic systems will be bounded by $|\Omega|$, but such an answer is quite unsatisfactory since the algorithm should typically be terminated before it memorizes the entire data set. Given that the system is not chaotic, however, the computational complexity issue will be arguably more interesting, but at the same time very difficult to analyse since there are some arguably malicious factors to include, e.g., properties of γ (of the SDTDS) in combination with the selected input sequences.

Chapter 16

Possible `CrySSMEx` Improvements

During the development of `CrySSMEx`, many dead ends were encountered. The algorithm is presented in Part II just as it is (since it arguably works quite well). Therefore many details may seem to be "out of the blue". However, the algorithm has been implemented in such a way so that it will be easy for users to change some of these constituents in order to also try out the refuted alternatives[1]. The many dead ends and tried out alternatives to the specifics of `CrySSMEx` as it is presented in Part II, represent past issues in that these alternatives, based on preliminary experiments, have not been selected for use in `CrySSMEx`. But they also represent possible open issues since their assessment as dead ends is neither fully tested nor documented.

In this chapter a number of possible `CrySSMEx`-enhancements is presented. Many of these suggestions are already partly or fully implemented, but would require more attention and testing before their full integration with `CrySSMEx` is feasible.

16.1 Critical issues

Perhaps the most critical current issue concerns the theoretical understanding of the algorithm. There are at least two central decisions in the algorithm that have been made on heuristic grounds:

- NDI-equivalent SEs are now grouped using UNDI-equivalence (Definition 9.11). There is, however, more than one way to group NDI-equivalent SEs if

[1]See `cryssmex.sourceforge.net` for more details.

the relation is non-transitive. The chosen solution is only one possible, quite restrictive, way. For example, if SE pairs q_1 and q_2 and q_2 and q_3 are NDI-equivalent respectively while q_1 and q_3 are not, then hypothetical equivalence sets are $\{\{q_1\}, \{q_2\}, \{q_3\}\}\}$, $\{\{q_1, q_2\}, \{q_3\}\}\}$ or $\{\{q_1\}, \{q_2, q_3\}\}$, of which only the first, which results in no merge, is generated by UNDI-equivalence. The equivalence set $\{\{q_1, q_2, q_3\}\}$ would, however, not be reasonable since it groups non-equivalent q_1 and q_3.

- When data is collected in `collect_split_data` (Algorithm 11.1) a single input symbol is selected based on conditional entropy. It is, however, possible that another symbol should be selected, or that more than one symbol should be included. The selected symbol very strongly affects what the model vectors will be, and even if the seemingly most informative symbol is selected, the selection mechanism includes no heuristics about the underlying geometrical consequences of the decision. Moreover, it is not entirely selecting the the output symbol over next SE when labelling data is the optimal strategy.

The solution to both of these issues should involve more than just finding alternatives to UNDI-equivalence and entropy-based selection of input symbols. I suspect that it may involve systematic testing of merging and splitting in a breadth first search manner. This is due to the simple fact that the suitability of a split or merge operation may not become clear until after actually performing the operation and testing the effect of using the quantizer to generate a new SSM. It may even be necessary to split one SE at a time instead of, as now, splitting a number of SEs simultaneously.

In this respect, it is perhaps reasonable to consider `CrySSMEx` a promising first step towards a more generic approach, rather than a final solution to the problem of RNN-RE. Moreover, the fact that the algorithm performs quite well on complex domains while there are still obvious ways to improve it can also be considered quite valuable. It could be regarded a motivation that would justify an effort of mathematically proving the correctness of the algorithm, or at least critical parts of it, such as `NDI_equivalent` (Algorithm 9.1).

16.2 SSM refinement and analysis

The SSM definition and operations on SSMs barely scratch the surface of what can possibly be accomplished with these kinds of dynamic stochastic models. Relations to other similar models, such as Bayesian Networks and Hidden Markov Modelsmay be identified and utilized for further analysis and refinement. Some refinements and possible analyses of SSMs that were encountered and sometimes implemented during the development of `CrySSMEx`, are briefly presented in the following subsections.

16.2.1 Moore SSMs

The algorithm is also implemented (`cryssmex.sourceforge.net`) with the possibility of extraction SSMs on Moore format rather than Mealy (cf. Figure 3.2) as presented in Part II. In a Moore SSM, the output is determined from a SE distribution rather than in the transition between state distributions. The experiments conducted in this thesis could therefore be repeated with extraction of Moore SSMs instead. Since the whole description of SSMs and the algorithms would have to be translated and repeated in Moore format, it is left for future work, however.

16.2.2 Modal logic possibility

The testing of equivalence of state elements of SSMs currently returns only in true or false (cf. Algorithm 9.1) depending on whether the two elements are NDI-equivalent or not. The NDI-equivalence is however more adequately answered with “true” (**T**), “false” (**F**) or “possible” (**P**). The answer **P** then refers to the cases where full equivalence could not be asserted due to dead transitions. The use of modal logic could potentially help in generating equivalence sets since two SEs that are truly equivalent must be merged, whereas possibly equivalent SEs can possibly be merged. It could thus be easier to systematically test all combinations of possible merges (cf. Section 16.1).

16.2.3 SE relations

When `CrySSMEx` currently extracts a sequence of SSMs, the SEs of an ssm^i is not traced from its predecessor ssm^{i-1}. There is however a potential relation of every

SE in ssm^i to the SEs in ssm^{i-1}, e.g., relations as "is-split-from", "is-merged-from" or "is-not-split-from". Since these relations can be formed between any succeeding SSMs they could be used to generate something that resembles a *genealogy of SEs* (cf. Figure 16.1). Such a genealogy could be used in many ways, e.g., to define the *integrity* of SEs as the number of `CrySSMEx` iterations (of the main loop in Algorithm 11.2) in which it has not been part of any merge or split operation. In Figure 16.1, for example, the q_4 of ssm^4 has an integrity of 2 whereas q_5 has an integrity of 1. Another example is the *purity* of SEs in terms of whether or not it is the result of any merges. SEs $\{q_4, q_5, q_6, q_7\}$ of ssm^4 in Figure 16.1, are examples of pure SEs.

A genealogy can also be used to define the "relatedness" of SEs. Since the genealogy of SSM state elements involves multiple parents, the relatedness can be defined in several ways, for example, by counting how many `CrySSMEx`-iterations since two states belonged together in one and the same SE. In Figure 16.1, for example, q_6 and q_7 are then more related than q_5 and q_6. Relatedness could also be based on the number of parents in common in the last SSM. For example, q_1 and q_2 of ssm^4 has q_1 and q_2 of ssm^3 in common as parents but q_2 and q_3 (of ssm^4) also have q_3 (of ssm^3) in common. The CVQ graph could also be used in a similar way, of course, but the multiple splits due to the complete split (see Definition 10.8) may obscure the SE relations. The relatedness of two SEs could possibly be used as heuristics when forming equivalence sets to be used for merging by prioritizing the merging of related SEs over relatively unrelated SEs (cf. the discussion in Section 16.1).

Derived properties can possibly be quite informative. Properties such as relatedness, integrity, purity and others defined over the genealogy graph, could potentially be fruitful windows into the dynamics of the underlying system or the input sequence with which the system interacts. Perhaps the prevalence of pure SEs is, for example, correlated with a certain form of dynamics in the underlying system? Or, perhaps, states of lower integrity are typically more sensitive to noise? If nothing else, issues such as these could quite simply be investigated further.

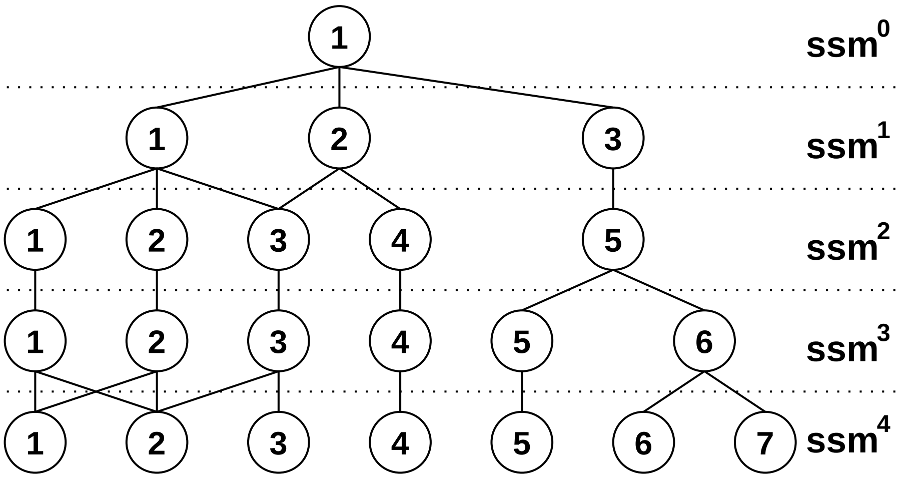


Figure 16.1: A genealogy of SSM SEs. The SSMs are shown without transitions and q_i is simply written as i. Since SEs are created from one or more SEs of the preceding SSM, through splitting and merging, there is a rich variety of possible genealogy relations and properties. For clarity, the transitions within the machines are not shown.

16.2.4 Always deterministic SSMs

Since an SSM strictly speaking *is* an SDTDS, it is possible to extract SSMs from other SSMs. One way to utilize this could be to automatically extract deterministic descriptions of the sequence of SSMs extracted by `CrySSMEx`. The sequence of (mostly) nondeterministic SSMs generated by `CrySSMEx`, $ssm^0 \dots ssm^n$, could then give rise to a sequence of deterministic SSMs, $\widehat{ssm}^0 \dots \widehat{ssm}^n$. To generate $\widehat{ssm}^i$ from ssm^i, the inputs of Ω is quantized by Λ_i and fed as symbolic input to ssm^i (using normalized parsing) and the substochastic state vectors are read as the state of the SSM (viewed as an SDTDS). The resulting data sequence, Ω_{ssm^i} is then used as input to `CrySSMEx`. The suggested procedure is depicted in Figure 16.2.

The output quantizer in this case could be chosen to be maximum likelihood, i.e. the output symbol distribution of ssm^i is quantized such that the symbol with the highest probability "wins". `CrySSMEx` should then generate an $\widehat{ssm}^i$ semi-equivalent to ssm^i over Ω. It will not be fully equivalent, however, since ssm^i needs to be initialized prior to generating Ω_{ssm^i} and this initialization may, or may not, correspond to the internal activation of the underlying system of ssm^i under its corresponding initialization.

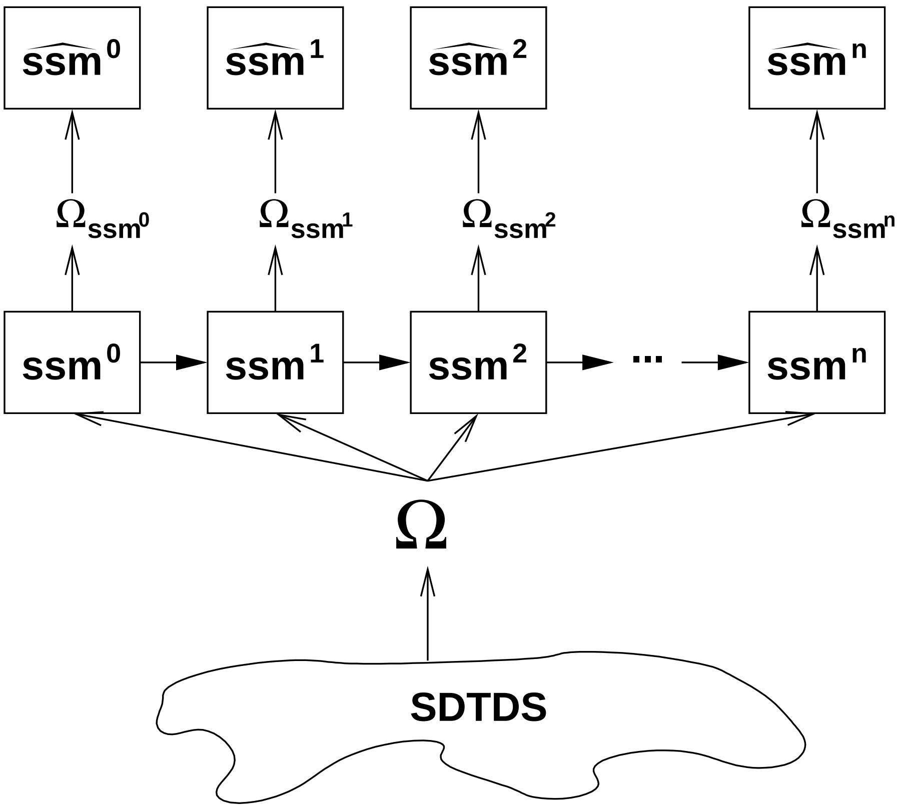


Figure 16.2: The possibility to derive SSMs from SSMs could be used to extract deterministic SSMs from the nondeterministic intermediate SSMs during extraction. From every ssm^i, a transition event set Ω_{ssm^i} is created (based on input events in Ω) and a deterministic SSM $\widehat{ssm}^i$ is extracted from it.

16.2.5 Additional information

The SSM is in its current form quite free from information regarding the underlying system since only the conditional probabilities are stored. One could, however, easily imagine that it could be desirable to add some more information to the SSM. For example, each SE in the SSM corresponds to an enumerated region in the SDTDS which in Ω has a certain visitation frequency. Therefore each SE can be associated with a frequency. This information could be added to the SSM which then would then convey information about, for example, which situations have only very rarely occurred in Ω. Such information could be vital to refine Ω (cf. discussion in Section 18.4).

Another possibility, if the underlying SDTDS, for example, is an RNN trained within a domain, is that the information of the error frequency of the RNN could also be stored for each SE or transition. The SSM would then hold information regarding the situations in which the RNN produces errors. This information could be used to pinpoint the error of the RNN in order to, for example, refine the training set (cf. discussion in Section 17.6.2).

The average state vector associated with each SE could also be saved, for example. SEs then could be associated with their corresponding "typical" SDTDS states. Other contextual information could also be added to accentuate certain aspects of the underlying SDTDS, e.g., if the state of the SDTDS has a temperature variable, the average temperature could be highlighted for each SE.

16.2.6 SSM comparisons

How big is the difference between two SSMs? This question may not have one unique answer. There are a number of possible ways to define difference measures. One could, for example, base it on relative entropy of the output distributions of the two systems[2] (a.k.a. Kullback-Liebler divergence) (Cover & Thomas, 1990). Relative entropy is used as a kind of distance measure between probability distributions. By using relative entropy over the output distributions of two SSMs one could measure the divergence of the machines from each other under various sequences of input symbols. The question is, which input sequences? Furthermore, how should the results be weighed if summed up for many input sequences?

The issue is problematic and may not have any satisfactory answers. But, if the distance measure between SSMs has a specified purpose, then possibly some domain specific way of measuring the distance could be appropriate. For example, if the underlying RNNs are used as predictors of critical events in a plant, the difference between them could be defined in terms of the difference in the situations for which they predict the critical event.

For example, if the distance is measured for the purpose of guaranteeing that RNNs in a large set exhibit diversity (e.g., if they are to be used as an ensemble (Krogh & Vedelsby, 1995)), then perhaps some fairly simple analytic procedure could be adequate.

Another, perhaps more generic, method is to extract the difference between SSMs by using `CrySSMEx` as suggested in Section 17.6.1.

[2]Many thanks to André Grüning for all the inspiring discussions regarding ways to do this.

16.3 CVQ refinement

16.3.1 Refined training

One aspect which has received little attention in this thesis is the CVQ graph and the CVQ as a quantization function (Chapter 10). The model vectors of the VQ-nodes are, for example, selected simply as the average of the data vectors without any further refinement. This, and other unoptimized aspects of the CVQ may result in CVQ graphs that are larger than necessary to quantize the state space of the SDTDS properly. The CVQ can be optimized in many ways; one is to refine the model vectors using LVQ- or SOM-techniques (Kohonen, 1995). This kind of optimization would be conducted in the learning phase of VQ-node creation. The currently suggested method of just choosing the average of each class of vectors (cf. Section 10.3) was actually originally chosen because it was the *least* optimal in order to show that `CrySSMEx` was not sensitive to the quality of the quantization made through the model vectors. To my surprise, replacing a fairly sophisticated resource allocating vector quantization training algorithm with a straightforward initialization, using averages, did not reduce the quality of the extracted machines at all. If model vectors were selected more carefully, however, the CVQ graph could in many cases be reduced.

16.3.2 Post-training refinements

Another improvement, given a trained CVQ-graph which corresponds to quantization function Λ_{cvq}, would be to attempt to generate a smaller CVQ graph with a quantization function $\Lambda_{cvq'}$ equivalent to Λ_{cvq}. I would like to separate two levels of such CVQ graph reduction: *analytical* and *empirical*. The analytical CVQ graph reduction would require the postcondition $\Lambda_{cvq}(\vec{v}) = \Lambda_{cvq'}(\vec{v})$ for all $\vec{v}$ in domain of quantizer. The empirical, however, needs only postcondition $\Lambda_{cvq}(\vec{v}) = \Lambda_{cvq'}(\vec{v})$ for all $\vec{v}$ that are sampled state vectors in Ω (or any other sample space of interest). The empirical compression can be likened to a lossy compression, which should be more efficient, but less accurate, than the analytical approach.

16.3.3 Further recursiveness

Apart from optimizing the CVQ graph, one could develop it further in other directions. The CVQ graph is a recursive organization of vector quantizers in which one or more of these are used to generate an enumeration of data vectors. But there is no reason why this recursive organization should be over *vector quantizers* explicitly. The VQ-nodes could instead contain *any* kind of quantizer (cf. the abstract description of quantizers in Definition 9.4). Let us call these nodes Quantization-nodes (Q-nodes) and the graph corresponding to the CVQ graph a Crystalline Quantizer graph (CQ graph). The Q-nodes could, for example, consist of decision trees, feed-forward artificial neural networks or simple if-then-else-functions that replaces the VQ in Equation 10.3. An interesting possible Q-node would have a quantizer which in itself is a CVQ- or CQ-graph.

Another recursive possibility would be to let a quantizer consist of several conjugated quantizers, i.e. multiple quantizers applied simultaneously and where the resulting quantization is a vector of enumerations in which each vector corresponds to the result of one quantizer. These vectors will then in turn be enumerated to generate a single enumeration in the end results. For example, if quantizers $\Lambda_1 \ldots \Lambda_m$ are used to quantize a vector $\vec{v}$, the result would be a vector $(n_1 \ldots n_m)$ where $n_i = \Lambda_i(\vec{v})$. This vector is then stored and enumerated in a look-up table of all such encountered vectors.

The reason for suggesting conjugated quantizers is that they could be used when splitting an SSM state element using multiple symbols (cf. discussion in Section 16.1). Each split-symbol would generate one CVQ (or CQ) which could then be conjugated.

16.3.4 Intelligible quantizers

The intelligibility of CVQs could be improved considerably. The CVQ can be visualized using Voronoi-diagrams (as in Figure 12.3 on page 93) if it is used only for two-dimensional vector spaces. But for higher-dimensional spaces, the underlying geometrical properties of the CVQ-quantized space becomes intractable. If, however, a CQ is based on quantizers with simple textual explanations, e.g., "if $v_4 > 0.56$ then return 1 else return 2", then the textual interpretation of each such quantizer could possibly be used to generate a textual interpretation of the whole CQ graph. Such textual descriptions can, for example, be generated from decision trees as nested if-then-else-statements. In the CQ case, however, it may become more complex as merged nodes allow several possible paths to the same leaf-node. Each leaf-node can therefore correspond to more than one explanation. For example, the leaf node n_6 of Figure 10.2 (page 79) could be explained as "if $\vec{m}_1$ of n_0 is the winner then return $ID = 3$" or as "if $\vec{m}_3$ of n_0 is the winner and $\vec{m}_1$ of n_3 the winner then return $ID = 3$". The concept of winning model vectors is not entirely easy to grasp since they are potentially multidimensional and the resulting quantization is a result of testing all these model vectors against the data-vector. Therefore, vector quantization may not be the way to generate "readable" quantizers, whereas CQs tailor-made for this purpose could.

If readable CQs can be generated, another form of CVQ optimization could therefore be to do an empirical translation (cf. the empirical graph reduction discussed in Section 16.3.2) from the "unreadable" CVQ to a more readable CQ.

Chapter 17

Possible `CrySSMEx` Challenges

In this chapter a number of possible challenges for the further development of `CrySSMEx` are suggested. This chapter is built on the previous one since some of the presented challenges would require significant improvements of `CrySSMEx`. However, the future work suggested in this chapter is more speculative and also intended as potential challenges for more RNN-RE algorithms than only `CrySSMEx`.

17.1 More interesting SDTDSs

The most obvious possibility of future work is the application of `CrySSMEx` to more problems. In this thesis, only a handful of systems are analysed using `CrySSMEx`. The applicability of `CrySSMEx` to more complex models remains an open issue. The SDTDS definition is broad enough to apply to a large number of interesting neural network models. The experiments also demonstrate that high dimensionality of the state space poses no immediate limits. Hence, it could be possible to extract rules from the otherwise hard-to-analyse networks such as, for example, Echo State Networks (ESN) (Jaeger, 2003; Jaeger & Haas, 2004). The ESNs are a rare species of RNNs because a large number of state nodes are interconnected in a random fashion (and not trained). These state nodes then exhibit a wide range of input-driven dynamic behaviours. The output neurons tap into these dynamics by using simple linear optimization techniques. In a network of several hundred state nodes, some may have the dynamics required to solve the task. Although remarkably simple, this architecture has proven to be quite successful. But it should, however be notoriously difficult to analyse since it has a highly multidimensional state space.

Another promising architecture is the Long Short-Term Memory architecture (LSTM) (Hochreiter & Schmidhuber, 1997). The LSTM has been used on the $\mathbf{a}^n\mathbf{b}^n$-domain with remarkably better results than, for example, simple recurrent networks (Schmidhuber, Gers & Eck, 2002). Although the testing procedure is somewhat unclear and can be questioned (Jacobsson & Ziemke, 2003a) (or perhaps because of this), the results invite further analysis: (a) partly to gain deeper understanding of how LSTMs represent the $\mathbf{a}^n\mathbf{b}^n$-problem as compared to SRNs, but also, (b) to see how `CrySSMEx` scales up to very deep grammar structures. The $\mathbf{a}^n\mathbf{b}^n$-experiments of Section 12.2 show extraction from RNNs with a deeper structure than to which any previous RNN-RE technique has been applied, but analysis of LSTM would increase the depth even further.

The most ambitious network architectures from which to extract rules, however, may be the ones that build on models of biological neurons. RNN-RE has so far been restricted to simple mathematically oriented RNNs rather than more biologically realistic models, possibly due to the problem of scaling up. The promising `CrySSMEx` results, however, imply that the analysis of some of these systems may be possible (as long as they fit into the SDTDS definition).

Apart from various neural network architectures, there is an abundance of other simulated models to choose from. As long as a system complies with the SDTDS definition, then it has the potential of being analysed by `CrySSMEx`. If some of the requirements of the SDTDS can also be alleviated bit by bit (cf. Section 17.9), the descendants of `CrySSMEx` could be applied to an even broader range of systems.

17.2 An SSM Query-language

The issue of comprehensibility has often been the central driving force in rule extraction research (Andrews et al., 1995; Tickle et al., 1998). Rule extraction has been seen as a means of helping researchers better understand their networks and therefore the comprehensibility of the rules is important. I would like to challenge this view, however. I find it problematic if comprehensibility becomes a required post-condition of rules not yet extracted. What if the underlying system cannot be described by a small set of rules? Should this impinge on the fidelity of the rules? Instead of establishing the sacrifice of fidelity in favour of comprehensibility as a basis for the algorithm, I would argue for the extraction being optimized for fidelity, but that the rules should be at least partly comprehensible through the possibility of *querying* them (cf. Chapter 18).

For `CrySSMEx` this preference for fidelity over comprehensibility, means that a large SSM, which more accurately describes a system than a smaller one, will be preferred. But if the larger SSM could be post-processed such that smaller-SSM "views" or "derivatives" of the underlying system can be extracted from it, then comprehensibility would be partly re-acquired. For example, in the $\mathbf{a}^n\mathbf{b}^n$-SSM of Figure 12.3, one could ask the question: "What sequences of inputs will generate a prediction of an **a** after the input of a **b**?" and receive a number of arguably comprehensible examples.

The possible number of queries for a single SSM is unbounded, e.g.:

- "What SEs precede **a** as output?"
- "What input sequences will take the SSM from q_1 to q_2 without passing q_3?"
- "What are the 100 alphabetically first sequences that are not modelled in the SSM?"
- "What is the shortest input sequence that will exhaust the SSM SE distribution from a uniform SE distribution?"
- "What is the probability of generating output sequence **ccdd** given given input sequence **aabb** and an initial uniform SE distribution?"
- "What is the probability that the $p(q_3) > 0.9$ five time steps ago, given the input history **aabab** if currently $p(x_5) = 1.0$?"
- etc.

A fairly complex query language would have to be developed to handle these questions. The query language (both in the question and answer spaces) would likely need to represent and handle elements of the following kind of data:

- probabilities,
- probability distributions,
- entropies,

- ranges of probabilities, entropies etc.,
- symbols,
- symbol sequences,
- sequences of probability distributions,
- sets of symbols, states, probabilities etc.,
- SSMs,
- quantizers,
- CVQs,
- vectors,
- SDTDS data,
- Modal logic (cf. Section 16.2.2)
- SSM genealogy (cf. Section 16.2.3)
- etc.

Moreover, some of the additional information suggested in Section 16.2.5 could be added, e.g., visitation frequency, error rates, or *ad hoc* information. Multiple SSMs and relations based on SE genealogy could also be part of the queries (cf. Section 16.2.3) A brief outline of the potential of such a language follows, but the examples are kept in English, as the syntax and semantics of such a query language would require much additional work.

17.2.1 Querying regarding SSM ignorance

If, for example, the information about how frequently states and state transitions are in Ω is added to the SSM (cf. Section 16.2.5), then questions regarding "SSM ignorance" could be asked. For example:

- "Disregarding dead transitions, what are the 20 transitions least frequently used?"
- "Generate a sequence of inputs that according to the latest available SSM will generate observations of the SDTDS that, if added to Ω, should make all SEs nearly equally frequently visited."
- "Remove all SEs that are visited only 10 times or less in Ω!"
- "Generate an input sequence that maximizes certainty of SSM knowledge of the underlying system!"

The "ignorance" of the SSM refers simply to what aspects of the underlying system it has little or no information about. Questions such as those above could therefore be used to resample a new Ω in order to reduce the overall ignorance of the SSM. The usefulness of such questions will be highlighted in Section 18.4.

17.2.2 Querying to achieve control

The SSMs are models of the underlying systems that could potentially be used to control the underlying SDTDS. The SSM could be interacted with simultaneously, as with the underlying SDTDS, using the same inputs sequences. The difference is that the SSM can be used for planning the estimated outcome of the underlying system. One could therefore consider questions such as:

- "What is the best sequence of input symbols to generate an output symbol **b** with highest possible likelihood within 10 time steps?"
- "Take system into one of SEs q_1 or q_2 as soon as possible!"
- "Get the system to generate an output sequence which in the following (additional) SSM would generate symbol **1** after the last input symbol."[1]

If `CrySSMEx` is used to extract a model of the environment of a robot, for example, then these kinds of queries could be the foundation for the robot to take advantage of its induced SSM model as a basis for planning.

17.2.3 SSM abstraction through queries

To increase the comprehensibility of SSMs, an SSM could be abstracted through queries that result in derivatives of the original SSM in which certain aspects are accentuated and others ignored. For example, if a large SSM predicts letters, a question resulting in a more abstract SSM would be: "Generate an SSM in which vowels are separated from dconsonants in the output, but the individuality of output symbols is otherwise ignored.", i.e. all vowels are grouped into one single output symbol and vice versa for consonants.

One way to resolve queries is to do it analytically, i.e. by processing an existing SSM based on the query. Another way is to extract one SSM from the other by

[1]The additional SSM is here thought of as a description of acceptable (i.e. grammatical) input strings.

using `CrySSMEx` and a more abstract output quantizer with fewer possible symbols (cf. discussion of how SSMs are SDTDSs in Section 16.2.4).

If queries make abstraction of the kind described above possible, then it could feasibly automate the abstraction process further. For example, "Generate a set of the smallest possible SSMs that explain at least 90% of the output symbols in Ω!".

The abstraction of SSMs can thus be viewed as a lossy compression of the underlying SSMs. In the same way, the SSM is a lossy compression of the underlying SDTDS.

17.3 User goals

As suggested in Chapter 7 (cf. Figures 7.1 and 7.2), the possibility for the user to choose between fidelity, efficiency or comprehensibility in the manner of a parameter of the RNN-RE algorithm should be considered a much desired feature.

In `CrySSMEx`, the extraction by default goes from small models to larger. The comprehensibility thereby decreases[2], whereas the fidelity increases. The fidelity versus efficiency tradeoff is thus trivially chosen by the user in the possibility to abort `CrySSMEx` at any time. More time means higher fidelity and less comprehensibility.

The query language, automated abstraction and automated extraction of deterministic SSMs during extraction could, however, help to boost comprehensibility at the cost of efficiency. If the SSMs can be queried as suggested above, there will, however, not necessarily be any tradeoff between comprehensibility and fidelity. Comprehensible answers will be generated from a model of high fidelity and these answers (possibly SSMs themselves) may coexist with the original high-fidelity SSM.

With regard to accuracy (Section 4.2.4), however, I would argue that this is an ill-defined measurement. If there is a domain in which the network has been trained, it is perhaps likely that a small SSM would generalize better than a larger one (according to the principle of Occam's razor). There will also be a fidelity-accuracy tradeoff (Zhou, 2004). The accuracy of rules is, nevertheless, a feature that depends on the definition of generalization ability which is not always obvious

[2]Supposing that we make the crude assumption that larger rules are necessarily less comprehensible.

in these temporal domains (Jacobsson & Ziemke, 2003a) (also in Appendix C). It is, for example, not obvious whether the rules should generalize towards only longer $\mathbf{a}^n\mathbf{b}^n$-strings or perhaps even balancing parenthesis languages (i.e. if you replace **a** with '(' and **b** with ')' and enforce that the parenthesis are balanced at the end of the string).

The potential fidelity-accuracy tradeoff could nevertheless be exploited as suggested in the following section.

17.4 Robust hierarchical stochastic degree-of-belief model

Since there are reasons to believe that smaller SSMs should be more adequate for generalizing to unseen examples, according to the Occam's razor argument, the `CrySSMEx`-iterations should progressively generate less and less accurate SSMs simply because they become gradually larger. They will, however, also be of increasingly higher fidelity.

In other words, ssm^n has a more robust stochastic information regarding the underlying system than has ssm^{n+m} (if $m > 0$). The first SSM, ssm^0, for example, has an extremely robust model of the underlying system which is almost guaranteed to be correct since it can only produce very vague answers regarding the actual state (and output) of the underlying system. Later SSMs will have higher probabilities with regards to generalizing incorrectly about the actual state of the underlying system. In other terms, the more developed SSMs become brittle since they will generate more specific output predictions. For example, ssm^0 will alway be correct regarding the state of the underlying SDTDS since all microstates are modelled as one macrostate. It will also give a vague probability distribution over the output symbols. After a few `CrySSMEx`-iterations (cf. Algorithm 11.2), however, the SSM will be described as consisting of several SEs and will attempt predicting the macrostate of the underlying system, a prediction that may be erroneous if the SSM does not generalize correctly.

The SSMs are thus progressing from *robustness* to *brittleness*[3]. Moreover, the

[3]They are also progressing from *unfalsifiable* SSMs towards more easily *falsifiable* SSMs, i.e.

CVQs will also divide the state space into increasingly smaller regions. If one considers the possibility of noise in the state of the system, then the smaller the region, the higher the possibility of misclassifying the state space. In other words, the CVQ also progresses from robustness to brittleness during the extraction, or put another way, when `CrySSMEx` extracts a sequence of SSMs from Ω, they will cover a spectrum from robust-but-stochastic to brittle-but-high-fidelity models.

Consider a situation where an SDTDS has been analysed using `CrySSMEx`, then the SSM and CVQ can be used to continuously monitor and predict the system (which we can assume to be noisy). In other words, the SSM is used for the prediction, whereas the CVQ is used to justify and adjust the SSM state as a correct stochastic representation of the SDTDS state. In this situation, the robust-brittle-spectrum could be very fruitful. The whole range of SSMs and CVQs can be run simultaneously on the same input for a robust and high-fidelity prediction of the system.

Potentially, if correctly implemented, one could achieve the best of both worlds. The spectrum could potentially be exploited by using the genealogy of SEs (cf. Section 16.2.3). An SSM SE or CVQ-observation of the SDTDS state will provide information regarding the state of a more brittle SSM. For example, if the CVQ cvq^n observes in the SDTDS that the SE (of the corresponding SSM) should be q_i this can be used to infer the states of subsequent, more brittle, SSMs through genealogical relations (together with conditional probabilities of how SEs of one SSM explain SEs of other SSMs).

17.5 Higher order extraction

As mentioned in Section 13.2, `CrySSMEx` differs from earlier RNN-RE-algorithms by adapting the state space quantizer to generate better rules. This, together with the fact that the adapted quantizer has a graph-like structure, could possibly be exploited for the extraction of higher-order rules. Specifically, the extraction of context-free grammars is a possibility. A finite state recognizer of a context-free grammar requires an external memory in the form of a stack. For example, the

the possibility of making observations which contradict the SSM increases with its fidelity. This is an important feature and its further explotation is suggested in Section 18.6.

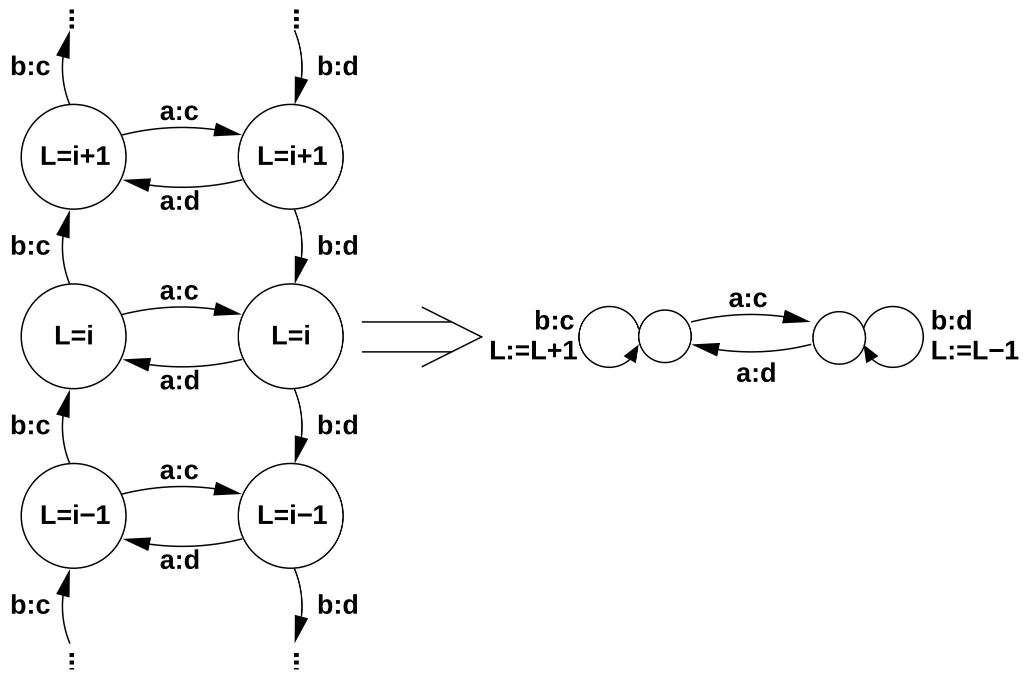


Figure 17.1: A partial example of how the level of CVQ nodes (cf. Definition 10.6) could be used to infer context free grammars. The SE-pairs of each level are equivalent of all other similar pairs in that symbols **a** and **b** trigger the same response in the output as well as the same relative changes in the CVQ level. The level information is to the right abstracted and replaced with an operation on a variable L. $L := L + 1$ corresponds to a push onto the stack and $L := L - 1$ corresponds to a pop. The example is simplified since there must also be an initial value of L and special considerations for when L reaches its minimum and/or maximum (for both the left and right machine).

$\mathbf{a}^n\mathbf{b}^n$-problem is a CFL (Context Free Language) where the number of **a**s needs to be stored, possibly by "storing" the **a**s themselves. An FSA approximation of a CFL is always an illusory abstraction, since, in principle, there is no upper limit on the length of the strings (i.e. n is unbounded).

To induce a CFL grammar using `CrySSMEx`, the CVQ-level (cf. Definition 10.6) could be imposed on the SEs of the SSM. It may then be possible to identify a subset of SEs, $\mathcal{Q} \subset Q$ such that another subset of SEs, $\mathcal{Q}' \subset Q$ can also be found in the same SSM and the SEs of $\mathcal{Q}'$ are semi-equivalent to the SEs of $\mathcal{Q}$ apart from that their levels always differ by the same amount. If a sequence of such subsets can be formed, it may be possible to replace transitions between the subsets by the pop and push operations of a stack. The hypothesis is that the levels of the CVQ may possibly correspond to the levels (i.e. amount of stored data) of the external stack. Figure 17.1 depicts a simplified example of a possible situation.

In the $\mathbf{a}^n\mathbf{b}^n$-domain, this would correspond to a simple machine connected to a stack which takes care of the counting of **a**s and **b**s. Of course, this extracted CFL would be a hypothesis that an unbounded number of **a**s can be counted by the underlying system although no observations of this can be made. In other words, this kind of extraction would relax the closed world assumption of `CrySSMEx`.

The CVQ is not only holding the information about the level which could be used as suggested above, the graph is itself a data structure that may hold information of the underlying domain. For example, Elman (1990) trained an RNN to recognize simple sentences. His analysis of the state representation revealed that the RNN had grouped related words in a fairly semantically oriented hierarchical structure (cf. Figure 5.1). For example, the RNN separated, in its state, animates from inanimates and humans from animals etc. The study was conducted by the use of hierarchical cluster analysis, which does not take into account the dynamic properties of the RNN at all (Elman's paper was written before any RNN-RE paper, cf. Section 5.1).

With `CrySSMEx`, it is possible for both the semantical and syntactical information to be extracted simultaneously. It would be very interesting to replicate Elman's analysis by using `CrySSMEx` to investigate to which degree semantical information can be traced in the CVQ graph[4]. Further interest lies in the sense that it would replicate a scientist's analysis using an automated analyser.

17.6 Relative SDTDS analysis

17.6.1 $d(sdtds_1, sdtds_2)$

Consider two SDTDSs, $sdtds_1$ and $sdtds_2$ with an identical input and output domain, but with different state spaces and transition functions. It could, for example, be two RNNs trained on the same domain. From these two systems, it is possible to create a third SDTDS, $sdtds_{1+2}$, in which the state and output domains are simply augmented and the corresponding transition functions of the systems handle their part of the augmented state and output space. In other words, it is possible to describe several SDTDSs as one larger SDTDS. There is nothing strange in doing this, as the SDTDS state and output vector spaces are not bounded (see Figure 17.2

[4]Many thanks to Nick Chater for inspiring this idea.

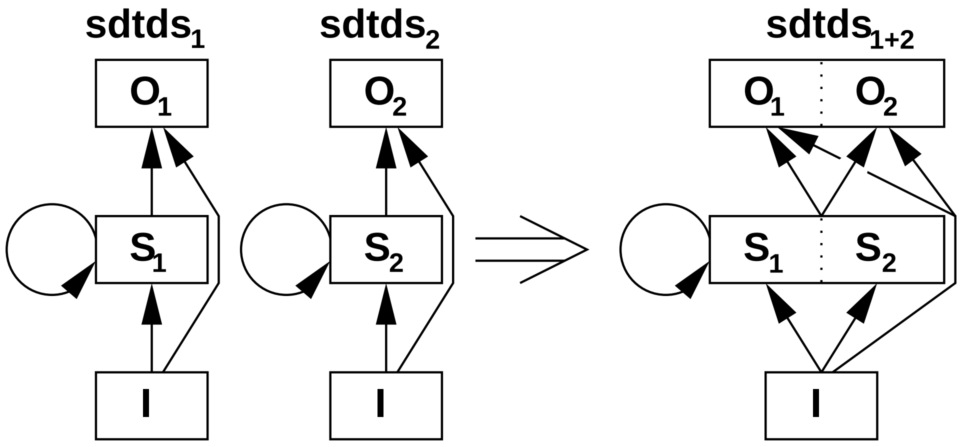


Figure 17.2: An illustration of how two systems really constitute one system. The two SDTDSs to the left, with an identical input domain, can instead be described as one larger SDTDS where both of them are fully preserved.

for an illustration).

The system $sdtds_{1+2}$ describes both systems $sdtds_1$ and $sdtds_2$ simultaneously. Therefore, `CrySSMEx` applied to $sdtds_{1+2}$, corresponds to applying it to both systems at once. If Λ_o is subsequently chosen to reflect the difference of the output of the two systems, then the extracted SSM would also describe this difference. Λ_o could, for example, correspond to the sign of the result of a simple elementwise subtraction of the output vectors. Another possibility, when the output has a symbolic interpretation, is to let Λ_o result in 1 if there is a difference, and 0 if not. Or one could have unique enumeration per each observed combination of simultaneous outputs of the two systems. However Λ_o is chosen, as long as it reflects the divergence of the two systems from each other, the extracted machine should also reflect this difference.

The extracted SSM will be an abstraction of the differences between the two systems since the SSMs extracted from the individual systems are abstractions of their corresponding systems. The consequence is that the difference between two systems is a third system. In the same sense as the difference between two vectors could be a third, difference vector, (if elementwise subtraction is used), the difference of two systems is not necessarily described using a simple scalar. If Λ_o is chosen properly, however, two "similar" $sdtds_1$ and $sdtds_2$ should result in a small SSM extracted from $sdtds_{1+2}$. If the difference of the systems is big, then the SSM extracted from $sdtds_{1+2}$ should, consequentially, also be big.

Thereby, one could define a distance measure between systems by using the extracted SSM as a measure. It basically means that the difference between two systems is described by an SSM that defines a finite state grammar of differences between the two systems.

The SSM could be used as it is to describe the details, or the SSM could be the basis of a quantitative difference measure. For example, $\log |Q|$ (where $|Q|$ is the number of SEs in the SSM extracted from $sdtds_{1+2}$) could be used. If there is no difference between the systems, or if this difference is only in terms of a direct one-to-one translation of output symbols, and the resulting SSM thereby only has one SE, then this results in a distance of 0. This is appealing since the difference between two identical things should be zero (cf. that $d(x,x) = 0$ if d here denotes the difference between two real numbers). All more complex differences between the systems will return a higher number. The distance from a non-active SDTDS (always producing the same output) to an active SDTDS would be isomorphic to the latter. This is also appealing if we consider the non-active system a point zero from which the deviation should only be a consequence of the active system (cf. $d(x,0) = x$).

If any of the underlying systems are chaotic, however, this kind of difference measure could become problematic since `CrySSMEx` will not terminate at a deterministic machine. However, it could be possible to generate one distance measure per each SSM of the `CrySSMEx` iterations, especially if the automatic extraction of deterministic SSMs is implemented (cf. Section 16.2.4).

Every kind of distance measurement comes with weaknesses, however. Not even the typical choice of Euclidean distances between vectors is entirely obvious (for example, city block distance is sometimes more appropriate). However, while pitfalls are kept in mind, this kind of distance measure between dynamic systems could turn out to be quite useful in many contexts. For example, to ensure diversity in sets of RNNs that are to be used as ensembles (Krogh & Vedelsby, 1995).

17.6.2 A "grammar of mistakes"

One possible usage for the difference between two systems as a basis for Λ_o is to extract a "grammar of mistakes". If the underlying system is an RNN trained to perform in a domain, the output of the network typically deviates from the desired target output. Then Λ_o could be chosen to reflect this discrepancy (as it is suggested to reflect the divergence of two SDTDSs in the previous section). It could, for example, result in 1 every time there is an error and 0 if not. Alternatively it could be a more detailed enumeration of each possible error, e.g., one unique symbol for the specific error corresponding to "the output was **a** but it should have been a **b**".

The resulting machine would not describe the output of the RNN, but only the ways in which the RNN conducts errors. This description of the RNN mistakes could be used as guidance for generating more data on which to train the RNN. It could potentially also be used in other ways to refine the RNN training procedure. For example, it was sometimes possible to see exactly when some of the $\mathbf{a}^n\mathbf{b}^n$-RNNs performed mistakes for longer strings based on the extracted SSMs, e.g., the RNN predicted eleven **b**s after twelve **a**s but was otherwise correct.

A problematic issue, however, is that Λ_o would not be a function purely of the output of the system, but also of the external domain. This would mean that Λ_o cannot necessarily be described as a function of the output domain of the SDTDS. For example, the exact same output vector of the system may at one instance be correct and at another erroneous. This would corrupt some of the assumptions required for `CrySSMEx` to extract rules. Whether or not this poses a problem in realistic cases, and if so, if this problem and others can be alleviated, remain open issues. One possible way to circumvent the problem could be to extract the SSM as normal and then apply it, instead of the RNN, to the domain and record when and how the SSM performs errors (cf. Section 16.2.5). From an SSM with such information, the output symbols could be replaced with symbols referring to the existence (i.e. binary error/no error information) or the frequency of errors in a transition. That SSM would after minimization be a grammar of mistakes of the underlying RNN, describing with a finite state description exactly for what situation an RNN generates erroneous output symbols.

17.7 CrySSMEx^2

17.7.1 Meta-SDTDS

When CrySSMEx is used on an RNN to generate a deterministic machine it means a transition from an uncountable domain into a countable one. The RNNs themselves cannot be counted or enumerated since they build upon the uncountable real-valued input, state, and output domains. The deterministic SSMs, however, can be enumerated. Thus, if the translation from an RNN into a deterministic SSM is successful, the RNN can be indirectly enumerated by the extracted SSM. Several other attempts have been made to enumerate RNNs with discrete "signatures" that describe their dynamics. For example, by using the recurrent "self-weights" of the state nodes (Tonkes et al., 1998) or an analysis of the Eigenvalues of the Jacobian matrix in the vicinity of fixed points (Bodén et al., 2000). The use of rule extraction to create such an enumeration could, however, be a more profound way of enumerating RNNs since the extracted machines will in effect emulate the RNNs.

If CrySSMEx was used on a set of RNNs to generate a set of SSMs, these SSMs would potentially end up in a set of equivalence classes (cf. Blair and Pollack (1997)), where all SSMs of one equivalence class are indistinguishable from each other in terms of their output in all situations. These equivalence classes can each be enumerated with natural numbers, $\{1 \ldots n\}$. These CrySSMEx-enumerations of the underlying RNNs could then be viewed as a quantizer (cf. Definition 9.4) applied to the weight space of the RNNs. Of course, each RNN should be exposed to the exact same input sequence so that the difference between Ωs of different RNNs is only due to the RNNs themselves.

If we consider backpropagation (BP), or backpropagation through time (BPTT), these algorithms can in themselves be viewed as systems falling under the SDTDS definition (Definition 9.1). Given a fixed learning rate and a fixed training set, BP will make transitions in the weight space of the trained RNN, i.e. in one epoch, BP will make a transition from one weight configuration into another. The weight space corresponds to the state space of the SDTDS, the input space is empty and the output could be equal to the state space. The transition function is simply the

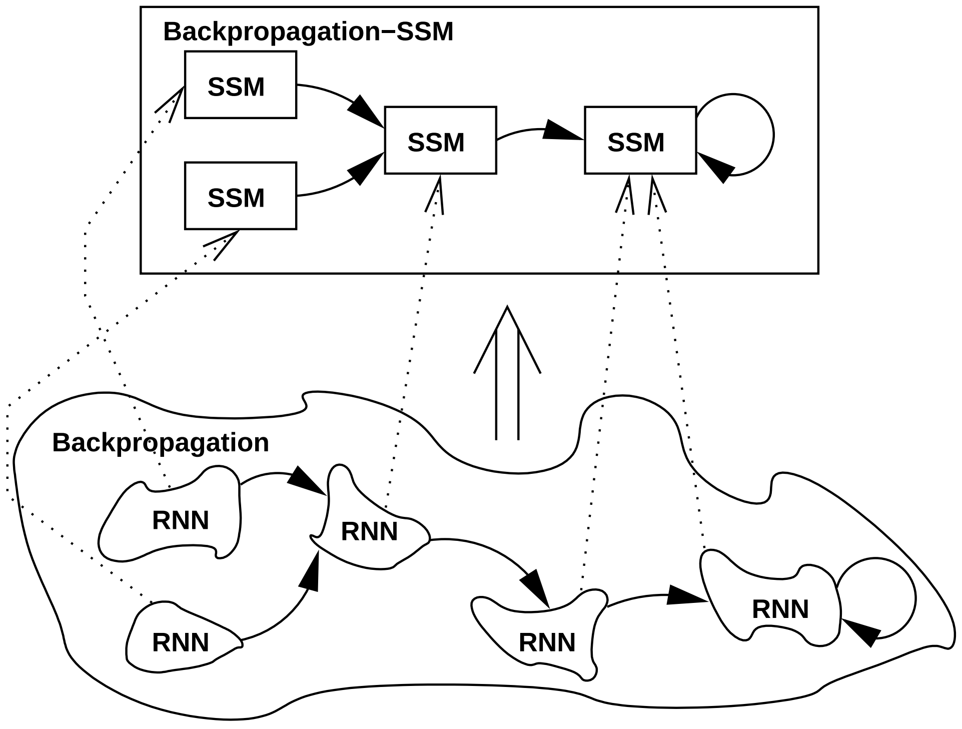


Figure 17.3: An illustration of how `CrySSMEx` could be used to analyse the backpropagation algorithm (or some other deterministic training algorithm) when used for training RNNs (or some other SDTDS). Backpropagation performs transitions in the weight space of RNNs. `CrySSMEx` is applied to each RNN and transitions between equivalence classes of RNNs can be described. `CrySSMEx` is then used on the level of backpropagation as the underlying SDTDS and the weight space is processed such that equivalent RNNs will be grouped together and inequivalent RNNs will be split. The two leftmost RNNs correspond to random initial weight configurations. In reality, the BP-SSM should become considerably larger.

gradient descent based updating of the weights of the RNN.

Since the weight space of BP can be viewed as a state space in the BP-SDTDS, and since `CrySSMEx` could be used to enumerate (or quantize) this state space it would be quite straightforward to use `CrySSMEx` to analyse BP or BPTT by letting `CrySSMEx` itself be the basis of Λ_o. `CrySSMEx` would then be used on two levels (hence the name `CrySSMEx`^{2}), one to enumerate RNNs, and one to extract the transitions between equivalence sets of RNNs (cf. Figure 17.3).

If successful, the results should be quite informative for the analysis of BP and BPTT. The extracted machine would describe sequences of transitions between different RNNs as the RNNs progress towards the desired solution. It would also be possible to see how the BP sometimes “forgets” successful solutions and drifts off to unwanted parts of search space (Bodén et al., 1999; Tonkes & Wiles, 1999;

Jacobsson, 1999). The main obstacles for using `CrySSMEx` in this manner are the computational time required and that the weight space may be too enormous to conduct this extraction. But this should be fairly straightforward to implement and test.

17.7.2 Dual systems

Another form of `CrySSMEx`^{2}, i.e. where `CrySSMEx` is used simultaneously in two different contexts, would be to extract from two interacting systems. For example, consider that one system is an agent, and the other one is the environment in which the agent operates in. The input-output relation of the two systems would be reversed, i.e. the input of the agent is the output of the environment and vice versa.

Let the state spaces of the agent and environment be termed S_A and S_E respectively. Viewed from the agent's perspective, its state S_A, would have the following basic interaction $I_A \Rightarrow S_A \Rightarrow O_A$ whereas its environment would have this interaction: $I_E \Rightarrow S_E \Rightarrow O_E$ where $O_A = I_E$ is the output/action of the agent and thereby the input to the environment, and $I_A = O_E$ is the sensory input of the agent and hence the output of the environment. Figure 17.4 depicts a schematic description of the agent-environment duality. The internal dynamics of the systems are here ignored, but naturally the state of the systems will also affect themselves.

The extraction using `CrySSMEx` requires Λ_o to be specified and Λ_i to be invertible, but that would not necessarily hold in this example (i.e. some of the obstacles described in Section 17.9 may have to be solved first). The extraction of a machine in one system may, however, be used to refine the Λ_i and Λ_o of the other system (remember that Λ_i of one system is the Λ_o of the other). This is highly speculative, of course, but the potential end result would be an extraction of a symbolic description of how the two systems interact (Figure 17.4).

A good start would perhaps be to limit the agent's repertoire of possible actions to a finite set so that a Λ_o of the agent could be easily specified. This would allow for a deeper analysis of some autonomous robot experiments where RNNs have been used, for instance, Meeden (1996) used a finite set of actions, e.g., "move forward-left" or "move backwards-right".

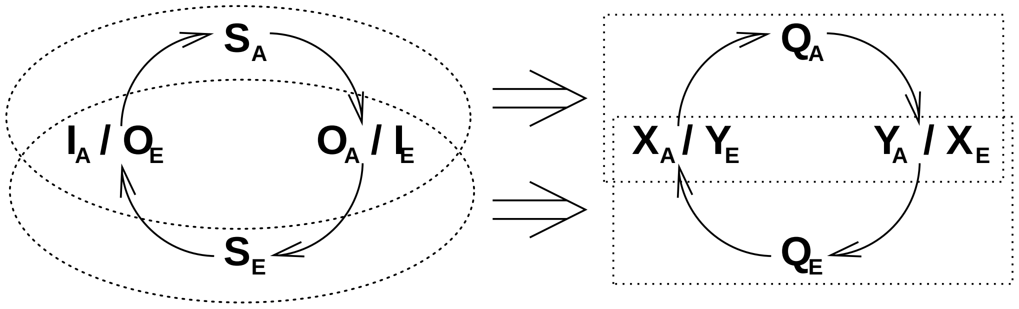


Figure 17.4: An abstract depiction of how `CrySSMEx` could simultaneously be applied to dual interacting systems. In this example A stands for agent and E for the environment in which the agent is situated. Both the agent and the environment are dynamic systems and if `CrySSMEx` successfully extracts finite state descriptions of them both, a symbolic description of the "language" between the two systems would also have been extracted. The sensorimotor agent-environment interaction (I_A/O_E and O_A/I_E) would be abstracted as interchanging symbols (X_A/Y_E and Y_A/X_E) between two discrete (and possibly stochastic) systems.

If the above is possible, then a possible next step is to create what I would call $\texttt{CrySSMEx}^{\texttt{n}}$, where an unspecified number of subsystems can be identified. The idea is that a single large system can sometimes be more adequately described as a set of interacting systems (Watson & Pollack, 2005). For example, if two FSMs of m and n states are translated into a single FSM, the resulting size of this FSM could require as many as $m \times n$ states. For large FSMs $m + n \ll m \times n$ and clearly, in such cases, the two separate FSMs are a more compressed description.

The description of the interacting subsystems, conducting symbolic interaction, could well be shorter than describing the full system using a single finite state model. If an automatic division into subsystems is at all possible, the result of such extraction would be very powerful. While genetic algorithms could possibly be used to speculate about subsystem divisions, the fitness of such speculations could be evaluated by the above described $\texttt{CrySSMEx}^2$.

17.8 Truly parameter free `CrySSMEx`

`CrySSMEx` is only truly free from parameters if we consider Ω and Λ_o as derivable from the domain somehow (see Algorithm 11.2). Λ_o can be seen as derivable when, for example, the domain is symbolic which has typically been the case when RNN-RE has been applied to RNNs trained on formal grammars (cf. Part I). But could the parameters be reduced in a broader range of situations?

17.8.1 Guessing Λ_o

If there is no known underlying symbolic domain or other natural symbolic interpretation of the SDTDS output, then inferring Λ_o (cf. Section 9.1.3) from the SDTDS alone seems an impossible task. The size of the resulting SSM could, for example, not be used as any indication. If Λ_o is cvq^0, for example, then the resulting SSM has only one state, all inputs leading to the same transition resulting in a single output symbol. If Λ_o is more finely grained, however, the SSM may be very large but with output symbols that are very subtly differentiated semantically. The very reason Λ_o is a parameter of `CrySSMEx` is that it should be chosen by the user to reflect something meaningful. For example, if the underlying RNN is trained on symbolic data, it makes sense to let Λ_o reflect the symbolic interpretation of the RNN output.

It could, however, be possible to use a whole range of output quantizers simultaneously, each resulting in its own "SSM-view" of the SDTDS. Perhaps there could be context dependent features from which can be estimated the adequateness of these different views? For example, if the underlying system predicts the stock market, and the output is in a range from -10 to 10%, then the accuracy of the numerical prediction of the stock value, calculated using a numerical interpretation of the symbolic output of the SSM, could be used as an evaluation criterion.

17.8.2 Generating Ω

It may not seem to be easy to avoid the sample set generated from the SDTDS as a parameter. The algorithm should after all need some examples from which to induce the model. But since the underlying system could potentially be immediately interacted with, i.e. by feeding it input to see what happens, it would suffice to give only the system itself as an input parameter. `CrySSMEx` would then itself choose what inputs to use to generate Ω. Some ideas about how to perform this interaction are, however, discussed in more detail in Sections 18.4 and 18.6.

17.9 Gradual removal of SDTDS constraints

The SDTDS definition is used in this thesis in order to not restrict the possible systems to RNNs only. The definition is therefore quite wide and details such as activation functions and weights are ignored so that many other architectures comply with the description. It may, however, still be too restrictive for a wide range of simulated systems. These restrictions are obvious targets for further development of the algorithm.

17.9.1 Determinism ⇒ Nondeterminism

One major obstacle for the use of `CrySSMEx` for a broader range of simulated systems, is that it cannot handle noisy systems. Random noise is often added in simulated systems to create more realistic simulations and to "smear out" possible systematic mistakes due to erroneous assumptions.

`CrySSMEx` will have problems with an underlying noisy system for many reasons, for example:

- If an SE is nondeterministic, is it so because it should actually be split or is it due to noise? Indeterminism stemming from a poor quantization can, and should, be handled by a SE split. But indeterminism due to noise will not be helped by such splits.
- If two SEs are almost equivalent, but not quite, is their inequivalence due to the noise, or an actual inequivalence? Should they be merged or not? And how do you determine if two SEs are almost equivalent[5]?
- When should `CrySSMEx` terminate? A fully deterministic SSM cannot be achieved as the underlying system is not deterministic. Hence, some other termination criterion must be used.

My conjecture is that the solution lies in abandoning the simple deterministic progression from ssm^0 and upwards. Instead I find it likely that a heuristic or breadth first search needs to be conducted. There is a need for a backtrack possibility since the consequence of a split or a merge may not be fully apparent until it

[5]Kullback Leibler distance of output distributions may be a good start (Cover & Thomas, 1990).

is conducted and a new SSM is formed after it. Therefore a number of alternatives may need to be tested for each SSM. For example, if an SE is split but the split creates two SEs with no significantly decreased indeterminism, perhaps the split should not have been conducted.

The generated search tree would have some practical consequences on the CVQ graph. It would not be reasonable to create a different CVQ graph for each possible vertex in the search tree. The CVQs will be related to each other and have large overlaps. Rather, a multi-version CVQ should be created so that the quantization of a vector using multiple versions can be conducted simultaneously.

17.9.2 Discrete input $\Rightarrow$ Continuous input

The input space of the SDTDS is not explicitly limited in the SDTDS definition. But for `CrySSMEx` to function, it needs to be discrete and Λ_i must be invertible. This is due to the fact that each transition in the final SSM requires a unique input symbol to label it.

Consider an SDTDS with a discrete set of input patterns, but with no predefined input quantizer. Let Λ_i be cvq^0. Then `CrySSMEx` could at each nondeterministic SE perform the split on either Λ_s or Λ_i. If the split of the input space reduces the ambiguity of the output symbol then it is successful. If not, then split the SE as usual. If, for example, the input space of the system illustrated in Section 12.1 (cf. Figure 12.2) has no known quantizer, the first SSM would have one SE, and one input symbol $\mathbf{x}$, and at the first iteration the input space would be sampled since the state space is in the `collect_split_data`-function (cf. Algorithm 11.1). This would continue until a deterministic SSM is extracted.

If the input space is truly continuous, another problem will occur. Even if the underlying system is truly deterministic, any finite description of the input space could give the impression that the system is not deterministic. If, for example, two input vectors $\vec{\imath}_1$ and $\vec{\imath}_2$ result in two significantly different states of the SSM but are quantized as the same input symbol, then from this input symbol alone, the state could not be predicted. Hence, the same search procedure which is suggested in the previous section would have to be used (with a different termination criteria etc.). In other words, a breadth first, or heuristic, search is suggested, as in the case of indeterministic SDTDSs, but with the additional operation to split input symbols.

17.9.3 Full observability $\Rightarrow$ Partial observability

A problem related to that of noise, is when the state quantizer has no full information of the underlying system. This is typically always the case in real world domains; some things will always remain hidden since no full nondisruptive measurements can be done (cf. discussion of Plato's cave in Chapter 1). If the state space is not fully observable, it will again resemble the situation of indeterministic SDTDSs since the effect of hidden variables that cannot be modelled will be observed as noise. A solution similar to the one suggested in Section 17.9.1 could therefore apply.

The partial observability problem may, for example, arise when the state is not directly observable, but first passes through some function which reduces the information content in comparison to the full state. For example, if the full state is a physical environment, the environment state will only be indirectly accessible through sensors.

17.9.4 Discrete time $\Rightarrow$ Continuous time

The restriction to consider only discrete time is not necessarily required in the SSM description of the underlying system. The transition functions could possibly be replaced with continuous time differential equations with an arbitrary choice of Δt. The modelling of continuous time SDTDSs could thereby also be a potential possibility. To do this from scratch would, however, probably mean reinventing large portions of control theory. Clearly, this is one direction in which the well developed theories of other fields would have to be used (cf. discussion in Chapter 15).

17.9.5 Real environments

If the above obstacles (of Sections 17.9.1–17.9.4) can be surmounted, then the road is open to real world environments. The real world is noisy and continuous, and only partially and indirectly observable all at once. Thus, if these kinds of systems are to be analysed, all of the above mentioned problems must be handled together and not in isolation. I would, however, suggest that dealing with each problem in isolation would be a good way to start. Furthermore, the best place to start would probably be to work on SDTDSs of the kind studied in Chapter 12 and let `CrySSMEx` find a suitable Λ_i. Then controlled noise could be added to the systems gradually.

If successful, it would be very interesting to implement this kind of system on an autonomous robot which could then explore its environment and build an increasingly accurate model of its actions and their consequences. There is, of course, much earlier work to consider. For example, Fox, Ghallab, Infantes and Long (2006) present a Hidden Markov Model approach for creating finite state models of robot behaviour. While their approach did require some human interpretation of observations, it would be very interesting to see if SSMs and `CrySSMEx` could be used in a similar way and if the procedure then could be more fully automated.

Chapter 18

Sciences of Simulated Universes

In this chapter future ambitions for the RNN-RE field are suggested in two frameworks; Empirical and Popperian Machines. Within these descriptions, `CrySSMEx` serves as a basis and central component of all examples, but the ideas presented are intended as goals that could guide RNN-RE development in general.

Firstly, some properties of simulated systems are discussed from an epistemological perspective, suggesting that simulated systems are very accessible for scientific analysis, and for automated scientific analysis. Subsequently in Sections 18.2 and 18.3, the necessity, feasibility, and revenues of the automatic analysis of simulated systems are discussed. In Section 18.4 Empirical Machines are suggested as an active learner for modelling simulated systems. While Karl Popper's philosophy of science is briefly compared with Herbert Simon's machine learning ideas for solving scientific problems in Section 18.5, Section 18.6 presents the furthest ambition for rule extraction of this thesis: Popperian Machines, i.e. fully automated generators and verifiers of statements, of highest possible empirical content, about populations of underlying simulated systems.

18.1 The golden properties of simulated systems

A single simulated system has some properties that make it very suitable for conducting active learning (e.g. Cohn et al., 1994; Bryant et al., 1999), on it (cf. Chapter 15). Real physical systems are by far much more complex to analyse. If we, for example, want to implement an active learner in the context of, for example, neuroscience or molecular biology, we need to automate not only the ability to put forward theories and test them, but also all other competences involved. A researcher conducting biological experiments needs skills in handling biological tissue as well as planning expertise regarding the cost of the experimentation, etc. To become a skilled experimental biologist may take a very long time. If an automated learner should interact with physical systems in the same manner as a human expert, a considerable amount of sophisticated automation needs to be implemented. In other words, the complete automated empirical loop becomes a huge project compared to simply conducting experiments manually. There must be some kind of gain expected from automating something to motivate the automation in the first place. An even more difficult situation occurs if we move from a laboratory environment into the so called "real world" where repeatable experiments are perhaps only an idealization. In such domains human skills and experiences become even more valuable and, at the same time, more difficult to automate. In psychological studies, for example, how are test subjects selected and interpreted? In astronomy, how are space probes designed and put into space and what probes should be prioritized?

I suggest that if we want to automate scientific processes of any kind, instead of focusing on the big scientific questions, we should more modestly start by looking at systems with properties more suitable for automated analysis. Simulated systems naturally have such inviting properties (but are not necessarily trivial to comprehend, cf. Section 18.2). If we compare the study of simulated dynamic systems with the study of physical dynamic systems, there are some quite obvious differences that make them perfect subjects for systematic analysis. Let us call these the "golden properties" of simulated systems, which when simulated on a computer, allow us to (among other things):

- fully observe every single variable of the system,
- replicate results with arbitrarily high accuracy,
- repeat experiments without much additional effort after the framework for the first experiment has been implemented,
- duplicate and distribute them among research colleagues,
- study the effect of arbitrary pertubations of the systems under controlled

conditions,

- do nonperturbative studies of internal properties to an arbitrary degree of detail.

In other words, they are almost perfect experimental subjects. Very few scientific communities have the luxury of studying entities with properties so inviting for conducting research on them. In fact, some of these properties lay the ground for the possibility of conducting rule extraction from RNNs (cf. the "implicit requirements" discussed in Section 6.5).

For example, one central aspect in science is to infer causality from observations (Pearl, 2000). Sometimes it is obvious which event causes which effects, for example, a glass shatters as a consequence of it falling to the floor, not the other way around. But for some systems causality may become a chicken-or-egg matter, for example, if the concentrations of two enzymes X and Y are correlated in a large set of samples, is a high concentration of X causing a high concentration of Y, or vice versa? Or is there perhaps an unknown cause Z, affecting both X and Y? Such issues are very problematic if there are no additional data.

For simulated systems, however, determination of causality is quite problem free. Let us assume instead that the $X-Y-Z$-system is a simulated one, then it becomes a simple matter of manipulating the levels of X and Y to see the effect of one or the other. Even if we do not directly alter X and Y (since it may be biologically implausible to do so) we can restart the system several times from the exact same initial state. One can also save and retrieve the state of the system at any arbitrary point in time. The controllability of the simulated system allows repeatability by copying and altering the state arbitrarily. In a biological system, the state can never be guaranteed to be exactly the same in two systems. Thus it will never be fully known if the effect of what you want to test, or some possibly uncontrolled aspect of the state of, e.g., a cell, is what is being measured. For simulated systems, however, the inference of cause and effect is trivial. For example, it is in principle trivial to answer questions such as "What would have happened to the simulated system if it at time t was affected this way instead of that way?" (just restart the simulation and simply try it out at time t). Imagine the richness of sciences with answers to such questions, were Reality susceptible to them: e.g., "What if dinosaurs had never become extinct?", "What if gravity was 5% weaker?", "What if I had taken mathematics instead of computer science?", "What if Alexander the Great had lost his first battle?". We will never know the answers to such questions targeting the Reality in which we live. For simulated systems, however, questions of that kind can in principle always be answered. The problem is of course to ask the most interesting questions.

As long as a simulator is properly implemented, any observed phenomena can be recreated and studied in detail. If, for example, one simulated experiment out of a million results in deviant, but highly interesting results, this exact experiment can be recovered and studied again. If one *real* experiment out of a million returns a freak result, then you may only hope to achieve the same result again.

I suggest that every simulated system is susceptible to a scientific method superior to the method of sciences studying the real world. One may even demand that every simulated system is more thoroughly analysed than their real counterparts; i.e. that the possibility infers an obligation. But it is not that simple.

The Achilles heel of simulated experiments is instead that the ease of generating clear observations is a double edged sword. It becomes very easy to generate new results for slightly different conditions or slightly different systems may produce unsurmountable amounts of data. This is also why there is a need for sciences of simulated systems. While these systems are widely used today and can be fully controlled, they may be incomprehensible due to the ease of conducting an arbitrary number of studies on arbitrarily many, arbitrarily complex systems. Each system can in principle have its own "science", including a scientific nomenclature, models and data.

18.2 Incomprehensibility due to abundance and complexity

John Horgan in his controversial book, "The End of Science" (Horgan, 1996), suggests several reasons why our scientific explorations may soon hit a solid brick wall. Horgan is a renowned science writer for Scientific American, and perhaps it takes a journalist with an unbiased perspective on science to dare to suggest there are fundamental limits to science and that those limits may already have been reached. The book should perhaps more properly have been titled "The *Ends* of Scientific *Revolutions*" since he suggests several different causes for scientific limits and predicts a future lack of scientific revolutions (Kuhn, 1962) rather than a lack of scientific progress in general. For example, quantum physics could only revolutionize physics once, whereas refinements and applications of quantum physics may be developed indefinitely. However, some areas such as particle physics, may soon reach a limit due to the physical unfeasibility of testing some hypotheses because the cost could become astronomical (quite literarily so, since required particle accelerators may surpass our solar system in size).

Potential scientific progress may also be impeded by human limitation in understanding a subject to the degree that accurate and meaningful hypotheses can be made. A potential solution to this is to exclude the human element from the equation and let computers without our cognitive limitations suggest and test the hypotheses. This solution is suggested in light of Horgan's book by Riegler (1998) and the subject is also briefly touched upon in Horgan's book itself. Therefore, if machine intelligence is the key to the science of the incomprehensible, why not start with simulated systems that have such inviting properties for conducting research on them (cf. previous section)?

It is quite easy to create simulated systems that behave in incomprehensible ways, even to the designer. Just create a system which alters itself as it runs and you may soon be scratching your head trying to figure out what it is doing. Clearly, the possibility that the scientific investigation of a simulated system may become intractable for a human is quite conceivable (despite the golden properties presented in Section 18.1).

A fundamental problem facing many empirical computer scientists is that it is much easier creating large numbers of new computational models and observations than actually understanding any of them. This situation is especially true in areas where automated model building is part of the research, e.g., much ANN research. This is because when one studies a phenomenon such as a neural network training algorithm, the phenomenon manifests itself in a class of computational models, i.e. the networks themselves.

Suppose the level of the model builder (trainer) is called *level 0* and the level of the resulting models (networks) *level 1*. In the study of backpropagation, which is a deterministic gradient descent algorithm for training ANNs, for example, the algorithm results in a trained network for every random initial network you start with (which is the standard procedure for training). Furthermore, the result varies with selection of learning rate etc. The backpropagation algorithm is in this context a level 0 object and the network a level 1 object. Level 0 objects create level 1 objects.

If the scientist wants to analyse some aspect of the system at level 0 (within the context of some specified domain), then the empirical study needs to take place at level 1, i.e. the level on which the system manifests itself. As in any empirical study, more than one object needs to be incorporated, and, in many cases, the differences between individual generated models are not insignificant, necessitating a considerable number of models to be generated and studied. In the case of backpropagation, the resulting generated networks may be very diversified despite being trained on exactly the same domain. Each network can then be studied and analysed in its own right. A few of the networks may, for example, have completely novel and surprising solution to a problem, as exemplified in Ziemke and Thieme (2002) when it was discovered that some evolved networks, controlling a robot, used the environment as its memory instead of using its internal representation. In order to discover such surprising behaviour in the networks, each must be studied in detail (or, at least, one must be lucky enough to study the interesting ones closer).

The level 1 phenomena manifest themselves in what we can call *level 2* (see Figure 18.1), which in the case of neural networks corresponds to the behaviour of the networks within the given domain (cf. Ω of Definition 9.3). The generated collective of level 1 models are almost always evaluated quantitatively at level 2, e.g., a performance estimation of the networks (e.g. Miller & Giles, 1993; Jacobsson & Ziemke, 2003a). There are also more qualitative evaluations of the networks based on visual analysis of the behaviour (e.g. Meeden, 1996; Ziemke & Thieme, 2002).

Based on the collective results at level 2, conclusions on the aspects of the models of level 0 are then drawn, typically without incorporating the individuality of the level 1 models. In other words, there is an explanatory gap between the

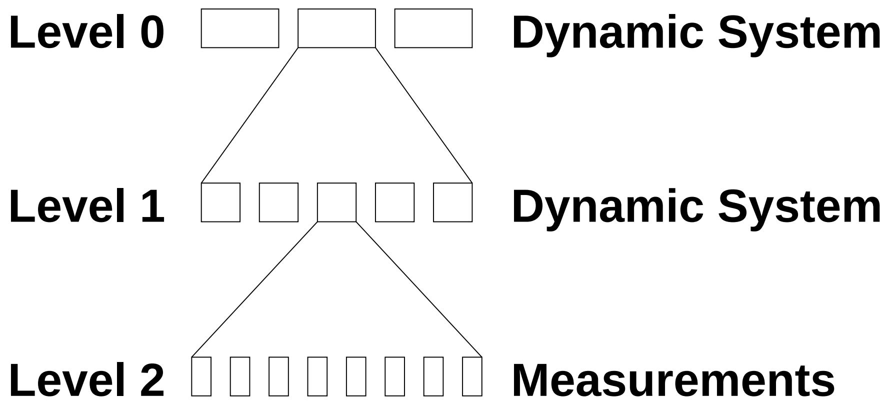


Figure 18.1: An illustration of the information explosion that many empirical computer scientists may encounter. Level 0 objects may for example be different training algorithms that each will generate one or more level 1 objects, e.g., neural networks. The trained models have one or more measurable behaviours in different situations. To explain level 0 systems, more than one level 1-system may therefore have to be examined in turn.

trainer of models and the models' behaviour. For example, if a number of potential backpropagation parameter settings are to be compared in a domain, the final performance of the resulting networks in terms of their generalization error would typically be used to evaluate which setting is the best. But if the specifics of the dynamics of the network is of interest then this performance analysis, of how the networks manifest themselves on their domains, may not be enough. For example, if the networks of Ziemke and Thieme (2002) had only been quantitatively evaluated and not visually inspected, the fact that some networks utilized the environment as memory would probably not have been recognized. The individuality of level 1 objects is lost when level 0 phenomena are evaluated only on an averaged collection of level 2 data.

For other fields of science, where data collection is more costly, this would seem absurd. For example, it would be unforgivable to not study data from space probes in great detail considering the cost of gathering it. Treating data from space probes as a collective set of data without accounting for the individuality of the probes or the planets they are probing would be considered quite absurd. Yet, this is precisely what is done when a training algorithm is used to generate systems that "probe" the search space of the training algorithm. Each system may be a unique solution to the problem found by the trainer, yet such individuality is lost when a mere performance measure is conducted and then averaged for several individuals.

The problem for the empirical computer scientist is that each model at level 1 is itself, although relatively easy to create, a potentially complex phenomenon for which theories can be put forward and tested. Theories which explain the *mechanisms* behind how the numerous level 1 models manifest themselves at level 2 may require more than a superficial analysis of quantifiable aspects of this manifestation. This is typically done only on selected individual models, due to the amount of effort needed to perform a complete empirical study on each object (e.g. Pollack, 1987; Meeden, 1996; Rodriguez et al., 1999; Bodén et al., 2000).

The basic problem here is not only whether or not the complexity of level 1 systems supersedes the human possibility of understanding them (as Horgan (1996) suggests as a reason for halting scientific progress). For example, there are many papers in which individual recurrent networks have been analysed in detail and have arguably been understood by the authors (and readers)[1]. The problem is rather that a detailed analysis of a handful level 1 objects may not be sufficient to understand the properties of the level 0 objects. It may, however, be too costly for humans to analyse each individual level 1 object.

There are many instances of human scientists spending entire careers on subject matters that are seemingly very narrow. For example, biologists working on just a few selected proteins for most of their careers. This is how some sciences have become organized through the success of reductionism (and as a consequence of some sociocultural aspects according to Kuhn (1962)). It does, however, seem sensible for someone to be funded for analysing a very specific neurotransmitter and its role in Alzheimer's decease, for example. The potential of such research lies in applications which may help people. There are, however, considerably fewer people (apart from some overly enthusiastic mathematicians, perhaps) building their careers on the analysis of one or several instances of *simulated* systems, even though some such systems may be sufficiently complex for researchers to spend a lifetime learning new things about them. One reason is that the knowledge acquired about a simulated system may only indirectly yield dividends in the real world. Another reason is that for every simulated model that can be created, an uncountable number of variants of it can also be created. Why focus on one model,

[1]See Section 2.1 for a number of examples of such papers.

when a new one can be created which may be more interesting? The problems for an experimental computer scientist are that there are too many choices every step of the way towards creating and analysing simulated systems.

The relative ease of creating new systems that can (and certainly should) be studied yields a very low revenue from the analysis of each individual system. Consider simulations of chemical reactions in an artificial molecular system with different reaction rules and concentration levels of reactants, or simulations of galaxies formed under different conditions. Another instance is simulations of thousands of recurrent neural networks created by genetic algorithms for the purpose of controlling a simulated robot arm. A detailed manual study of a randomly selected individual system in these example areas will most likely not be very rewarding.

Simulated systems are abundant in contemporary research and with the means of creating one system, another can easily be created by tweaking some parameters and running the level 0 simulator-generator again. Each individual level 1 system may hold the key to whatever problem you are trying to solve, but carefully conducted scientific studies on each of them become practically impossible. This is why automated analysis of simulated systems is important. For real world systems the potential prognosed pay-off, in terms of the knowledge gained and the application of some research may be sufficient to motivate financing humans to conduct the research. For individual simulated universes, however, the low payoff alone may be sufficient motivation to automate the analysis. Moreover, machine analysis rather than human reasoning may be more appropriate for some simulated systems. This is because a simulated system can easily be created to be counter-intuitive and abstract in a way that renders past human experience useless in the analysis process. See Table 18.1 for a brief summary of some of the differences between simulated worlds and reality.

If the golden properties are utilized to automate the analysis of simulated systems, what is then the purpose of the automatically generated models of these systems? The simulated system is of course in itself completely described in source code or something akin to it. This issue is central in rule extraction and the motivator is traditionally that a comprehensible model should be created from an incomprehensible system. In the following section I argue why this motivation is

Real world	Simulated world
There is only one observable real world.	Create as many simulated worlds as you like.
Acquired knowledge may yield high payoff (e.g., applications).	Knowledge is of low value since it will be only about the simulated system and nothing else.
Uncontrolled noise.	Controlled noise.
Repetition of experiments require skill.	Repetition of experiments require only copy-paste of system state and parameters.
Human intuition may be helpful since humans have experience of the nature of the real world.	Simulated systems may be entirely unintuitive.
Time is (or appears as) continuous, linear, divided into past, present and future, and cannot be controlled. Only if the present contains information about the past can historical analysis be conducted. Prediction is difficult.	Time can be linear, cyclic or tree-like and discrete, history is always accessible for analysis, future can always be predicted (i.e. presimulated in separate time line).
Can only be controlled indirectly, through interaction.	Can be controlled in a "hand-of-god"-like manner.

Table 18.1: Some highlighted examples of why it is easy as well as reasonable to conduct a scientific study of a simulated system.

not as important as it seems.

18.3 Models as proxies for queries

I would suggest that the comprehensibility of extracted rules should not be the sole basis for the assessment of the usefulness of rules (cf. Andrews et al. (1995); Tickle et al. (1997, 1998)). The rules, or models, of some underlying phenomena can be useful in other ways than being directly read and comprehended by humans. Traditionally, models of something should accentuate certain aspects and omit others in order to promote understanding and ability to control the phenomena (Föllesdal, Walløe & Elster, 1993). This is especially clear in control theory where the models should be simple enough for engineers to develop and scrutinize them, yet sophisticated enough to control the plant. But, with regard to automated model building, the role of the engineer is replaced by a machine. The virtue of the model as a means to achieve control is, in my view, not diminished by being machine created. For real control applications, however, legal problems may arise if no team of engineers can be held responsible for the system (the legality issue is also used as one motivator for rule extraction in Andrews et al. (1995)). When considering simulated systems, however, models of the systems can be built automatically without any risks involved (cf. initial discussion of Chapter 1).

The possibility of using models to control a phenomena is, nevertheless, not the most essential if the underlying system is a simulated one. There may be a desire to understand the system, but this may be rendered impossible if the model of the system becomes more complex as a consequence of optimizing the fidelity. The comprehensibility/fidelity tradeoff (Craven & Shavlik, 1999) means that the better the model mimics the underlying system, the bigger and more complex it may become.

I would however argue that if a model has certain properties, then, even if it is large and incomprehensible, it may still be meaningful in terms of comprehension. For example, consider a highly complex simulated model of hot plasma, for the purpose of building a fusion reactor. The model may have millions of state variables and build on quantum mechanical principles, as well as being highly non-linear. Despite being incomprehensible (within mortal limits of understanding), the researcher depends on the model to answer queries such as "will this magnetic field configuration result in a stable plasma?" and expects responses such as "Yes, in 90% of the cases.". The incomprehensibility of the system itself is of little significance (given that it is accurate with respect to the relevant underlying physics) when the researchers receive an answer which may very well be comprehensible.

Consequently, the idea is that models may be useful as a *proxy for queries*. I would hold that one strength of models, in science, mathematics and maybe even as mental representations, is that the model acts as a query-proxy between the question-holder and the "reality" that the question addresses. The virtue of any simulation lies in that the simulator is a model which is much cheaper, and more appropriate, to query than the reality itself. And when it comes to models of simulated models, the more abstract model should be constructed such that it is, in turn, more appropriate for queries than the underlying simulated system.

Consider for example SSMs as models of SDTDSs (cf. Part II). The user could of course interact with the simulated SDTDS directly, by testing various combination of input patterns. But `CrySSMEx` creates the SSM model of the SDTDS as a potential proxy for certain kinds of queries about the underlying system. Then, as suggested in Section 17.2, various questions could be asked of this model without the need to interact with the underlying system directly. The extracted SSM is more appropriate for queries since it has a well defined syntax in the structure of the SEs and transition as well as defined semantics represented by the input, output and state quantizers.

To illustrate the strength of a model as a proxy for queries, consider a very simple model of a population of real-valued measurements as a mean value and standard deviation. Let us say, for example, that you have measured the length of one thousand slimy earthworms, a nasty and cumbersome task by many standards. From this exercise you know that the average length is 15 cm with a standard deviation of 3 cm. To create a lossy model, in this case, you choose to assume that the lengths are normally distributed. The model is lossy in the sense the exact lengths of all measured worms cannot be recreated and other aspects of the worms, such as degree of sliminess, are completely ignored[2]. It is a very powerful model for the length of earthworms, not only for the ones that have been carefully measured, but a model that is assumed will hold also for many other earthworms collected under similar conditions. In fact, it may even be assumed that it holds for all earthworms that have ever existed or will ever exist. Even if an infinite number of earthworms will exist before the end of time, you will have a model for them too, accurate or not. From the data alone, without assuming normal distribution (or some other criteria) as your criteria for compression, you could not have expressed anything more substantial than statements about specific lengths of the 1000 individual earthworms you have encountered. Any statement about these *specific* 1000 worms you could have been more accurate, but without the compression of the model, would you really understand the domain? Compression is, if not the actual act of comprehension, clearly helpful for your comprehension. A deeper discussion of the suggested relation between compression and comprehension

[2]The normal distribution assumption also makes the model lossy in the sense that if it is a false assumption, the model will be inaccurate.

can be found in Chaitin (2005).

The power of the model is not only that it generalizes to more data than just the collected data. The power of it as a proxy for queries is realized when you may have concrete questions regarding the lengths of the earthworms. If you want to go fishing and need earthworms longer than 20 cm in order to catch a really big fish, then you could simply utilize your model of earthworm lengths to calculate the probability of finding such worms. Suppose you want to estimate the expected time it will take to get ten such worms if you dig up ten worms per minute. The probability of an arbitrarily selected worm being longer than 20 cm should be approximately one in twenty, according to your model. From this the expected time it will take to find ten long worms can actually be calculated. Consider if you want the same answer, without the use of your powerful "worm-length-model". Then you would actually have to dig up the desired amount of long worms, measure the time each one takes, and repeat this until you can make a model for the average time needed for the task. It would amount to a lot of worms compared to the elegant worm-length-model powered deduction.

By investing computational time in building a model of a simulated system, the cost of answering certain queries may decrease significantly. In the example above, the collection of data together with some assumptions made possible queries about an infinity of never seen examples. The answers may be wrong, if the model is incorrect. But a single model consisting of two real values eliminates the need to conduct any more measurements once the risk of errors in the model is accepted. This is of course an idealization, but any form of model building should produce revenues in the form of reduced (computational) cost for answering certain queries.

When a model, that is intended to be suitable for queries, is built upon a simulated system, the assumptions made should be such that the model is more suitable for queries than the simulated system is by itself. The assumption underlying `CrySSMEx`, for example, is that a finite state model is adequate. Even when it is not adequate, it may be used as a proxy for queries, although the answers may sometimes be inaccurate. What `CrySSMEx` does is to incrementally generate gradually better models so that the expected accuracy of query-answers, with respect to the actual underlying system, will gradually increase. It will, however, only increase with respect to the collected data, Ω, and when this data is perfectly modelled, `CrySSMEx` terminates.

One could object that there may also exist queries, and answers to these queries, that themselves are beyond our comprehension. Some of these incomprehensible queries may however be exactly the kind of queries that are necessary (given some ad hoc utility function). Thereby the rules can only be made partly comprehensible by being accessible through queries. The problem of incomprehensible models is merely temporarily avoided, and not solved, since, the most significant queries for a particular model may be beyond our comprehension. This is of course true. We cannot escape our finite ability to comprehend complex models. But, sometimes not even queries, or their answers, need to be humanly comprehensible to be useful. On a reasonable degree of abstraction `CrySSMEx` can be seen as asking questions of the latest SSM about what aspects of it need refining, and how this should be done through resampling of Ω. In `CrySSMEx`, the extraction of SSMs progresses with or without our comprehension. In the following section this form of querying, for the purpose of improving the queried model itself, is discussed further.

18.4 Future direction I: Empirical Machines

I will now define the first framework in which I think future RNN-RE algorithms should be developed: Empirical Machines, based on active learning and induction of models through querying of the underlying system (Angluin, 1981, 1987; Cohn, 1994; MacKay, 1992; Cohn et al., 1994; King et al., 2004; Angluin, 2004). A similar active learning rule extraction approach is also suggested by Craven and Shavlik (1994), but for feedforward networks only. For dynamic systems, the problems are quite different than for feedforward networks since a system is fundamentally different from a function in that it changes over time.

In `CrySSMEx` a sequence of models is built based on a predefined set of observations, Ω. As mentioned above, `CrySSMEx` can be seen as "querying" its latest SSM model about how it could be refined such that the data is more properly interpreted. From the answer of this query, the next SSM is then created. More precisely, the SDTDS is interpreted through the CVQ which is adapted to create an SSM description of the SDTDS that is minimal and consistent with the SDTDS sample, Ω. The adaptation of the CVQ is based on pinpointing ambiguous SEs of the SSM through measuring the conditional entropy and selecting data in Ω that may alleviate the ambiguity (nondeterminism). It could however be argued that the principle which is used to *select* data from Ω could potentially also be used to *add* data to Ω.

The reason the conditional entropy is used in Algorithm 11.1 is that it can be interpreted as a model of ignorance. For example, $H_{ssm}(Y|Q = q_i, X = x_k)$ can be interpreted as the degree of uncertainty regarding what the output symbol should be if the SE and input symbol is known. In other words, the constituents of the model that are the most ignorant or inexact are selected for refinement. The H_{ssm} entropies are defined (definitions 9.9 and 9.10) such that it does not consider dead transitions (Definition 9.7) as ignorance. These definitions were based on the choice of the closed world assumption, i.e. if a transition is dead, it is so because it is not represented in Ω and thereby does not indicate any ignorance of the SSM regarding Ω, but rather as ignorance in Ω regarding the underlying system itself. The closed world assumption says: if something is not in the sampled data, Ω, then it is also not in the model.

Dead transitions are, however, only one extreme case of insufficient data in Ω; i.e. when an input symbol has never been presented to the underlying SDTDS in certain situations. This is only at one end of the spectrum of transition frequencies and the only one which can be seen in the SSM at all since the frequencies of SEs and transitions in Ω are not modelled at all in the SSM. The extracted SSM, however, may have some SEs and transitions that could be very poorly supported by data in Ω. For example, if one transition is executed only one time and another 1000 times, in a quantized Ω, this will not be reflected in the SSM at all. It is quite conceivable that a transition supported by a handful of observations in Ω can be considered more volatile than a transition supported by thousands of observations. An SSM is more likely to fail to generalize with respect to unseen situations at the weakest links, i.e. infrequent SEs and transitions. It is also guaranteed not to generalize at all in dead transitions.

One goal of `CrySSMEx` is to create a model which minimizes the uncertainty of the output of the underlying system given a sequence of inputs. If the model mimics the underlying system well, this uncertainty will reach zero. This uncertainty is what is gradually eliminated in the `CrySSMEx`-loop.

Fully eliminated uncertainty terminates `CrySSMEx`, i.e. after the SSM fully describes the data in Ω there is nothing more for `CrySSMEx` to do. This is precisely the point at which `CrySSMEx` could be made part of an active learner; by resampling Ω to cover ignorance in the SSM regarding the underlying SDTDS (cf. Section 17.2.1). The resampling should be done by interacting with the underlying SDTDS in a manner which should make infrequent SEs and transitions more frequent as well as it should eliminate dead transitions.

There are of course many strategies for how to patch up the holes in the SSM. One is to generate an input sequence which according to the current SSM should result in more uniform SE frequencies, i.e. that states should be visited approximately the same number of times. Another method would be to interactively (while the CVQ quantizes the state space) force the SDTDS to follow previously dead transitions. This must be done interactively since, based on the SSM, it is impossible to know what will happen in the dead transition. A reasonable strategy could be to generate a new Ω that maximizes the probability that the underlying SSM should fail to predict the SDTDS. To prevent loops, it is probably beneficial to let the new Ω contain the previous Ω as a subset. When this new Ω has been used to create a new model, the whole resampling procedure could be started over again. For every iteration, the induced model should better mimic the underlying system since the data on which it has been trained was selected to be as problematic for the underlying system as possible[3].

By Empirical Machine, I refer not only to systems built on `CrySSMEx`. As currently implemented, `CrySSMEx` has its specific limitations and features which are not meant to constrain the concept of Empirical Machines. Empirical Machine means a system of model induction which should create a model of a simulated system that should be more accessible to queries than the underlying system itself. In particular, the model must be able to answer queries regarding its own inabilities (i.e.

[3]A similar idea was developed already in Jacobsson and Olsson (2000) (which in turn was based on Jacobsson (1998)) where, problematic, prototypical input patterns were extracted from feedforward networks by "inverting" them.

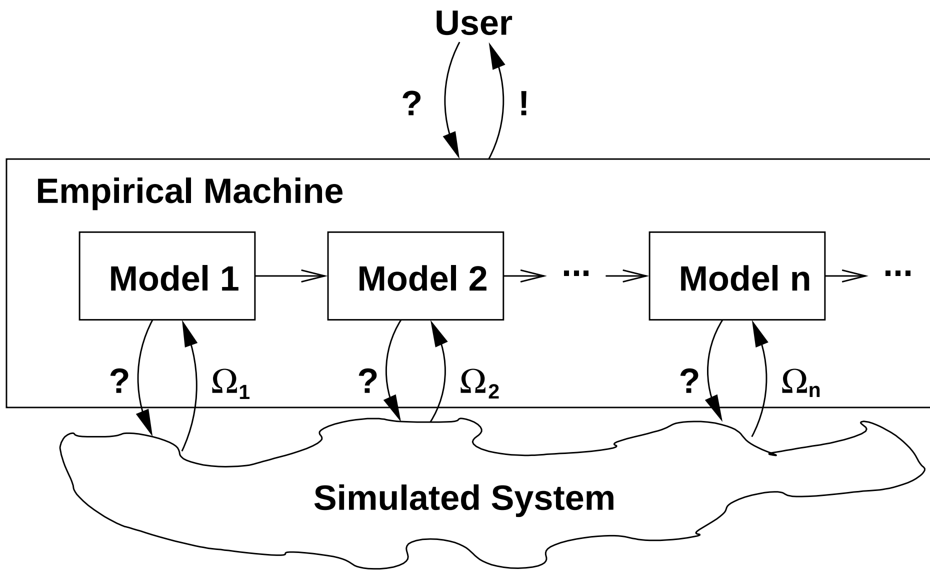


Figure 18.2: Outline of an Empirical Machine. The initial model of the underlying system queries the underlying system in order to improve itself. A sequence of models is thereby created, where increasingly detailed queries can be given as the deviance of the models from the actual system decreases. A user can potentially query the Empirical Machine which acts as an adapting proxy for queries. The queries from the user could be used to guide the refinement of the underlying models.

ignorance) to answer certain queries. Apart from creating the model, the Empirical Machine must also have a mechanism for generating a new set of observations which should remedy the ignorance in the current model. Traditionally, RNN-RE methods assume finite state models, but other models are of course possible. For example, a similar active learning rule extractor was suggested by Craven and Shavlik (1994), but it was limited to feed forward networks only. An Empirical Machine is to be regarded as an automatic method for creating models of simulated system, models that should in principle never stop being refined (or, at least verified) as long as the machine is running. An external user may of course provide guidance by providing additional queries regarding the underlying system. The outline of an Empirical Machine in conjunction with an external user is depicted in Figure 18.2.

Observant readers will remember it was previously argued that it is preferable to let `CrySSMEx` be *compositional*, i.e. to collect data from the SDTDS as it was operating in its domain (cf. discussion in Section 9.1.2). By recollecting data actively, the patterns of an underlying domain of the SDTDS will not be used as heuristics in generating the rules which will result in many aspects of the rules not being relevant for the SDTDS as it is actually operating in its domain. In Jacobsson and Ziemke (2003b) (and Appendix D) it was shown that by using the domain as heuristics, significantly fewer states were extracted than if breadth first search was used. This active learner is therefore more suitable for systems that are not strictly bound by a constrained external domain. For example, the $\mathbf{a}^n\mathbf{b}^n$-predicting RNNs (cf. Section 12.2 and appendices C and D) are not really intended to do anything else than predict $\mathbf{a}^n\mathbf{b}^n$-strings. An Empirical Machine might, however, "conduct experiments" on the RNN using any non-$\mathbf{a}^n\mathbf{b}^n$-string resulting in big SSMs with largely irrelevant aspects in terms of $\mathbf{a}^n\mathbf{b}^n$.

The reason I define and discuss Empirical Machines is that, apart from being a potential extension of `CrySSMEx`, it also provides a framework for other potential RNN-RE algorithms. If one wants to design a rule extractor for the purpose of building an Empirical Machine, some arguably important goals for RNN-RE algorithms and their rules are highlighted:

- By providing rules that can be queried, fidelity could potentially coexist with comprehensibility (cf. discussion in Section 7.2.2) since large incomprehensible rule sets can be viewed through queries that accentuate aspects of relevance for the user. This places a focus on the querability of rules as a quality criteria rather than the traditional criteria fidelity, accuracy, consistency and comprehensibility (cf. Section 4.2.4).
- The rules should be able to assess some aspects of their own ignorance. This is important not only for the Empirical Machine framework, but also for the possibility of providing estimations of confidence when the rules are used to predict or model the underlying system.
- The user can, but is not required to, guide the extraction. In essence, this means the extraction process is further automated since the user needs to do nothing more than provide the Empirical Machine access to the underlying system. Full automation means the Empirical Machine can more easily be incorporated as a constituent of larger systems (which is suggested in Section 18.6).
- In order to build an Empirical Machine from a rule extractor means it must be "user independent" since it must interact with the underlying system autonomously. The importance of freedom from, or consistency over, parameters

becomes accentuated since these parameters would be inherited from the rule extractor to the Empirical Machine.

The first point is perhaps the most important for the field of RNN-RE since it would motivate research on rule extraction also when the rules are beyond human comprehension. Human comprehension has its limits but I see no reason why extracted incomprehensible rules should be deemed worthless if they accurately describe the underlying phenomena. As Einstein once put it: “A scientific theory should be as simple as possible, but no simpler”. The challenge for rule extractors is to show that this may also be true beyond the limit of human comprehension.

Most likely, the extracted models will quickly explode in size as every hole patched in the SSM is likely to generate a larger SSM with even more dead transitions. Therefore some strategies, regarding what aspects of the SSM should be the focus of further resampling of the SDTDS, must be devised. Such *interestingness* measures are commonly used as heuristics in computational scientific discovery and this connection is investigated in more detail in the following sections.

18.5 Popper and machine learning

Scientific discovery involves two main subprocesses; creativity and criticism. Or as Popper states it; “the work of the scientist consists in putting forward and testing theories.” (Popper (1990), p. 31). Traditionally, the machine learning field has been more involved with the former rather than the latter. Ironically, however, this aspect of science is perhaps not the most accessible for automation. To automate something, you must first understand it enough to program it (Chaitin, 2005). Popper states: “The initial stage, the act of conceiving or inventing a theory, seems to me neither to call for logical analysis nor to be susceptible of it” (Popper (1990), p. 31). This has of course received criticism from proponents of machine learning approaches to science; “It is unusual for an author, less than one-tenth of the way through his work, to disclaim the existence of the subject matter that the title of his treatise announces. Yet that is exactly what Karl Popper does in his classic, *The Logic of Scientific Discovery*” (Simon (1973), p. 471,). This could simply be attributed to a poorly titled book. The original title in German was “*Logik* *der Forschung*" (Popper, 1935) which is more accurately translated as "*Logic of Research*" (which sounds less powerful, I suppose). Even more accurately, the book should perhaps be titled "*The Aspects of Science that can Actually be Reduced to a Logical Description*" or, "*How to Separate Science from Non-science*". The last title would indeed reveal the main ambition that Popper seemed to have with his book; to give a detailed description of what science is and how to recognize pseudoscience disguising as science.

The machine learning literature is strongly influenced by Herbert A. Simon, a strong proponent of machine intelligence applied to realistic scientific problems. Simon's articles present a strongly *descriptive* view of science. A paper on a machine learning technique applied to a scientific domain is typically introduced by a description of a success story where a scientist has discovered a novel law. In Simon (1992), for example, diaries, correspondence and laboratory notebooks of a few noteworthy scientists are studied to find patterns in their creativity, intuition, assessment of the validity of ideas and planning of experiments etc. A challenging task indeed. As I see it, the basic problem is, however, that intuition and creativity are not matters easily approached by other means than introspection. Ideas about scientific creativity may possibly be no more than sophisticated guesses at best, since the problem of scientific creativity itself may not be a problem open to the scrutiny of scientific methods (Popper, 1990). The science of creativity is not a science at all, in fact, if we follow Popper's definition of science.

Popper's demarcation of science from non-science, or pseudo-science, is based on his view that science should deal exclusively with *falsifiable* statements. If a statement cannot be falsified through observations, then it is not scientific. Falsification is, however, a property of the statement itself, not of the source of the statement. In other words, in Popper's philosophy of science, the source of statements is a wild-card. Popper never states that there are no logically built up methods that can come up with falsifiable statements, i.e. he never excludes the possibility of the creative element of the scientist being automated. He merely claims that the analysis of human creativity is intractable, and this does not, in my view, exclude the possibility that logically built systems may have "creative" features. Simon and others have certainly been able to develop several such logical programs for artificial creativity in scientific domains such as mathematics, chemistry, physics astronomy biology, medicine etc. (Simon, 1995/96; Colton & Steel, 1999). Novel discoveries are rare (typically known facts are rediscovered) but it does happen. Others have embraced the non-logical nature of scientific discovery and let "random" evolutionary processes be the basis for creative discoveries (Koza et al., 2003). Through the use of genetic programming which builds on random mutations, random crossovers and fitness-based stochastic selection, Koza et al. have been able to find novel non-intuitive solutions to complex engineering problems (typically in the field of electronics).

What defines computational scientific discovery? I would hold that Popper's definition is a good one to describe the middle word; i.e. that only falsifiable statements are "scientific". "Computational discoveries" are discoveries made by an algorithm run on a computer. The process of computational discovery should also involve minimal, or no, human intervention, to distinguish it from *computer aided research* where the computer is used as a tool in the hand of humans. "Discovery" is, in my view, a creation of a falsifiable, yet not falsified, statement about something. The creation itself can be made in any arbitrary way. Since the source of statements bears no relevance in the assessment of their falsifiability, the nature of the source needs no further specification. For the current discussion, we can assume it to be random, human or a highly sophisticated machine learning algorithm. It would be possible to call such statements "facts", but in Popper's philosophy, the notion of a fact is problematic. Nothing can be known for sure, but some statements can be stronger than others by being logically improbable to be true unless they really are true. That is the essence of falsifiability.

You might react to the word "something" in "statement about something". Surely science must be about scientific subject matters, such as physics, medicine or chemistry? But, such a definition of science would be purely descriptive and provide no indication of when or if the study of a particular subject matter becomes a science. On the contrary, I would hold that a proper definition of science is a definition of the scientific method, not of the subject matter. It is the nature of how we approach a subject matter that makes some knowledge scientific and other knowledge not. If a subject is approached with a sound scientific method, then the knowledge generated deserves to be labelled *scientific knowledge*[4]. This is however seemingly not viewed as a sufficient criterion by researchers in the field of computational scientific discovery. The problem domains under study are typically within traditional natural sciences or mathematics (e.g. Simon, 1995/96; Colton & Steel, 1999; King et al., 2004).

Another striking difference between Popper's philosophy of science and traditional machine learning is the anticipated difficulty of approaching the matter systematically. "The central problem of epistemology has always been and still is the problem of the growth of knowledge. *And the growth of knowledge can be studied best by studying the growth of scientific knowledge.*" (Popper (1990), p. 15). The reason that scientific knowledge is considered easier to approach systematically is simply that it is a very limited form of knowledge for which methodologies can be defined. Consider common sense knowledge on the other hand; we all have it (more or less), but can we single out a method for acquiring common sense knowledge? In comparison, scientific progress is a social and well documented process (Kuhn, 1962). Interestingly, however, proponents of traditional computational scientific discovery hold: "Scientific discovery is generally viewed as one of the most complex human creative activities" (Langley et al. (2002), p. 1). I do believe however, that this argument is more a consequence of analysing the result of scientific method, than the scientific models themselves. The scientific method for testing these models is in itself very simple in principle.

The ambition of Simon and his followers is indeed impressive. They attempt to mimic the processes by which the great scientific minds of the past have achieved success. But it is like deciding that Mount Everest is a good place to start if you want to learn mountain climbing. The principle of climbing mountains is very simple: just walk or crawl or climb towards higher ground until you reach the top. The difficulty is more a consequence of the mountain. Likewise, the scientific process is elementary; the resulting complexity is simply a consequence of applying it to complex systems. Ironically, Simon himself provides an appropriate analogy to this in his well known ant on the beach metaphor (Simon, 1969): the complex path taken by an ant on the beach may be a consequence of the complexity of the

[4]Not to be confused with "truth" or "true knowledge". Scientific knowledge is, and should always be susceptible to change.

environment rather than the complexity of the ant. A simple mechanism may result in complex phenomena if put in complex contexts. And I believe this is precisely the case for the scientific method.

The ambitions set aside, in practice, the machine learning field is typically focused on induction of theories from data. Data is gathered from which models are subsequently induced. Various heuristics are used to guide the model induction towards interesting and comprehensible models. Some measures of interestingness are (Colton et al., 2000):

- *Empirical plausibility* of conjectures. They do not suggest always discarding conjectures refuted by observations, instead the conjecture could be altered to fit the data. But the bottom line is that plausibility is taken as a criteria for interestingness.
- *Novelty.* If a conjecture or concept can be deductively derived it cannot be considered very novel.
- *Surprisingness.* Tautologies are the least surprising of conjectures.
- *Applicability.* The proportion of models in a database to which the conjecture or concept is applicable.
- *Comprehensibility and complexity.* Simpler conjectures can be considered more interesting.
- *Utility.* Ability for user to explicitly guide the search for conjectures by specifying a focus that indicates interestingness in the domain from the user's perspective.

Interestingly, Popper's falsifiability is not in the list. In fact, Popper is seldom cited at all in the computational scientific discovery community. This may of course be due to Simon's early criticism of Popper's refutation of analytically approaching the nature of human creativity.

In my opinion, Popper provides machine learning with a very sound philosophical, as well as practical basis, for automating science. He could well be criticized for providing a very poor description of how science is conducted in practice. Most scientists do not focus their attention and experiments on falsifying their own claims. Moreover, much that we consider scientific knowledge may not be entirely falsifiable. But Popper's philosophy of science is not descriptive, it is *normative.* He simply states what he considers scientists *should* do with their conjectures, not what they are *actually* doing. As a consequence, he gives a fairly nonanthropomorphic view of science. The act of falsifying statements is not a typical human thing to do. We prefer confirming our ideas and subsequently applying them. Striving for falsification is, however, arguably a very logical ambition (if you believe in Popper's arguments, that is) and falsifiability should thereby be a good heuristic for evaluating statements we want to be scientific. A good heuristic, that is, also when computer-evaluated.

Moreover, as Popper denies any methodological approach for understanding creativity, there may also not be any methodological approaches for designing devices that exhibit creativity. Thus creativity becomes a wildcard. Theories could be generated by the throw of a dice a la Genetic Programming (Koza et al., 2003), or by a sophisticated guessing game a la Inductive Logic Programming (Muggleton & Raedt, 1994). With any arbitrary generator of statements, it would still fit the Popperian framework as described here. The degree to which the creativity is successful can in a Popperian framework be evaluated by the degree to which the statements are falsifiable, but not yet falsified. Consider Einstein, for example: the assessment of him being a successful creative scientific genius comes from his, quite falsifiable ideas still being unfalsified (despite some considerable effort) a hundred years after his 1905 *annus mirabilis.* But if instead a monkey at a type-writer had put forward the theories by an incredible coincidence, these would have been just as powerful (even though it is unlikely the monkey would have been given any credit in the unlikely event anyone actually started taking them seriously).

Of course, if we developed machine learning techniques that use Popperian falsificationism as a basis, we should not expect this science to resemble human science. Human scientists do not follow the strict schemes of falsificationism. To better understand human scientific creativity, we should instead follow Simon's initiative to be inspired by descriptive philosophies of science.

A similar problem is mentioned by Witkowski (2002) who created a Popperian model of animal behaviour. Although it may be reasonable to assume that a limited form of assessment of falsification of theories could occur in animal brains, the analogy should not be taken too seriously: "Clearly it will not be appropriate to suggest that the principles embodied in 'The Logic of Scientific Discovery' can be wholly or directly incorporated into an animat controller, where the aim is to provide engineering analogues of animal learning and behavior." (Witkowski (2002), Section 6). The same can perhaps be said if we were trying to provide engineering analogues of the human epistemology of science which seems to be the ambition of Simon et al. But my ambition is, rather than to strive for a model of human epistemology as it manifests itself in traditional science, I want to develop a scientifically based *Machine Epistemology* directed specifically at simulated systems[5].

18.6 Future direction II: Popperian Machines

In the following section I suggest how the Popperian framework could be used as a basis for future RNN-RE algorithms that conduct an automatic scientific process on simulated systems: Popperian Machines[6]. In the suggested Empirical Machine, a model is induced through a series of queries to the underlying simulated system for the purpose of acquiring a better model. Every simulated system that is analysed within an Empirical Machine will thereby have an adaptable query-proxy (the induced model) to which a user can ask certain questions. If a particular question requires aspects of the model that are not yet supported by data, the Empirical Machine will, as suggested in Section 18.4, automatically interact with the underlying system in order to acquire this data. In effect, the Empirical Machine acts as an automated experimenter conducting tests on the underlying system.

The Empirical Machine should be able to falsify statements, firstly by consulting its model directly, and secondly by acquiring data that potentially could falsify the statement. Queries to the Empirical Machine (let us adhere to SSMs in these examples) could be in the form of statements, such as "There exists an SE to which the input symbol **a** will always cause a transition from all other SEs". The

[5]The relation between machine learning and the philosophy of science is also arguably a strong one (e.g. Williamson, 2004; Korb, 2004) and this strong connection is what I propose should be utilized in practice. Moreover, in recent interesting arguments against the widespread use of "data-driven" data mining in the bioinformatics field, Popper has been used as an argument against machine learning induction (Allen, 2001a, 2001b; Gillies, 2001)

[6]Not to be confused with Dennett's Popperian creatures (Dennett, 1996). The Popperian creatures are based on the idea that the anticipations of the outcome of different actions (through a sophisticated enough internal mental representation of the world) allow the creatures to select among their actions before performing them.

Empirical Machine should, to test (i.e. to falsify) the statement, based on this specific example query, check all **a**-transitions and see if they all lead to the same SE. Moreover, if any **a**-transitions are dead (Definition 9.7), it should attempt to extend Ω, through interaction with the underlying SDTDS, so that data is collected regarding these transitions. Clearly, the implementation of an Empirical Machine requires a number of complex declarative programming problems to be solved, but let's assume that these are solved for the relevant cases.

If the Empirical Machine can be entrusted to actually collect the data necessary to falsify statements, then populations of Empirical Machines, each adjusted to their own underlying system, could serve as a basis for falsifying statements that are over populations of systems. For example, the statement in the previous example could be expressed as: "In all systems of this population, there exists an SE to which symbol **a** will always cause a transition from all other SEs." If such a statement is falsifiable for just one of the underlying systems then it is falsifiable. If it is subsequently *proved* false in just one of the underlying systems, then it is false.

Although the creativity aspect of this framework was previously referred to as a "wildcard", it should be noted that a successful falsification could be very informative for generating new statements. For example, statements about all underlying systems, falsified merely for one system, X, could be refined as "For all systems except system X...". Such divisions could lay the ground for dividing the underlying systems into subclasses based on what statements can be given about them. Concepts such as "Systems for which statement S is true" could then be introduced into the query language (cf. concept induction Colton et al. (2000)).

The framework for generating falsifiable statements about the simulated system I term a Popperian Machine and is depicted in Figure 18.3. The concept is fairly simple; the generator of statements fills a list of statements which the Empirical Machines attempt to falsify. The statement list should only contain falsifiable, yet unfalsified statements. How to populate the list and what statements should be prioritized is discussed next.

Popper describes the scientific process following the creation of a novel hypothesis as:

"First, there is the logical comparison of the conclusions among them-

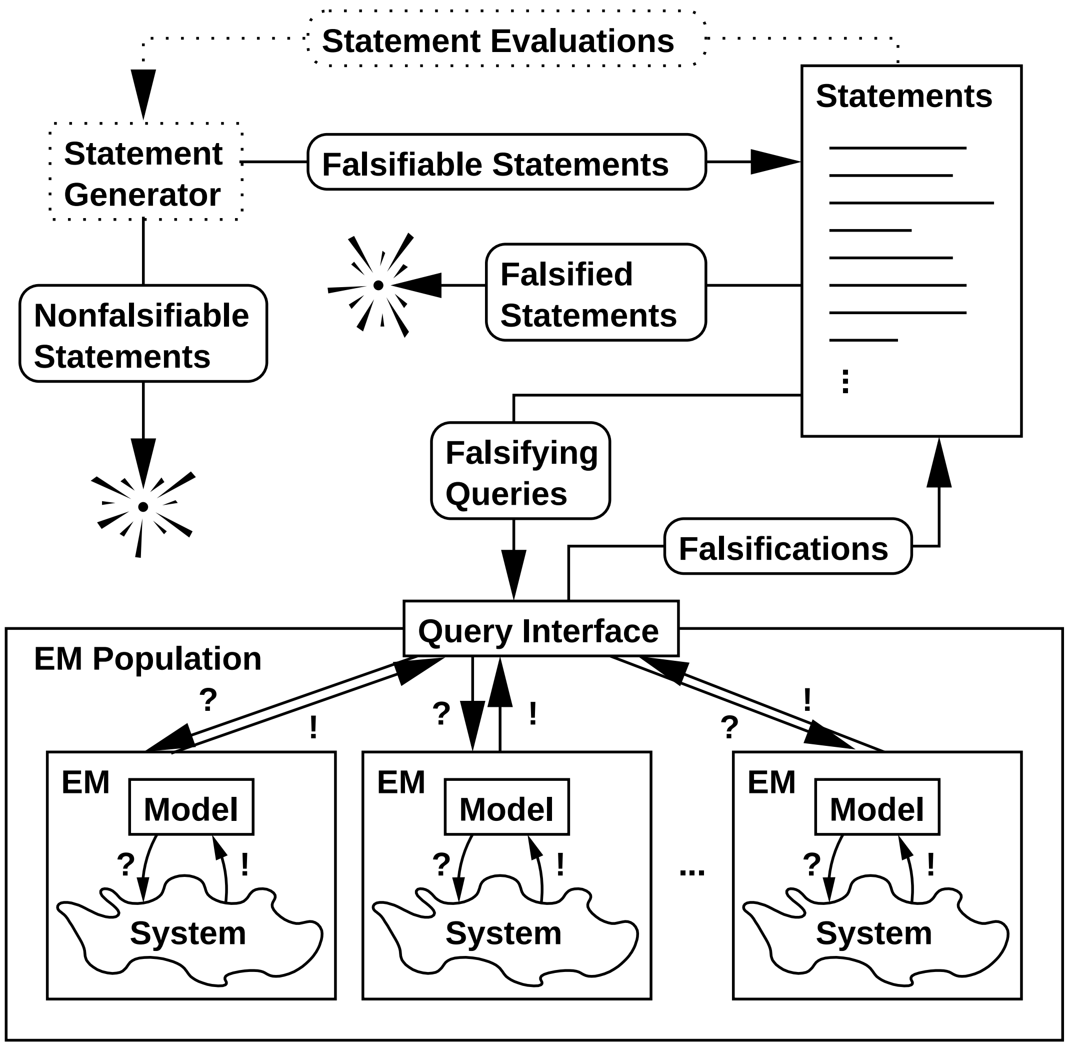


Figure 18.3: Outline of a Popperian Machine. A statement generator (which is undefined and could very well be a human user) feeds a statement list falsifiable statements about a set of underlying systems. The statements are reformulated as queries (aimed at falsifying the statements) to a set of Empirical Machines that interact with their associated underlying system in order to build models that can answer the queries. Falsified statements are then deleted from the list of statements. Over time, the list of statements should have increasingly higher empirical content, in terms of them being falsifiable, yet not falsified (Popper, 1990).

> selves, by which the internal consistency of the system is tested. Secondly, there is the investigation of the logical form of the theory, with the object of determining whether it has the character of an empirical or scientific theory, or whether it is, for example, tautological. Thirdly, there is the comparison with other theories, chiefly with the aim of determining whether the theory would constitute a scientific advance should it survive our various tests. And finally, there is the testing of the theory by way of empirical applications of the conclusions which can be derived from it." (Popper (1990), p. 32)

In accord with the suggested framework in Figure 18.3 I maintain that many aspects of what Popper considers a scientific process could be automated. How to implement the logic required for the deductive reasoning regarding, for example, "internal consistency" and "logical form" is not in the scope of this thesis. But such matters are highly central in the field of Inductive Logic Programming (Muggleton & Raedt, 1994), since it involves generating (guessing) statements that are of internal consistency and of particular logical forms. The fourth step, "the testing of the theory" is, in my suggested framework, the responsibility of the Empirical Machine.

The aspect of falsifiability becomes relevant in the third step, i.e. in the assessment to which degree a statement constitutes a scientific advance. How to exactly define and implement the assessment of falsifiability itself is also a grand issue beyond the scope of this thesis. I would, however, suggest some basic directions. Firstly, some statements are inherently unfalsifiable by their nature (e.g., tautologies). Others require enormous resources in order to test them, which thereby renders them less falsifiable. Other statements are open-ended since they involve infinity. For example, if the statement "Transitions over symbol **a** from SE q_1 will always lead to the same SE$'$ is not falsified after 1000 consecutive **a**s, should 1000 more be tested?

There will be degrees of falsifiability as well as degrees of how much falsification has been attempted through experiments targeted at a specific statement. The concept of "degrees of falsifiability" occupies large portions of Popper (1990). There is also a number of possibilities of how to exactly formalize and implement the assessment of the falsifiability of statements. I will not attempt to suggest any particular strategy for the general case. For example, Popper proposes that the "*logical probability of a statement is complementary to its degree of falsifiability*"

(Popper (1990), p. 119). In other words, if it seems very probable that a statement *will* be falsified through observations, it should thus be considered falsifiable. How exactly this logical probability is assessed is, however, likely to depend on the underlying logical language in use.

Within the context of SSMs and underlying SDTDSs, however, I would suggest that falsifiability could quite easily be translated into *universality* and *precision* (Popper (1990), section 36). Universality and precision are described by Popper as the two outstanding demands for statements with the highest possible empirical content. A statement is more universal than another if it applies to more situations. A statement is more precise than another if it forbids more outcomes in those situations. For example, a statement about all days of the week is more universal than a statement only about Mondays. And a statement that on the referred days 100% of all people drink coffee is more precise than one stating that only at least 80% drink coffee, since the latter allows more observations without falsifying it[7].

Universality of SSM-statements could be translated into the number of situations for which a statement applies, i.e. the number of systems for which a statement applies, or number of SEs. Precision could be translated into an assessment of how well the statement constrains the behaviour of the system into a limited set of possibilities. Universal statements will thereby be more falsifiable since more systems and situations would occur in which the statement can be falsified. Precise statements would be more falsifiable since fewer of the probable observed situations will allow the statement to be considered unfalsified.

Universality and precision could potentially be competing goals. It is, for example, probably easier to give a very precise statement about a single system compared to one for a wide range of systems. For example, an SSM extracted from a single SDTDS is a very precise (and falsifiable) statement that "This SSM describes how this specific SDTDS behaves". The full range of generated statements should thus ideally cover a spectrum of universal and precise statements (cf. Figure 18.4).

By promoting universality and precision alone, short and simple statements should become more prevalent than complex ones. A statement about *all* systems requires no lengthy explicit list of what systems it applies to, for example. And pre-

[7]To falsify the statements you need, for the first one, only observe that one person does not drink coffee, whereas the other one require observations of at least 20% of the population.

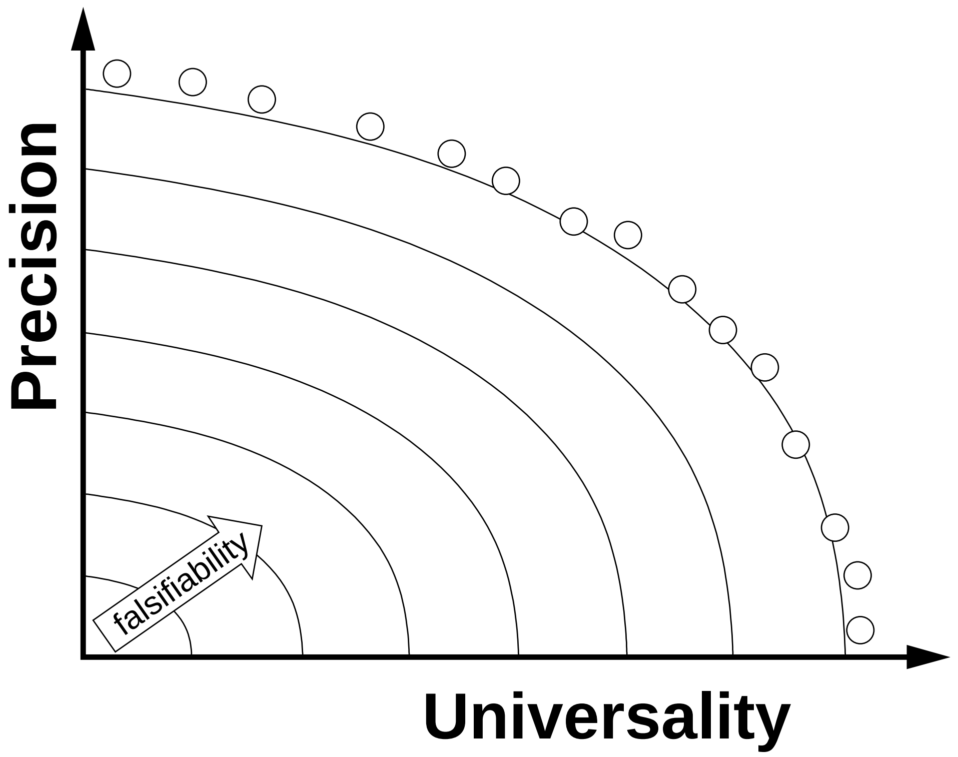


Figure 18.4: Falsifiability as universality and precision. Statements about systems (cf. Figure 18.3) occupy points on the axes (illustrated by the circles) and would ideally cover a spectrum of from precise to universal ones.

cise statements should describe as few allowed situations as possible. The simplicity could also be further, and implicitly, promoted in the generation of statements:

> "Simple statements, if knowledge is our object, are to be prized more highly than less simple ones *because they tell us more; because their empirical content is greater; and because they are better testable.*" (Popper (1990), p. 142).

With regard to the simplicity of statements, I would again argue that Popper's philosophy is ideal for machine learning. His chapter on simplicity (Chapter 7 in Popper (1990)) includes, for example, the section "*Elimination of the Aesthetic and the Pragmatic Concepts of Simplicity*". Simplicity is a very central theme in epistemology, yet with few successful logical definitions of the concept, according to Popper. His approach is, quite naturally, to relate simplicity with falsifiability in an attempt to find a nonanthropomorphic definition of the concept.

The Popperian Machine could fit very well into the context of rule extraction since it would not only induce rules from underlying systems, but also statements about the systems that are based on a sound scientific principle and well tested.

Moreover, the extracted rules themselves would reflect and support these scientifically guided statements. The whole ambition is to maximize the empirical contents of the statements and thereby also of the underlying rules. Therefore I suggest that Popperian Machines are an important future direction for the field of rule extraction.

18.7 Chapter summary

In this chapter I argue that automated analysis of simulated systems is both promising and required. Promising in the sense that the ease of observability and manipulability is unmatched in reality (cf. Section 18.1), and required in the sense that these system may be large, complex, counter-intuitive and numerous since creation of simulated systems is easy (cf. Section 18.2). The researcher per system ratio is low today and likely to decline. In my opinion, the basic reasons for automating anything are *necessity* and *possibility*. These criteria are certainly fulfilled when it comes to automated analysis of simulated systems. The analysis of simulated systems should be automated not because the most significant research questions are found in them, but because the process is too expensive for humans to do it. The volume and insignificance of the many individual simulations renders them too unrewarding for human reasoning.

In Section 18.3 I also challenge the notion of comprehensibility as the primary motivation for RE (cf. Section 4.2.4). A model has more virtues than being readable by humans. Many simulators themselves are good examples of these. A weather simulator is, for example, very complex but acts as a proxy for queries about the actual weather (which is even more complex). The weather presenter in turn acts as a proxy for the simulator, giving us a presentation that laymen may understand. Although the weather simulator itself is very complex and incomprehensible to most of us, it generates a result we may understand and appreciate; a weather forecast. Similarly, rules extracted from an SDTDS may be incomprehensibly complex, but it acts as a model with a clearly defined syntax of which queries can be asked.

In Sections 18.4–18.6 two abstract frameworks for future RNN-RE research were suggested. These frameworks are suggested on the basis that fidelity should be considered more important than comprehensibility (cf. Section 4.2.4) since models that correctly mimic the underlying system should generate better answers to queries about the system. Empirical Machines are proposed as active learners that target the ignorance of their best models in order to gather interesting data from the system through interaction (i.e. experimentation). Based on a philosophical discussion of Popper in relation to machine learning and automated scientific discovery (in Section 18.5), it is suggested Popperian Machines provide a scientifically based selection that guides the Empirical Machines towards scrutinizing statements of high empirical content. These statements (i.e. theories) about populations of simulated systems, that should be falsifiable but not falsified, is the desired output of the Popperian Machines.

Chapter 19

Summary and Final Thoughts

19.1 Contribution highlights

The contributions of this thesis are distributed in its three parts: the first provides an account of the history of the field, the second makes a contribution to the field, and the final part views the field from a new, more speculative perspective and suggests future directions.

The goal of Part I is to provide structure to the RNN-RE field through a taxonomy and review of earlier techniques. In Part II `CrySSMEx` is suggested as an alternative to the reviewed techniques. It is not only a new technique, but is also separated from the pattern of the previous techniques by integrating elements that were separated earlier. In Part III, not only some more or less speculative ideas for future work are suggested, but also concepts that question the very idea of rule extraction by viewing it as an automated scientific process.

To summarize, the main contributions of this thesis are:

- Part I: A taxonomy for RNN-RE to organize the field of RNN-RE and to suggest some possible common goals for the field.
  - A taxonomy of RNN-RE techniques.
  - A collection of references of (hopefully) all earlier RNN-RE papers.
  - A historical account of how RNN-RE has developed as a field.
  - A description of RNN-RE separated into four constituents: quantization, observation, construction and minimization.

- Part II: The `CrySSMEx`-algorithm which distinguishes itself from all earlier RNN-RE approaches in several ways.
  - The first integration of quantization, observation, construction and minimization into one algorithm.
  - The SSM as a new form of an extracted model.
  - The CVQ as a novel quantization algorithm which is both divisive and agglomerative.
  - The source code, and its open source availability[1], is itself a contribution which unfortunately is too technically detailed to be dealt with more thoroughly in this thesis[2].
- Part III: New connections to other fields and future directions are suggested.
  - A connection of RNN-RE is made to other fields of machine learning (and of control theory etc.).
  - More than ten possible improvements (some of which have actually been implemented) and approximately 20 challenges for RNN-RE and `CrySSMEx` are suggested.
  - A motivation for the automation of scientific analysis of simulated systems is given.
  - Empirical and Popperian Machines are suggested.

While the thesis began with references to Plato and ended with references to Popper, the contribution that should be emphasized above all the others, and which is very far from an abstract philosophical discussion is `CrySSMEx` and its implementation.

19.2 Final thoughts

RNNs, and simulated systems in general, are, since they are simulated entities, very "studyable" once we have the tools to study them (cf. the "golden properties"

[1]On `cryssmex.sourceforge.net`.

[2]It may be worth mentioning that I spent far more time on the implementation of `CrySSMEx` than on the thesis text and the presented experiments combined. This implementation involved solving some interesting problems (e.g., the Voronoi diagram plotter used to generate Figures 12.2 and 12.3, which can plot within arbitrarily merged Voronoi compartments) that unfortunately never made it into the thesis.

of Section 18.1). Furthermore, the algorithms reviewed in this thesis, together with `CrySSMEx`, may hold the seed of a deeper and more general notion of analysis than previously employed for RNNs. Better analysis tools may in turn help RNN research to progress more rapidly once we obtain a deeper understanding of what the networks are actually doing. After all, in many other disciplines of science, the quantum leaps in progress often stem from more sophisticated analysis tools and measuring devices producing qualitatively new data conflicting with existing models (anomalies) that eventually may result in scientific revolutions (Kuhn, 1962). Today we have deep, though partially conflicting theories of what the RNNs will be able to do in practice (i.e. the Turing machine equivalence vs. the difficulty of acquiring correct behaviour through learning), but we have no means of evaluating in an efficient manner what particular instances of RNNs are actually doing.

With critical eyes, rule extraction from recurrent neural networks may seem an infinitesimal subfield within another infinitesimal subfield and thereby it has very limited potential to deliver interesting scientific results. But if there were a future microscope for zooming in on RNNs, I would maintain that there are good reasons to believe rule extraction mechanisms are the operational parts, or "lenses", of that microscope. And as any real-world microscope, this RNN-microscope will, if general enough, be able to zoom in on other types of simulated dynamic systems and thus contribute to the scientific community in a considerably broader sense. Not in the sense that the biggest research questions are found in these systems, the reason for automating the simulated system analysis is precisely the opposite; it is simply too expensive to let humans do it when the systems are individually too uninteresting and when the number of them per researcher grows too large. The Empirical and Popperian Machines are suggested with this in mind. My hope is that the ideas suggested in this final part of the thesis will help populating the artificial "Plato caves" (cf. Chapter 1) with prisoners that have epistemic hunger[3] and the capability to explain their most informative conclusions about their universes to the creators of these universes.

[3]I.e. curiosity (Dennett, 1996, p. 92).

References

Allen, J. F. (2001a). Bioinformatics and discovery: induction beckons again. *BioEssays*, *23*(1), 104–107.

Allen, J. F. (2001b). In silico veritas – data-mining and automated discovery: the truth is in there. *EMBO reports*, *2*(7), 542–544.

Alquézar, R. & Sanfeliu, A. (1994a). A hybrid connectionist symbolic approach to regular grammar inference based on neural learning and hierarchical clustering. In *Proceedings of ICGI'94* (pp. 203–211).

Alquézar, R. & Sanfeliu, A. (1994b). Inference and recognition of regular grammars by training recurrent neural networks to learn the next-symbol prediction task. In F. Casacuberta & A. Sanfeliu (Eds.), *Advances in pattern recognition and applications: Selected papers from the Vth spanish symposium on pattern recognition and image analysis* (pp. 48–59). Singapore: World Scientific.

Alquézar, R., Sanfeliu, A. & Sainz, M. (1997). Experimental assessment of connectionist regular inference from positive and negative examples. In *VII simposium nacional de reconocimiento de formas y análisis de imágenes* (Vol. 1, pp. 49–54). Barcelona, Spain.

Andrews, R., Diederich, J. & Tickle, A. B. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, *8*(6), 373–389.

Angluin, D. (1981). A note on the number of queries needed to identify regular languages. *Information and Control*, *51*(1), 76–87.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, *75*, 87–106.

Angluin, D. (2004). Queries revisited. *Theoretical Computer Science*, *313*(2), 175–194.

Bakker, B. (2004). *The state of mind: Reinforcement learning with recurrent neural networks.* Phd thesis, Unit of Cognitive Psychology, Leiden University.

Bakker, B. & Jong, M. de. (2000). The epsilon state count. In J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat & S. Wilson (Eds.), *From animals to animats 6: Proceedings of the sixth international conference on simulation of adaptive behavior* (pp. 51–60). Cambridge, MA: MIT Press.

Barreto, G. A., Araújo, A. F. R. & Kremer, S. C. (2003). A taxonomy for spatiotemporal connectionist networks revisited: The unsupervised case. *Neural Computation*, *15*(6), 1255–1320.

Bengio, Y., Simard, P. & Frasconi, P. (1994, March). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Bergadano, F. & Gunetti, D. (1996). Testing by means of inductive program learning. *ACM Transactions on Software Engineering and Methodology*, *5*(2), 119–145.

Blair, A. & Pollack, J. (1997). Analysis of dynamical recognizers. *Neural Computation*, *9*(5), 1127–1142.

Blanco, A., Delgado, M. & Pegalajar, M. C. (2000). Extracting rules from a (fuzzy/crisp) recurrent neural network using a self-organizing map. *International Journal of Intelligent Systems*, *15*, 595–621.

Bodén, M. & Blair, A. (in press). Learning the dynamics of embedded clauses. *Applied Intelligence: Special issue on natural language and machine learning.*

Bodén, M., Jacobsson, H. & Ziemke, T. (2000). Evolving context-free language predictors. In D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee & H.-G. Beyer (Eds.), *Proceedings of the genetic and evolutionary computation conference* (pp. 1033–1040). San Fransisco: Morgan Kaufmann.

Bodén, M. & Wiles, J. (2000). Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*, *12*(3/4), 196–210.

Bodén, M. & Wiles, J. (2002). On learning context free and context sensitive languages. *IEEE Transactions on Neural Networks*, *13*(2), 491–493.

Bodén, M., Wiles, J., Tonkes, B. & Blair, A. (1999). Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units. In *Proceedings of ICANN 99* (pp. 359–364). Edinburgh: IEEE.

Bruske, J. & Sommer, G. (1995). Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, *7*, 845–865.

Bryant, C. H., Muggleton, S. H., Page, C. D. & Sternberg, M. J. E. (1999). Combining active learning with inductive logic programming to close the loop in machine learning. In *Proceedings of the AISB'99 symposium on AI and scientific creativity.* (informal proceedings)

Bullinaria, J. A. (1997). Analyzing the internal representations of trained artificial neural networks. In A. Browne (Ed.), *Neural network analysis, architectures and applications* (pp. 3–26). IOP Publishing.

Carrasco, R. C. & Forcada, M. L. (2001). Simple strategies to encode tree automata in sigmoid recursive neural networks. *IEEE Transactions on Knowledge and Data Engineering*, *13*(2), 148–156.

Carrasco, R. C., Forcada, M. L., Muñoz, M. A. V. & Ñeco, R. P. (2000). Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. *Neural Computation*, *12*(9), 2129–2174.

Casey, M. (1996). The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, *8*(6), 1135–1178.

Cechin, A. L., Pechmann Simon, D. R. & Stertz, K. (2003). State automata extraction from recurrent neural nets using k-means and fuzzy clustering. In *XXIII international conference of the Chilean computer science society* (pp. 73–78).

Chaitin, G. J. (1987). *Algorithmic information theory.* Cambridge University Press.

Chaitin, G. J. (2005, June). *Epistemology as information theory from Leibniz to* Ω. Alan Turing Lecture on Computing and Philosophy, E-CAP'05, European Computing and Philosophy Conference, Mälardalen University, Västerås.

Chalup, S. K. & Blair, A. D. (2000). *First order recurrent neural networks learn to predict a mildly context-sensitive language* (TR 2000-06 No. ISBN 0-7259-1109-3). Department of Computer Science and Software Engineering, The

University of Newcastle.

Christiansen, M. H. & Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, *23*(2), 157–205.

Cicchello, O. & Kremer, S. C. (2003). Inducing grammars from sparse data sets: A survey of algorithms and results. *Journal of Machine Learning Research*, *4*, 603–632.

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

Cohn, D. A. (1994). Neural network exploration using optimal experiment design. In J. D. Cowan, G. Tesauro & J. Alspector (Eds.), *Advances in neural information processing systems* (Vol. 6, pp. 679–686). Morgan Kaufmann Publishers, Inc.

Cohn, D. A., Atlas, L. & Ladner, R. E. (1994). Improving generalization with active learning. *Machine Learning*, *15*(2), 201-221.

Colton, S., Bundy, A. & Walsh, T. (2000). On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies*, *53*(3), 351–375.

Colton, S. & Steel, G. (1999). Artificial intelligence and scientific creativity. *Artificial Intelligence and the Study of Behaviour Quarterly*, *102*.

Cover, T. M. & Thomas, J. A. (1990). *Elements of information theory.* John Wiley, New York.

Craven, M. W. & Shavlik, J. W. (1994). Using sampling and queries to extract rules from trained neural networks. In W. W. Cohen & H. Hirsh (Eds.), *Machine learning: Proceedings of the eleventh international conference.* San Fransisco, CA: Morgan Kaufmann.

Craven, M. W. & Shavlik, J. W. (1996). Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*, *8*, 24–30.

Craven, M. W. & Shavlik, J. W. (1999). *Rule extraction: Where do we go from here?* (Tech. Rep. No. Machine Learning Research Group Working Paper 99-1). Department of Computer Sciences, University of Wisconsin.

Crutchfield, J. P. (1994). The calculi of emergence: Computation, dynamics, and induction. *Physica D*, *75*, 11–54.

Crutchfield, J. P. & Young, K. (1990). Computation at the onset of chaos. In W. Zurek (Ed.), *Complexity, entropy and the physics of information.* Addison-Wesley, Reading, MA.

Das, S. & Das, R. (1991). Induction of discrete-state machine by stabilizing a simple recurrent network using clustering. *Computer Science and Informatics*, *21*(2), 35–40.

Das, S., Giles, C. L. & Sun, G. Z. (1993). Using prior knowledge in a NNPDA to learn context-free languages. In S. J. Hanson, J. D. Cowan & C. L. Giles (Eds.), *Advances in neural information processing systems* (Vol. 5, pp. 65–72). Morgan Kaufmann, San Mateo, CA.

Das, S. & Mozer, M. (1998). Dynamic on-line clustering and state extraction: an approach to symbolic learning. *Neural Networks*, *11*(1), 53–64.

Das, S. & Mozer, M. C. (1994). A unified gradient-descent/clustering architecture for finite state machine induction. In J. D. Cowan, G. Tesauro & J. Alspector (Eds.), *Advances in neural information processing systems* (Vol. 6, pp. 19–26).

Morgan Kaufmann Publishers, Inc.

de la Higuera, C. (2005). A bibliographical study of grammatical inference. *Pattern Recognition*, *38*, 1332–1348.

Dennett, D. C. (1996). *Kinds of mind.* New York: Basic Books.

Devaney, R. L. (1992). *A first course in chaotic dynamical systems.* Addison-Wesley.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Everitt, B. S., Landau, S. & Leese, M. (2001). *Cluster analysis.* London: Arnold.

Fanelli, R. (1993). *Grammatical inference and approximation of finite automata by elman type recurrent neural networks trained with full forward error propagation* (Tech. Rep. No. NNRG930628A). Dept. of Physics, Brooklyn College of the City University of New York.

Föllesdal, D., Walløe, L. & Elster, J. (1993). *Argumentationsteori, språk och vetenskapsfilosofi.* Thales.

Forcada, M. L. (2002, January). *Neural networks: Automata and formal models of computation.* (`http://www.dlsi.ua.es/~mlf/nnafmc/`, accessed March 31, 2006)

Forcada, M. L. & Carrasco, R. C. (2001). Finite-state computation in analog neural networks: steps towards biologically plausible models? In S. Wermter, J. Austin & D. Willshaw (Eds.), *Emergent computational models based on neuroscience* (pp. 482–486). Springer-Verlag.

Fox, M., Ghallab, M., Infantes, G. & Long, D. (2006). Robot introspection through learned hidden Markov models. *Artificial Intelligence*, *170*, 59–113.

Frasconi, P., Gori, M., Maggini, M. & Soda, G. (1996). Representation of finite state automata in recurrent radial basis function networks. *Machine Learning*, *23*(1), 5–32.

Friedman, N. & Halpern, J. Y. (1999). Belief revision: A critique. *Journal of Logic, Language, and Information*, *8*, 401–420.

Garg, V. K., Kumar, R. & Marcus, S. I. (1999). Probabilistic language formalism for stochastic discrete event systems. *IEEE Transactions on Automatic Control*, *44*, 280–293.

Gers, F. A. & Schmidhuber, J. (2001). LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, *12*(6), 1333–1340.

Giles, C. L., Chen, D., Miller, C., Chen, H., Sun, G. & Lee, Y. (1991). Second-order recurrent neural networks for grammatical inference. In *Proceedings of international joint conference on neural networks* (Vol. 2, pp. 273–281). Seattle, Washington: IEEE Publication.

Giles, C. L., Horne, B. G. & Lin, T. (1995). Learning a class of large finite state machines with a recurrent neural network. *Neural Networks*, *8*(9), 1359–1365.

Giles, C. L., Lawrence, S. & Tsoi, A. (1997). Rule inference for financial prediction using recurrent neural networks. In *Proceedings of IEEE/IAFE conference on computational intelligence for financial engineering (CIFEr)* (pp. 253–259). Piscataway, NJ: IEEE.

Giles, C. L., Lawrence, S. & Tsoi, A. C. (2001, July/August). Noisy time series prediction using a recurrent neural network and grammatical inference. *Machine Learning*, *44*(1/2), 161–183.

Giles, C. L., Miller, C. B., Chen, D., Chen, H. H. & Sun, G. Z. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, *4*(3), 393–405.

Giles, C. L., Miller, C. B., Chen, D., Sun, G. Z., Chen, H. H. & Lee, Y. C. (1992). Extracting and learning an unknown grammar with recurrent neural networks. In J. E. Moody, S. J. Hanson & R. P. Lippmann (Eds.), *Advances in neural information processing systems* (Vol. 4, pp. 317–324). Morgan Kaufmann Publishers, Inc.

Giles, C. L. & Omlin, C. W. (1993). Extraction, insertion and refinement of symbolic rules in dynamically driven recurrent neural networks. *Connection Science*, *5*(3 – 4), 307–337.

Giles, C. L. & Omlin, C. W. (1994). Pruning recurrent neural networks for improved generalization performance. *IEEE Transactions on Neural Networks*, *5*(5), 848–851.

Gillies, D. A. (2001). Popper and computer induction. *Bioessays.*

Gold, M. E. (1967). Language identification in the limit. *Information and Control*, *10*(5), 447–474.

Golea, M. (1996). *On the complexity of rule extraction from neural networks and network-querying* (Tech. Rep.). Canberra, Australia: Australian National University.

Gori, M., Maggini, M., Martinelli, E. & Soda, G. (1998, May). Inductive inference from noisy examples using the hybrid finite state filter. *IEEE Transactions on Neural Networks*, *9*(3), 571–575.

Gori, M., Maggini, M. & Soda, G. (1994, August). Scheduling of modular architectures for inductive inference of regular grammars. In *ECAI'94 workshop on combining symbolic and connectionist processing* (pp. 78–87). Wiley.

Goudreau, M. W. & Giles, C. L. (1995). Using recurrent neural networks to learn the structure of interconnection networks. *Neural Networks*, *8*(5), 793-804.

Goudreau, M. W., Giles, C. L., Chakradhar, S. T. & Chen, D. (1994). First-order vs. second-order single layer recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(3), 511–518.

Hammer, B. & Tiňo, P. (2003). Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, *15*(8), 1897–1929.

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, *46*(1–2), 47–75.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hopcroft, J. & Ullman, J. D. (1979). *Introduction to automata theory, languages, and compilation.* Addison-Wesley Publishing Company.

Horgan, J. (1996). *The end of science.* Little, Brown and Company.

Horne, B. G. & Giles, C. L. (1995). An experimental comparison of recurrent neural networks. In G. Tesauro, D. Touretzky & T. Leen (Eds.), *Advances in neural information processing systems 7* (pp. 697–704). MIT Press.

Horne, B. G. & Hush, D. R. (1994). Bounds on the complexity of recurrent neural network implementations of finite state machines. In J. D. Cowan, G. Tesauro & J. Alspector (Eds.), *Advances in neural information processing systems* (Vol. 6, pp. 359–366). Morgan Kaufmann Publishers, Inc.

Husbands, P., Harvey, I. & Cliff, D. T. (1995). Circle in the round: State space attractors for evolved sighted robots. *Robotics and Autonomous Systems*, *15*(1-2), 83–106.

Jacobsson, H. (1998). *Inversion of an artificial neural network mapping by evolutionary algorithms with sharing.* Bachelor's thesis, University of Skövde (report number: HS-IDA-EA-98-113).

Jacobsson, H. (1999). *A comparison of simple recurrent and sequential cascaded networks for formal language recognition.* Master's thesis, University of Skövde (report number: HS-IDA-MD-99-005).

Jacobsson, H. (2005). Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, *17*(6), 1223–1263.

Jacobsson, H. (2006). The crystallizing substochastic sequential machine extractor - `CrySSMEx`. *Neural Computation*, *18*(9), 2211–2255.

Jacobsson, H. & Olsson, B. (2000). An evolutionary algorithm for inversion of artificial neural networks. In P. P. Wang (Ed.), *Proceedings of the fifth joint conference on information sciences* (pp. 1070–1073). Association for Intelligent Machinery.

Jacobsson, H. & Ziemke, T. (2003a). Improving procedures for evaluation of connectionist context-free language predictors. *IEEE Transactions on Neural Networks*, *14*(4), 963–966.

Jacobsson, H. & Ziemke, T. (2003b). *Reducing complexity of rule extraction from prediction RNNs through domain interaction* (Tech. Rep. No. HS-IDA-TR-03-007). Department of Computer Science, University of Skövde, Sweden.

Jacobsson, H. & Ziemke, T. (2005a). Rethinking rule extraction from recurrent neural networks. In A. d'Avila Garcez, J. Elman & P. Hitzler (Eds.), *IJCAI-05 workshop on neural-symbolic learning and reasoning.*

Jacobsson, H. & Ziemke, T. (2005b). `CrySSMEx`, a novel rule extractor for recurrent neural networks : Overview and case study. In W. Duch, J. Kacprzyk, E. Oja & S. Zadrozny (Eds.), *Artificial neural networks: Formal models and their applications - ICANN 2005 - part II* (pp. 503–508). Berlin: Springer.

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In S. T. S. Becker & K. Obermayer (Eds.), *Advances in neural information processing systems 15* (pp. 593–600). Cambridge, MA: MIT Press.

Jaeger, H. & Haas, H. (2004, April). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, *5667*(304), 78–80.

Jagota, A., Plate, T., Shastri, L. & Sun, R. (1999). Connectionist symbol processing: Dead or alive? *Neural Computing Surveys*, *2*, 1–40.

Jain, A. K., Murty, M. N. & Flynn, P. J. (1999, September). Data clustering: A review. *ACM Computing Surveys*, *31*(3), 264–323.

King, R. D., Whelan, K. E., Jones, F. M., Reiser, P. G. K., Bryant, C. H., Muggleton, S. H. et al. (2004, January). Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, *427*(6971), 247–252.

Kohonen, T. (1995). *Self-organizing maps.* Berlin, Heidelberg: Springer.

Kolen, J. F. (1993). Fool's gold: Extracting finite state machines from recurrent network dynamics. In J. Cowan, G. Tesauro & J. Alspector (Eds.), *Neural information processing systems 6* (pp. 501–508). San Francisco, CA: Morgan Kaufmann.

Kolen, J. F. (1994a). *Exploring the computational capabilities of recurrent neu-*

ral networks. Unpublished doctoral dissertation, The Ohio State University, Department of Computer and Information Sciences.

Kolen, J. F. (1994b). Recurrent networks: State machines or iterated function systems? In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman & A. S. Weigend (Eds.), *Proceedings of the 1993 connectionist models summer school.* Hillsdale, NJ: Lawrence Erlbaum.

Kolen, J. F. & Kremer, S. C. (Eds.). (2001). *A field guide to dynamical recurrent networks.* IEEE Press.

Kolen, J. F. & Pollack, J. (1995). The observers' paradox: Apparent computational complexity in physical systems. *Journal of Exp. and Theoret. Artificial Intelligence*, *7*(3).

Korb, K. B. (2004). Machine learning as philosophy of science. *Minds and Machines*, *14*(4), 433–440.

Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J. & Lanza, G. (2003). *Genetic programming IV: Routine human-competitive machine intelligence.* Boston, MA: Kluwer Academic Publishers.

Kremer, S. C. (2001). Spatiotemporal connectionist networks: A taxonomy and review. *Neural Computation*, *13*(2), 248–306.

Krogh, A. & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. S. Touretzky & T. K. Leen (Eds.), *Advances in neural information processing systems* (Vol. 7, pp. 231–238). Cambridge, MA: MIT Press.

Kuhn, T. S. (1962). *The structure of scientific revolutions.* Chicago: University of Chicago Press.

Kumar, R. & Garg, V. K. (2001). Control of stochastic discrete event systems modeled by probabilistic languages. *IEEE Transactions on Automatic Control*, *46*(4), 593–606.

Lang, K. J. (1992). Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of the fifth ACM workshop on computational learning theory* (pp. 45–52). New York.

Langley, P. (1998). The computer-aided discovery of scientific knowledge. In *First international conference on discovery science* (pp. 25–39). Fukuoka, Hapan: Springer.

Langley, P. (2000). The computational support of scientific discovery. *International Journal of Human-Computer Studies*, *53*, 393–410.

Langley, P. (2002). Lessons for the computational discovery of scientific knowledge. In *Proceedings of first international workshop on data mining lessons learned* (pp. 9–12). Sydney.

Langley, P., Shrager, J. & Saito, K. (2002). Computational discovery of communicable scientific knowledge. In L. Magnani, N. J. Nersessian & C. Pizzi (Eds.), *Logical and computational aspects of model-based reasoning.* Dordrecht: Kluwer Academic.

Lawrence, S., Giles, C. L. & Fong, S. (2000). Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, *12*(1), 126–140.

Lawrence, S., Giles, C. L. & Tsoi, A. C. (1998). Symbolic conversion, grammatical inference and rule extraction for foreign exchange rate prediction. In Y. Abu-Mostafa, A. S. Weigend & P. Refenes (Eds.), *Neural networks in the capital* *markets nncm96* (pp. 333–345). Singapore: World Scientific Press.

Linåker, F. & Jacobsson, H. (2001a). Learning delayed response tasks through unsupervised event extraction. *International Journal of Computational Intelligence and Applications.*, *1*(4), 413–426.

Linåker, F. & Jacobsson, H. (2001b). Mobile robot learning of delayed response tasks through event extraction: A solution to the road sign problem and beyond. In B. Nebel (Ed.), *Proceedings of the seventeenth international joint conference on artificial intelligence, IJCAI-2001* (pp. 777–782). San Fransisco: Morgan Kaufmann.

Ljung, L. (1999). *System identification: theory for the user.* Prentice Hall.

MacKay, D. (1992). Information-based objective functions for active data selection. *Neural Computation*, *4*(4), 590–604.

Maggini, M. (1998). Recursive neural networks and automata. In C. L. Giles & M. Gori (Eds.), *Adaptive processing of sequences and data structures* (pp. 248–295). Springer-Verlag.

Manolios, P. & Fanelli, R. (1994). First order recurrent neural networks and deterministic finite state automata. *Neural Computation*, *6*(6), 1155–1173.

Marculescu, D., Marculescu, R. & Pedram, M. (1996). Stochastic sequential machine synthesis targeting constrained sequence generation. In *DAC'96: Proceedings of the 33rd annual conference on design automation* (pp. 696–701). New York, NY, USA: ACM Press.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133.

Medler, D. (1998). A brief history of connectionism. *Neural Computing Surveys*, *1*(1), 61–101.

Meeden, L. A. (1996). An incremental approach to developing intelligent neural network controllers for robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *26*(3), 474–85.

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.

Mirkin, B. (1996). *Mathematical classification and clustering* (Vol. 11). Kluwer.

Moore, E. F. (1956). Gedanken-experiments on sequential machines. In C. E. Shannon & J. McCarthy (Eds.), *Annals of mathematical studies* (Vol. 34, pp. 129–153). Princeton University Press.

Muggleton, S. & Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19,20*, 629–679.

Niklasson, L. & Bodén, M. (1997). Representing structure and structured representations in connectionist networks. In A. Browne (Ed.), *Neural network perspectives on cognition and adaptive robotics* (pp. 20–50). IOP Press.

Omlin, C. W. (2001). Understanding and explaining DRN behaviour. In J. F. Kolen & S. C. Kremer (Eds.), *A field guide to dynamical recurrent networks* (pp. 207–228). IEEE Press.

Omlin, C. W., Giles, C. & Miller, C. (1992). Heuristics for the extraction of rules from discrete-time recurrent neural networks. In *Proceedings of the international joint conference on neural networks* (Vol. I, pp. 33–38). New York: International Neural Network Society, IEEE.

Omlin, C. W. & Giles, C. L. (1992). Training second-order recurrent neural networks using hints. In D. Sleeman & P. Edwards (Eds.), *Proceedings of the ninth international conference on machine learning* (pp. 363–368). San Mateo, CA: Morgan Kaufmann Publishers.

Omlin, C. W. & Giles, C. L. (1996a). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, *43*, 937–972.

Omlin, C. W. & Giles, C. L. (1996b). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, *9*(1), 41–51.

Omlin, C. W. & Giles, C. L. (1996c). Rule revision with recurrent neural networks. *Knowledge and Data Engineering*, *8*(1), 183-188.

Omlin, C. W. & Giles, C. L. (2000). Symbolic knowledge representation in recurrent neural networks: Insights from theoretical models of computation. In I. Cloete & J. M. Zurada (Eds.), *Knowledge-based neurocomputing.* MIT Press.

Omlin, C. W., Thornber, K. K. & Giles, C. L. (1998). Deterministic fuzzy finite state automata can be deterministically encoded into recurrent neural networks. *IEEE Transactions on Fuzzy Systems*, *6*(1), 76–89.

Paz, A. (1971). *Introduction to probabilistic automata.* Academic Press.

Pearl, J. (2000). *Causality : Models, reasoning, and inference.* Cambridge University Press.

Plato. (1991). *The republic - the complete and unabridged Jowett translation.* Vintage Classics.

Pollack, J. B. (1987). Cascaded back-propagation on dynamic connectionist networks. In *Proceedings of the 9th annual conference of the cognitive science society* (pp. 391–404). Hillsdale, NJ: Lawrence Erlbaum Associates.

Popper, K. R. (1935). *Logik der forschung.* Julius Springer Verlag.

Popper, K. R. (1990). *The logic of scientific discovery* (14 ed.). London: Unwin Hyman. (Originally published 1959)

Rabin, M. O. (1963). Probabilistic automata. *Information and Control*, *6*, 230–245.

Riegler, A. (1998). The end of science: Can we overcome cognitive limitations? *Evolution and Cognition*, *4*(1), 37–50.

Rodriguez, P. (1999). *Mathematical foundations of simple recurrent neural networks in language processing.* Unpublished doctoral dissertation, University of California, San Diego.

Rodriguez, P., Wiles, J. & Elman, J. L. (1999). A recurrent network that learns to count. *Connection Science*, *11*, 5–40.

Sanfeliu, A. & Alquézar, R. (1995). Active grammatical inference: a new learning methodology. In *Shape, structure and pattern recognition* (pp. 191–200). World Scientific Pub.

Schellhammer, I., Diederich, J., Towsey, M. & Brugman, C. (1998). Knowledge extraction and recurrent neural networks: An analysis of an Elman network trained on a natural language learning task. In D. M. W. Powers (Ed.), *Proceedings of the joint conference on new methods in language processing and computational natural language learning: NeMLaP3/CoNLL98* (pp. 73–78). Somerset, New Jersey: Association for Computational Linguistics.

Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, *4*(2), 234–242.

Schmidhuber, J., Gers, F. & Eck, D. (2002). Learning nonregular languages: A comparison of Simple Recurrent Networks and LSTM. *Neural Computation*, *14*(9), 2039–2041.

Servan-Schreiber, D., Cleeremans, A. & McClelland, J. L. (1989). Learning sequential structure in simple recurrent networks. In D. S. Touretzky (Ed.), *Advances in neural information processing systems* (Vol. 1, pp. 643–652). San Mateo, CA: Morgan Kaufmann.

Servan-Schreiber, D., Cleeremans, A. & McClelland, J. L. (1991). Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, *7*, 161–193.

Sharkey, A. J. C. (1996). [special issue]. Combining artificial neural nets: Ensemble approaches. *Connection Science*, *8*(3/4).

Sharkey, N. E. & Jackson, S. A. (1995). An internal report for connectionists. In R. Sun & L. A. Bookman (Eds.), *Computational architectures integrating neural and symbolic processes* (pp. 223–244). Kluwer, Boston.

Siegelmann, H. T. & Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, *50*(1), 132–150.

Sima, J. & Orponen, P. (2003). General purpose computation with neural networks: a survey of complexity theoretic results. *Neural Computation*, *15*, 2727–2778.

Simon, H. A. (1969). *The sciences of the artificial.* Cambridge, MA: MIT Press. (2nd edition)

Simon, H. A. (1973). Does scientific discovery have a logic? *Philosophy of Science*, *40*, 471–480.

Simon, H. A. (1992). Scientific discovery as problem solving. *International Studies in the Philosophy of Science*, *6*, 3–14.

Simon, H. A. (1995/96). Machine discovery. *Foundations of Science*, *1*(2), 171–200.

Stening, J., Jacobsson, H. & Ziemke, T. (2005). Imagination and abstraction of sensorimotor flow: Towards a robot model. In *AISB'05: Proceedings of the symposium on next generation approaches to machine consciousness - imagination, development, intersubjectivity and embodiment* (pp. 50–58). The Society for the Study of Artificial Intelligence and the Simulation of Behavior.

Sun, G. Z., Giles, C. L. & Chen, H. H. (1998). The neural network pushdown automation: Architecture, dynamics and learning. In C. Giles & M. Gori (Eds.), *Adaptive processing of sequences and data structures* (pp. 296–345). Springer.

Sun, R. & Giles, C. L. (Eds.). (2001). *Sequence learning: Paradigms, algorithms, and applications* (Vol. 1828). Springer.

Sun, R., Peterson, T. & Sessions, C. (2001). The extraction of planning knowledge from reinforcement learning neural networks. In *Proceedings of wirn'2001.* Heidelberg, Germany: Springer-Verlag.

Tabor, W. & Tanenhaus, M. (1999). Dynamical models of sentence processing. *Cognitive Science*, *24*(4), 491–515.

Tickle, A. B., Andrews, R., Golea, M. & Diederich, J. (1997). Rule extraction from artificial neural networks. In A. Browne (Ed.), *Neural network analysis, architectures and applications* (pp. 61–99). IOP Publishing.

Tickle, A. B., Andrews, R., Golea, M. & Diederich, J. (1998). The truth will come to light: directions and challenges in extracting the knowledge embedded within mined artificial neural networks. *IEEE Transactions on Neural Networks*, *9*(6), 1057–1068.

Tiňo, P., Dorffner, G. & Schittenkopf, C. (2000). Understanding state space organization in recurrent neural networks with iterative function systems dynamics.

In S. Wermter & R. Sun (Eds.), *Hybrid neural symbolic integration* (pp. 256–270). Springer Verlag.

Tiňo, P. & Hammer, B. (2003). Architectural bias in recurrent neural networks - fractal analysis. *Neural Computation*, *15*(8), 1931–1957.

Tiňo, P., Horne, B. G., Giles, C. L. & Collingwood, P. C. (1998). Finite state machines and recurrent neural networks – automata and dynamical systems approaches. In J. E. Dayhoff & O. Omidvar (Eds.), *Neural networks and pattern recognition* (pp. 171–220). Academic Press.

Tiňo, P. & Köteles, M. (1999). Extracting finite-state representations from recurrent neural networks trained on chaotic symbolic sequences. *IEEE Transactions on Neural Networks*, *10*(2), 284–302.

Tiňo, P., Čerňanský, M. & Beňušková, L. (2004). Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, *15*(1), 6–15.

Tiňo, P. & Vojtek, V. (1998). Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences. *Neural Network World*, *8*(5), 517–530.

Tiňo, P. & Šajda, J. (1995). Learning and extracting initial mealy automata with a modular neural network model. *Neural Computation*, *7*(4), 822–844.

Tomita, M. (1982). Dynamic construction of finite-state automata from examples using hillclimbing. In *Proceedings of fourth annual cognitive science conference* (pp. 105–108). Ann Arbor, MI.

Tonkes, B., Blair, A. & Wiles, J. (1998). Inductive bias in context-free language learning. In *Proceedings of the ninth Australian conference on neural networks* (pp. 52–56). Brisbane: Department of Computer Science and Electrical Engineering, University of Queensland.

Tonkes, B. & Wiles, J. (1999). Learning a context-free task with a recurrent neural network: An analysis of stability. In R. Heath, B. Hayes, A. Heathcote & C. Hooker (Eds.), *Dynamical cognitive science: Proceedings of the fourth biennial conference of the Australasian cognitive science society.* Newcastle.

Towell, G. G. & Shavlik, J. W. (1993). The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, *13*(1), 17–101.

Trakhtenbrot, B. A. & Barzdin, J. M. (1973). *Finite automata: behavior and synthesis.* Amsterdam: North-Holland.

Vahed, A. & Omlin, C. W. (1999). *Rule extraction from recurrent neural networks using a symbolic machine learning algorithm* (Tech. Rep. No. US-CS-TR-4). University of Stellenbosch, South Africa: Computer Science Department.

Vahed, A. & Omlin, C. W. (2004). A machine learning method for extracting symbolic knowledge from recurrent neural networks. *Neural Computation*, *16*, 59–71.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*(11), 1134-1142.

Watrous, R. L. & Kuhn, G. M. (1992). Induction of finite-state automata using second-order recurrent networks. In J. E. Moody, S. J. Hanson & R. P. Lippmann (Eds.), *Advances in neural information processing systems* (Vol. 4, pp. 309–317). Morgan Kaufmann Publishers, Inc.

Watson, R. A. & Pollack, J. B. (2005). Modular interdependency in complex dynamical systems. *Artificial Life*, 445–457.

Wiles, J. & Elman, J. L. (1995). 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* (pp. 482–487). Cambridge MA: MIT Press.

Williamson, J. (2004). A dynamic interaction between machine learning and the philosophy of science. *Minds and Machines*, *14*(4), 539–549.

Witkowski, M. (2002, August). Anticipatory learning: The animat as discovery engine. In *Adaptive behavior in anticipatory learning systems (ABiALS-02).* Edinburgh.

Young, K. & Crutchfield, J. P. (1993). Fluctuation spectroscopy. *Chaos, Solutions, and Fractals*, *4*, 5–39.

Young, S. & Garg, V. K. (1995). Model uncertainty in discrete event systems. *SIAM Journal on Control and Optimization*, *33*(1), 208–226.

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

Zhou, Z.-H. (2004). Rule extraction: using neural networks or for neural networks? *Journal of Computer Science and Technology*, *19*(2), 249–253.

Ziemke, T. (2000). On 'parts' and 'wholes' of adaptive behavior: Functional modularity and diachronic structure in recurrent neural robot controllers. In *From animals to animats 6 - proceedings of the sixth international conference on the simulation of adaptive behavior (SAB 2000).* Cambridge, MA: MIT Press.

Ziemke, T. & Thieme, M. (2002). Neuromodulation of reactive sensorimotor mappings as a short-term memory mechanism in delayed response tasks. *Adaptive Behavior*, *10*(3/4), 185–199.

Appendix A

Substochastic vectors

Some important types of, and operations on, substochastic vectors are defined below (some of these are also found in Paz (1971)):

Definition A.1 A *substochastic vector* $\vec{v}$ is a vector where all elements are nonnegative and the sum of the elements is ≤ 1. □

A special case of the substochastic distribution is where all probabilities are zero:

Definition A.2 An *exhausted substochastic vector* $\vec{v}$ is the special case of a substochastic vector where all elements are 0. □

And, as another special case, we find vectors with more conventional probabilistic properties:

Definition A.3 A *stochastic vector* $\vec{v}$ is the special case of a substochastic vector where the sum of the elements is exactly 1. □

And a special case of stochastic vectors is where only one element is probable:

Definition A.4 A *degenerate vector* is a stochastic vector one element with probability 1 and the rest 0. □

Definition A.5 The entropy of an n-dimensional substochastic vector $\vec{v}$ is here denoted as $H(\vec{v})$ and is calculated by

$$H(\vec{v}) = -\sum_{i=1}^{n} \vec{v}_i \log \vec{v}_i$$

□

By definition $0 \cdot \log 0 = 0$. Entropy is not really well defined for substochastic vectors, but in the algorithm of this thesis, entropy will only be calculated over stochastic or exhausted vectors. Therefore the entropy as described here will be according to proper theory (Cover & Thomas, 1990) unless the distribution is exhausted in which case function, here called entropy, will return zero.

Definition A.6 The function `normalize` is used to transform a substochastic vector into a stochastic vector, if possible, according to

$$\texttt{normalize}(\vec{v}) = \begin{cases} \frac{\vec{v}}{\sum_{i=1}^{n} \vec{v}_i} & \text{if } \sum_{i=1}^{n} \vec{v}_i > 0 \\ \vec{v} \cdot 0 & \text{otherwise} \end{cases}$$

□

Definition A.7 The *support set* of a substochastic vector $\vec{v} = (\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n)$ is the set $\{i : \vec{v}_i > 0\}$ and is denoted $\sup(\vec{v})$. □

Appendix B

List of abbreviations

`CrySSMEx`	Crystallizing SSM Extractor
CVQ	Crystalline Vector Quantizer
NDI-equivalence	Not Decisively-Inequivalent
RE	Rule Extraction
RNN	Recurrent Neural Network
RNN-RE	RNN specific RE
SDTDS	Situated Discrete Time Dynamic System
SE	State Element (of an SSM)
SSM	Substochastic Sequential Machine
UNDI-equivalent	Universally NDI-equivalent
VQ	Vector Quantizer
Λ	Quantizer function
Ω	Transition event set (from an SDTDS)

Table B.1: List of important abbreviations.

Appendix C

Jacobsson & Ziemke (2003a)

Improving Procedures for Evaluation of Connectionist Context-Free Language Predictors[1]

Henrik Jacobsson, Tom Ziemke

Abstract
This paper shows how seemingly minor differences in training and evaluation procedures used in recent studies of recurrent neural networks as context free language predictors can lead to significant differences in apparent network performance. We therefore suggest standard evaluation procedures whose use would facilitate better reproducability and comparability.

C.1 Introduction

A number of recent papers have investigated the use of Recurrent Neural Networks (RNNs) for predicting strings belonging to the class of the Context Free Language (CFL) $\mathbf{a}^n\mathbf{b}^n$ and the Context Sensitive Language (CSL) $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$ (Wiles & Elman, 1995; Tonkes et al., 1998; Rodriguez et al., 1999; Tonkes & Wiles, 1999; Bodén et al., 1999, 2000; Chalup & Blair, 2000; Bodén & Wiles, 2000; Gers & Schmidhuber, 2001; Bodén & Blair, in press; Schmidhuber et al., 2002). Each of these papers makes valuable contributions, but when we compared them, we noticed two problems: Firstly, sometimes a number of details of the evaluation method (for evaluating the generalization ability of the networks) were undocumented. Secondly, where details of evaluation were provided, minor differences between the methods used in different papers were found. This led us to carry out a series of experiments with the aim to systematically investigate whether these differences may affect the Estimated Generalization Ability (EGA) for a *given population* of RNNs. Such differences may be an indicator that the reproducability and comparability of the generalization ability presented in these papers might be questioned.

In our experiments we have varied three aspects of the testing procedure in order to see how the EGA of the RNNs is affected. These aspects are: Firstly, the

[1]This is a verbatim copy of Jacobsson and Ziemke (2003a). Only the formatting and contact information differs from the original (the bibliography is also not included here since it can be found elsewhere in the thesis).

string order, i.e. the order in which strings of different lengths from the grammar $\mathbf{a}^n\mathbf{b}^n$ are concatenated into the string which the RNN should predict. Secondly, the *maximum string length*, i.e. the highest value of n of the $\mathbf{a}^n\mathbf{b}^n$ strings in the test set. The third aspect, *error tolerance* is the degree to which the network is allowed to make mistakes. The reason that the two first aspects are important is that an RNN is a dynamical system with a potential sensitivity to its initial state which can be based on previous inputs. Variations of these three aspects exist in the above mentioned papers, but are in some cases just vaguely described, if at all. In addition to these three, other important aspects, such as the number of networks, number of repeated tests per network and basic definitions such as "success" are varied and in some cases quite vaguely described.

The structure of this paper is as follows: First the investigated papers are briefly summarized to give an overview of their experimental strategies. Then our experiments designed to evaluate the sensitivity of the EGA with respect to testing procedure are presented. The results of the survey and experiments are then fused into some concluding remarks and recommendations.

C.2 Background

The papers that present results of CFL and CSL predictions with RNNs and their testing approaches are summarized in Table C.1. The architectures focused on in these papers were Simple Recurrent Networks (SRNs) (Wiles & Elman, 1995; Tonkes et al., 1998; Rodriguez et al., 1999; Tonkes & Wiles, 1999; Bodén et al., 1999, 2000; Chalup & Blair, 2000), Sequential Cascaded Networks (SCNs) (Bodén et al., 2000; Bodén & Wiles, 2000; Bodén & Blair, in press) and Long Short-Term Memory (LSTM) (Gers & Schmidhuber, 2001; Schmidhuber et al., 2002). The training algorithms used in these papers are either based on gradient descent (Wiles & Elman, 1995; Tonkes et al., 1998; Rodriguez et al., 1999; Tonkes & Wiles, 1999; Bodén et al., 1999, 2000; Bodén & Wiles, 2000; Gers & Schmidhuber, 2001; Bodén & Blair, in press; Schmidhuber et al., 2002) and/or Evolutionary Hillclimbing (EH) (Tonkes et al., 1998; Bodén et al., 2000; Chalup & Blair, 2000). There are, of course, many other important papers in the field of CFL/CSL prediction and related fields, but those not presenting quantitative studies of the generalization ability have been omitted as they have no direct bearing on our results. Other papers in the field of CFL- and CSL-prediction have also been omitted to make comparisons simpler, i.e. only $\mathbf{a}^n\mathbf{b}^n$ and $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$ papers are included.

The training and test set sizes used in the cited papers are presented in Table C.1, as well as the ordering of strings in the test set. Where there has been any chance of misunderstanding the structure of the testing set/procedure, we have chosen not to make any assumptions. For example, when the test set is explained as "from depth 1 to 30" (Wiles & Elman, 1995) or "strings up to $n = 12$" (Bodén et al., 1999) it may be implicit that the strings are ordered in an ascending order, but as no explicit definition of string order is found, these papers are marked as being ambiguous about the test set order.

Among these papers, we found three different test set orderings: *random*, *ascending* and *descending* order. Six out of eleven papers did not explicitly define the order of their test set. The maximum string length of the test set also varied among the papers. Furthermore, the details of the error tolerance were usually not discussed, i.e. it was actually quite unclear in some of the papers whether correct prediction *once* per string occurrence was enough to consider the prediction successful or if the network needed to *consistently* predict all strings correctly. It seems, however, that the former is most commonly used.

It may also be worth noting that two papers (Gers & Schmidhuber, 2001; Schmidhuber et al., 2002) used slightly different domains, $\mathbf{a}^n\mathbf{b}^n\mathbf{T}$ and $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n\mathbf{T}$, which strictly speaking are not the same languages as $\mathbf{a}^n\mathbf{b}^n$ or $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$. The terminal symbol $\mathbf{T}$ gives the network a mechanism for resetting its state in a more deterministic manner than otherwise. The comparison across these domains may therefore not be reliable. Considering only comparisons *within* the domains, however, the terminal symbol may in fact improve comparability due to the potential increase of determinism.

C.3 Experiments

The experiments presented in this paper are aimed towards evaluating whether the *string order*, *maximum string length* and *error tolerance* when testing RNN predictors affect the EGA significantly for given trained populations of networks. We therefore consider the training of the networks a secondary matter, i.e. no effort has been spent on finding optimal parameters for the EH. In effect, the results may not be comparable to other papers (a comparison that should not be done anyway). Instead the training should just be seen as a necessary step to generate populations of networks in which some effects of the testing parameters can be demonstrated.

C.3.1 The Testing Procedure

The test set is determined by the string order and maximum string length. Three orderings of string are used; random, ascending, and descending. We let the maximum string length of the test set vary between 10 and 100. In each test, exactly 1000 strings of each length are included. The strings are concatenated into the sequence which the network is trained to predict.

The performance of the network is recorded for the 1000 strings of each length it receives. If we consider just one network we will have an estimate of the performance of the network on each individual string length. This performance is typically higher for short strings and lower for long strings. The performance is, however, not necessarily decreasing monotonically and a string with a high n may be predicted completely accurately, while the strings of length $n-1$ could at the same time be completely inaccurately predicted. We have chosen to record the maximum string length that the network processes correctly (string length is something which all previous papers have mentioned when talking about the generalization ability of their networks), but this measurement needs to take into account the nonmonotonic performance degradations for longer strings. The following definition will lead to such a measurement.

The *correctness*, $c(n)$, of a network in terms of predicting a given length is defined as

$$c(n) = \frac{\text{no. of correctly predicted strings of length } n}{\text{no. of strings of length } n} \tag{C.1}$$

where the total number of strings of length n in this case was 1000 for all n up

Reference	Domain	Training set	Test set order	Test set
Wiles & Elman (1995)	$\mathbf{a}^n\mathbf{b}^n$	$1 \leq n \leq 12$	*	$1 \leq n \leq 30$
Tonkes et al. (1998)	$\mathbf{a}^n\mathbf{b}^n$	$1 \leq n \leq 10$	*	$1 \leq n \leq 12$
Rodriguez et al. (1999)	$\mathbf{a}^n\mathbf{b}^n$	$1 \leq n \leq 11$	asc	until failure
Tonkes & Wiles (1999)	$\mathbf{a}^n\mathbf{b}^n$	$1 \leq n \leq 10$	*	$1 \leq n \leq 12$
Bodén et al. (1999)	$\mathbf{a}^n\mathbf{b}^n$	$1 \leq n \leq 10$	*	$1 \leq n \leq 12$
Bodén et al. (2000)	$\mathbf{a}^n\mathbf{b}^n$	$1 \leq n \leq 10$	rand	*
Chalup & Blair (2000)	$\mathbf{a}^n\mathbf{b}^n$	$1 \leq n \leq 20$	rand	$1 \leq n \leq 20$**
—”—	$\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$	$1 \leq n \leq 20$	rand	$1 \leq n \leq 20$**
Bodén & Wiles (2000)	$\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$	$1 \leq n \leq 10$	desc	$1 \leq n \leq$ “large n”
Gers & Schmidhuber (2001)	$\mathbf{a}^n\mathbf{b}^n\mathbf{T}$	$1 \leq n \leq 10$ to $1 \leq n \leq 50$	*	$1 \leq n \leq 1000$
—”—	$\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n\mathbf{T}$	$1 \leq n \leq 10$ to $1 \leq n \leq 50$	*	$1 \leq n \leq 500$
Bodén & Blair (2002)	$\mathbf{a}^n\mathbf{b}^n$	$1 \leq n \leq 10$	*	*
Schmidhuber et al. (2002)	Refers to the data in Gers & Schmidhuber (2001)			

*=not explicitly defined.

**=incrementally tested during training.

Table C.1: A summary of CFL and CSL prediction experiments using various neural network architectures.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$c(n)$	1.00	1.00	1.00	1.00	1.00	1.00	0.91	1.00	1.00	0.77	0.10	0.00	1.00	0.00
$c_r(n)$	1.00	1.00	1.00	1.00	1.00	1.00	0.91	0.91	0.91	0.70	0.07	0.00	0.00	0.00

Table C.2: A realistic example of an evaluation of an RNN by using $c(n)$ and $c_r(n)$ of equation C.1 and C.2. If requiring a strong network, this network's EGA is up to string length 6 and if only requiring a weak network, the EGA is 11.

to the maximal string length. A correctly predicted string means that at least the predictable part (i.e. not the first `b`) of the string is correctly predicted. As $c(n)$ is not monotonically decreasing it can not be used directly to unambiguously define up to which string length the network is successful. In Table C.2 an example of a string evaluation is shown. From this example it is clear that there is no obvious way to give statements of which maximum string length the network can handle. In the example, the network can handle all strings up to $\mathbf{a}^6\mathbf{b}^6$ but fails on some of $\mathbf{a}^7\mathbf{b}^7$, $\mathbf{a}^{10}\mathbf{b}^{10}$ and $\mathbf{a}^{11}\mathbf{b}^{11}$. It can also handle all of $\mathbf{a}^{13}\mathbf{b}^{13}$, but none of $\mathbf{a}^{12}\mathbf{b}^{12}$ or $\mathbf{a}^{14}\mathbf{b}^{14}$. Up to what string length should we then say that the network is performing correctly?

To solve this we introduce a recursive definition of correctness, reflecting that the performance on one string length depends also on the performance on all shorter string lengths. The *recursive correctness*, $c_r(n)$, is defined as:

$$\begin{aligned} c_r(1) &= c(1) \\ c_r(n) &= c_r(n-1) \cdot c(n) \text{ for } n > 1 \end{aligned} \tag{C.2}$$

In the example of Table C.2, $c_r(n)$ is monotonically decreasing and only accepts string lengths for which previous string lengths also have been correctly predicted. The correctly predicted $\mathbf{a}^{13}\mathbf{b}^{13}$ are now ignored since no correct predictions of $\mathbf{a}^{12}\mathbf{b}^{12}$ were made.

The *error tolerance* is the quality demand on the network by the experimenter. The highest error tolerance corresponds to the experimenter being satisfied with the RNN correctly predicting strings only *at least once* and the lowest error tolerance is when the RNN needs to correctly predict *all strings*. Chalup and Blair (Chalup & Blair, 2000) addressed the issue of error tolerance explicitly and defined "weak solutions" and "strong solutions" to correspond to networks satisfying the highest and lowest error tolerance requirements respectively. We adopt these terms in this paper. The EGA (using $c_r(n)$) of the network in the example in Table C.2 is then 6 if we consider only strong solutions, and 11 if we only require weak solutions.

C.3.2 Architecture & Training Algorithm

The network architecture used in our experiments is an SRN and the optimisation algorithm is an EH, see (Bodén et al., 2000) for details. The fitness is proportional to the number of correctly predicted strings in a concatenated sequence of strings from $\mathbf{a}^n\mathbf{b}^n$ with $1 \leq n \leq 10$ where each string length occurred exactly three times (cf. the testing procedure in the previous section). Three separate fitness functions are used; F_{rand}, F_{asc} and F_{desc} for random, ascending and descending string length order respectively, i.e. the only difference between the fitness functions is the ordering of the strings. It should be noted that the aim of the experiment is *not* to evaluate the differences between these populations but to evaluate how the EGA varies for these fixed populations under different testing strategies. The use of three different populations may reveal different effects the testing procedure may have on the estimated results. In fact *any* sufficiently large population would do as the goal basically is to show that there are populations for which testing procedure differences significantly affect the estimated performance.

The evolutionary algorithm was run for 10,000 generations with a mutation rate of $\sigma = 1.0$ and a population size of 100 of which 20 were selected as elite. The elite group was saved to the next generation and was the group from which new networks were generated. 120 runs were carried out for each fitness function with different random seeds and the best network of each successful end-population was saved for further analysis. A population was deemed successful if at least one of its networks correctly predicted (the predictable part of) *all* strings in the training set.

C.4 Results

C.4.1 Training Results

Of the 120 experiments with each of the three fitness functions F_{rand}, F_{asc} and F_{desc} the number of successful (in terms of correctly predicting the entire training set) runs were 114, 75 and 76 respectively. All the statistics will be based on the best network of each successful population. It is worth noting that the success rate is much higher for F_{rand} than for F_{asc} and F_{desc}. This is probably due to higher sensitivity to local optima for the deterministic fitness functions. Subsequent experiments (not documented here) indicated that for higher values of the mutation parameter, σ, this problem vanishes.

C.4.2 Estimated Generalization Abilities

The resulting EGA of networks generated with the three fitness functions tested under different conditions are shown in Table C.3. The maximum correctly predicted string length of each successful network was calculated according to equations C.1 and C.2 as in the example in Table C.2.

The Effect of Error Tolerance Level

The effect of demanding weak or strong networks is clearest when the networks are tested on strings in a random order. The EGA is half or lower for the strong solutions given a high enough maximum string length of the test set. The error tolerance effect is still there with a test set in ascending order, but weaker.

Interestingly, the error tolerance has virtually no effect at all when testing on strings in a descending order. We speculate that this is due to the RNN gradually receiving simpler and simpler strings, resulting in the exact same behaviour every time, i.e. the network either correctly predicts all strings of a specific length or none at all.

One should keep in mind that, as the test set has 1000 replicas of each string length, strong solutions correctly predict 1000 out of 1000 strings, whereas weak solutions need only predict 1 out of 1000 correctly. In our opinion, this makes strong solutions much more interesting.

Effects of Maximum String Length

The effect of the maximum string length (N in Table C.3) differs depending on test set order, error tolerance and fitness function. When only considering strong solutions and random test set order, a higher N leads to a significantly lower EGA for all networks. The opposite seems to be true for most weak solutions for all test set orderings and networks. For ascending test set order, the degrading performance for higher values of N is not as clear as when testing on randomly ordered strings. For tests on strings in descending order, N has no degrading effect.

The Effects of String Order

String order is perhaps one of the more interesting aspects of the testing procedures, as there were three distinct orderings found in previous work while most papers did not describe this aspect of testing explicitly. In our experiments, string order played two roles, in the training and testing of networks. The networks trained on the different training sets can be clearly ranked in terms of performance. Networks trained on F_{rand} are clearly better than F_{asc} which is clearly better than F_{desc}.

A ranking of the test sets is not as straightforward. Considering only strong solutions it is, however, clear that a randomly ordered test set is tougher than the ascending order which is in turn tougher than the descending order. For weak solutions the randomly ordered test set gives the highest results. This is not surprising as weak solutions need only 1 out of 1000 strings correctly predicted of every string length and a randomly ordered set gives the network a higher variety of initial states of which some may lead to a correct prediction.

It is interesting to see that, as a validation of the network training, all networks handle their training sets perfectly and that the networks trained with F_{rand} also handle the other training sets perfectly. Networks trained on randomly ordered strings thus seem more robust.

Although the results of the randomly ordered test set seem to be most sensitive to the other parameters (i.e. string length and error tolerance), in our opinion, this test provides the most interesting results, as the network will be tested more rigorously.

C.5 Discussion and Conclusions

It is clear from table C.3 that changes in the testing procedure render significantly different results. These effects are also not consistent for the three populations and can therefore at this stage not be predicted. These results are not surprising, as it is well known that initial conditions may affect the behavior of dynamical systems, and hence affect the performance of RNNs, a subset of dynamical systems. The cited papers, implicitly or explicitly, touch the dynamical nature of RNNs, but in the construction or description of the experimental setup this important issue often does not receive much attention. All papers describe the architectures and algorithmic details of the learning techniques quite thoroughly and present insightful, detailed analyses of individual networks. But without a proper description of the testing procedures used to generate quantitative results, reproducibility and comparability are lost. Three papers also make cross paper comparisons (Gers & Schmidhuber, 2001; Bodén & Wiles, 2002; Schmidhuber et al., 2002) in the domain of these papers, comparisons that, due to the problems pointed out here, may be questioned. For the same reasons, it would also not make sense to compare our results to those of any other paper using different testing procedures.

Some practical recommendations for future research in this area: Train and test sets should be ordered randomly to give both robust networks and a thorough testing of these networks. Only strong networks (or perhaps a slightly relaxed version of "strong", e.g. 90-99% correct) should be considered. A network solving a task only (at least) once is far less interesting than those solving it consistently. Since the results also indicate that the maximum string length in the test set has a significant effect on the results the *expected* performance may affect the *measured* performance directly, since the maximum string length in the test set will probably be chosen based on the expected performance. Hence, the maximum string length in the test set should be varied, perhaps starting with a low value and then increasing stepwise.

What can be learned from this is that to guarantee reproducability, the description of the generation of testable objects has to be complemented with a description of the testing procedure applied to these objects. In the cited papers the architectures, training procedures and analysis of individual RNNs came out mostly crystal clear to the reader, while some crucial details of the testing methods did less so. So our final, and most important recommendation, is to recognize that the analysis tools are as important a part of the data generation as the networks themselves.

	Networks trained on F_{rand} (114 RNNs)				Networks trained on F_{asc} (75 RNNs)				Networks trained on F_{desc} (76 RNNs)			
	strong		weak		strong		weak		strong		weak	
	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
N	**Test set in random order**											
10	10.00 (0.00)	10	10.00 (0.00)	10	6.79 (0.45)	10	10.00 (0.00)	10	4.07 (0.48)	10	10.00 (0.00)	10
15	8.95 (0.45)	15	12.09 (0.16)	15	5.11 (0.62)	15	12.13 (0.26)	15	3.46 (0.56)	15	12.36 (0.21)	15
20	7.77 (0.53)	20	12.48 (0.22)	20	3.33 (0.58)	17	13.12 (0.31)	20	1.58 (0.37)	14	12.96 (0.36)	20
25	6.51 (0.50)	20	12.55 (0.23)	23	2.72 (0.54)	17	13.53 (0.36)	25	1.37 (0.35)	14	13.55 (0.38)	25
50	5.81 (0.48)	20	12.63 (0.24)	23	2.00 (0.45)	17	14.19 (0.61)	49	1.32 (0.35)	14	14.53 (0.59)	36
100	5.81 (0.48)	20	12.62 (0.24)	23	2.00 (0.45)	17	14.08 (0.60)	49	1.33 (0.35)	14	14.04 (0.49)	30
	Test set in ascending order											
10	10.00 (0.00)	10	10.00 (0.00)	10	10.00 (0.00)	10	10.00 (0.00)	10	6.49 (0.49)	10	8.80 (0.31)	10
15	10.84 (0.34)	15	11.71 (0.16)	15	9.24 (0.61)	15	11.43 (0.28)	15	6.49 (0.64)	15	8.78 (0.52)	15
20	11.10 (0.37)	20	11.97 (0.22)	20	8.88 (0.66)	20	11.65 (0.33)	20	5.16 (0.63)	20	8.75 (0.56)	20
25	10.26 (0.44)	21	11.98 (0.22)	21	7.32 (0.75)	25	11.71 (0.36)	25	5.26 (0.67)	21	8.59 (0.58)	21
50	10.80 (0.38)	20	11.98 (0.22)	21	8.07 (0.67)	19	12.00 (0.54)	45	5.00 (0.64)	21	8.51 (0.59)	21
100	10.80 (0.38)	20	11.98 (0.22)	21	8.07 (0.67)	19	12.00 (0.54)	45	5.00 (0.64)	21	8.51 (0.59)	21
	Test set in descending order											
10	10.00 (0.00)	10	10.00 (0.00)	10	9.33 (0.22)	10	9.36 (0.22)	10	10.00 (0.00)	10	10.00 (0.00)	10
15	11.64 (0.16)	15	11.64 (0.16)	15	10.97 (0.36)	15	10.97 (0.36)	15	10.50 (0.24)	15	10.50 (0.24)	15
20	11.89 (0.22)	20	11.89 (0.22)	20	11.08 (0.45)	20	11.08 (0.45)	20	10.51 (0.32)	20	10.51 (0.32)	20
25	11.90 (0.22)	21	11.90 (0.22)	21	11.16 (0.46)	25	11.16 (0.46)	25	10.63 (0.33)	23	10.63 (0.33)	23
50	11.90 (0.22)	21	11.90 (0.22)	21	11.43 (0.62)	45	11.43 (0.62)	45	10.61 (0.32)	21	10.61 (0.32)	21
100	11.90 (0.22)	21	11.90 (0.22)	21	11.43 (0.62)	45	11.43 (0.62)	45	10.61 (0.32)	21	10.61 (0.32)	21

Table C.3: The average, standard deviation (in parentheses), and maximum length the networks was deemed to process correctly. The performance is evaluated on networks generated with the three different fitness functions, F_{rand}, F_{asc} and F_{desc}. The results are separated into the three different test sets and results for weak and strong solutions are presented separately. The results for different maximum string lengths N are also shown separately.

Appendix D

Jacobsson & Ziemke (2003b)

Reducing Complexity of Rule Extraction from Prediction RNNs through Domain Interaction[1]

Henrik Jacobsson, Tom Ziemke

Abstract
This paper presents a quantitative investigation of the differences between rule extraction through breadth first search and through sampling the states of the RNN in interaction with its domain. We show that for an RNN trained to predict symbol sequences in formal grammar domains, the breadth first search is especially inefficient for languages sharing properties with realistic real world domains. We also identify some important research issues, needed to be resolved to ensure further development in the field of rule extraction from RNNs.

D.1 Introduction

An RNN can be painstakingly difficult to analyze. Very often RNN analysis becomes a matter of creating small enough networks to allow a direct visualization of the internal activations. There are almost as many approaches to RNN analysis as there are papers about RNN and the methods are often *ad hoc* and adapted to specific domains and network architectures. Rule extraction (RE) from RNNs (Giles, Miller, Chen, Chen & Sun, 1992; Zeng et al., 1993; Tiňo & Šajda, 1995; Blair & Pollack, 1997; Tiňo & Köteles, 1999) offers a very promising tool for analyzing RNNs as it generates a functional model (usually a finite state automaton, FSA) of the of the RNN, providing an abstract symbolic model of the potentially complex analog network dynamics. In comparison to other analysis tools, such as cluster analysis, vector flow fields, analysis of fixed points etc., RE gives insight not only to the "passive" clusters resulting in the state space, but also to the "active" role of these clusters in the RNN interaction with the domain. RE is also not inherently limited by the dimensionality of the state space as are visualization methods. However, RE suffers from an apparent increasing space and time complexity for larger

[1]This is a verbatim copy of Jacobsson and Ziemke (2003b). Only the formatting and contact information differs from the original (the bibliography is also not included here since it can be found elsewhere in the thesis).

and more complex networks and therefore various heuristics need to be developed to allow RE to tackle more 'difficult' RNNs.

The effect of one such heuristic will be investigated quantitatively in this paper. The complexity of the behavior of an RNN is a product of its internal functional mappings generating sequences of states and output and of the complexity of the domain from which the network is fed input patterns. It is well known that even relatively simple systems can exhibit surprisingly complex behavior in interaction with a complex environment but the opposite is true also: the complexity of the behavior of a potentially complex system can be restricted by a simple environment. We will in this paper show an example of how this can be exploited as a heuristics for RE from RNNs by using the domain as a means for generating the states of the network that are the basis for the extracted rule set as opposed to performing a breadth first search based on the possible input patterns. Both methods have been used previously in RE algorithms, but to our knowledge no comparative study has been presented.

We will first introduce our definition of RNNs, rule extraction and some theoretical prerequisits. Then the experiments and results are presented. In the last section we draw some conclusions and discuss possible future directions.

D.2 Background

In this paper we will, for simplicity, stick to a very simple definition of recurrent neural networks. The activations of the input, state and output nodes are for example restricted to values in the interval $[0, 1]$ and the output is functionally dependent on the state alone (excluding for example some forms of second ordered networks).

Definition D.1 A *Recurrent Neural Network* is a 6-tuple $R = \langle I, O, S, \delta, \gamma, \mathbf{s}_0 \rangle$ where
$I \subseteq [0, 1]^{n_I}$ is the *input space*,
$S \subseteq [0, 1]^{n_S}$ is the *state space*,
$O \subseteq [0, 1]^{n_O}$ is the *output space*,
$\delta : S \times I \to S$ is the *state transition function* and
$\gamma : S \to O$ is the *state interpretation function*
$\mathbf{s}_0 \in S$ is the initial state vector $\Box$

Where n_I, n_S and n_O are the dimensionality of each respective space. Note that the weights of the connections and activation functions of the individual nodes are subsumed by δ and γ in this definition. Those details are simply ignored by existing RE algorithms and the neurons of the network are treated as ensembles rather than as individuals. The term *compositional* was suggested by (Tickle et al., 1998) to denote this level of granularity of the rule extraction algorithm's view upon the underlying network. The other modes of granularity are decompositional (white box), pedagogical (black box) and eclectic (containing elements of both decompositional and pedagogical).

States in the state space S will be visited when the network is fed input vectors from the input space. However, the full set of possible input patterns is seldomly needed to take into account for training or analysis of the RNN, e.g. if different input features are strongly correlated. Instead we can define the set $\hat{I} \subset I$ as a *finite* set of patterns that the network actually will receive *in situ*, i.e. when receiving input from the domain. We have here chosen to define $\hat{I}$ as finite since in previous approaches to RE from RNNs, formal language tasks have almost exclusively been considered. For this reason we introduce a set of symbols, Σ, isomorphic to $\hat{I}$, i.e. for every symbol in Σ, there is exactly one corresponding member in $\hat{I}$. In many papers where a formal language recognition/prediction task is studied, the symbols of Σ are encoded in I through 'one hot' encoding, i.e. every symbol of Σ 'activates' only one corresponding element of the input vector.

When the network is fed patterns from $\hat{I}$ a number of states will be visited. This set can formally be defined as the set of $\hat{I}$*-accessible states* from the initial state $\mathbf{s_0}$, let us call it $\mathcal{A}_0^{\hat{I}} \subseteq S$. $\mathcal{A}_0^{\hat{I}}$ is composed of those states in S that will be visited through the iterative mappings induced by *all possible input patterns* in $\hat{I}$ in *all possible orders* as defined in equations D.1 and D.2. In other words, $\mathcal{A}_0^{\hat{I}}$ is the set of states that would be visited if all possible sequences over Σ (denoted Σ^*) were fed to the network (with the network reset to s_0 before each new string).

$$\mathcal{Y}_0 = \{\mathbf{s}_0\}, \quad \mathcal{Y}_{n+1} = \mathcal{Y}_n \cup \bigcup_{\mathbf{i} \in \hat{I}} \delta(\mathbf{i}, \mathcal{Y}_n) \text{ for } n \geq 0 \tag{D.1}$$

$$\mathcal{A}_0^{\hat{I}} = \lim_{n \to \infty} \mathcal{Y}_n \tag{D.2}$$

A similar definition (for binary languages only) of accessible states is found in (Blair & Pollack, 1997). The production of these states is equivalent to that of an iterated function system (IFS) (Kolen, 1994b).

In rule extraction algorithms the state space needs to be quantized to a finite set of classes. This quantization function is here denoted $Q : S \to \{0, 1, 2, \ldots, N\}$ in its general form. In previous RE approaches Q is typically a simple orthogonal lattice dividing the state space into hypercubes (e.g. (Giles, Miller, Chen, Chen & Sun, 1992)), dividing the activation range of each individual state dimension into q intervals of equal size. This results in q^{ns} hypercubes that can be uniquely enumerated. In this paper we refer to these hypercubes as *bins* and the degree of quantization in each dimension of the state space will be referred to as q. Other clustering methods used for RE from RNNs are for example k-means clustering (e.g. (Zeng et al., 1993)) or a self organizing map SOM (e.g. (Tiňo & Šajda, 1995)).

D.2.1 Rule extraction through breadth first search

One of the most common algorithm for rule extraction from RNNs is that of Giles et al. (Giles, Miller, Chen, Chen & Sun, 1992). The algorithm conducts a breadth first search in the state space to extract a finite state machine from the RNN. The RNNs were prior to RE trained to classify strings as grammatical or non-grammatical. In the general case, any string in Σ^* should then be possible for the network to process.

The algorithm starts with an initial state s_o and generates the outgoing transitions from this state by computing all new states for all input symbols, i.e. $\delta(\mathbf{s}_0, \mathbf{i})$ for all $\mathbf{i} \in \hat{I}$. This is then repeated for all first states in each visited bin until all these states have been tested in this way and no new bin is visited. The number of the bin and the corresponding output of the first encountered state vector of each bin is then transformed to the extracted FSA. This FSA is then minimized using a standard minimization algorithm (Hopcroft & Ullman, 1979). The RE algorithm starts with a small q and is repeated with increased values of q until the machine is consistent with the training data.

One way to view this algorithm is to see that the search generates a tree of symbols that generates a set of states in the network. From the root node (equivalent to the initial state of the network) all symbols expand to subtrees that are expanded likewise until all leaf-nodes lead to loops in $\hat{S}$. From the root node the path to each leaf node is the equivalent to a string of symbols. If all these substrings are fed to the network with a network reset between each string, the exact same states as visited during breadth first RE will be visited. This set of substrings will be called X_B, where $X_B \subset \Sigma^*$ and the states visited during the extraction of rules will be called $\mathcal{A}_0^{X_B}$, i.e. the set accessible from the initial state $\mathbf{s_0}$ through breadth first search RE, $\mathcal{A}_0^{X_B} \subseteq \mathcal{A}_0^{\hat{I}}$.

D.2.2 Rule extraction in a domain context

As mentioned above, in many tasks the full set of strings in Σ^* is not relevant for the training of the network. Much research on RNN is focused on *prediction tasks* which in many ways are much less restrictive than classification tasks since the role of an external "teacher" is reduced to a minimum. For prediction tasks the network is not required to correctly predict *all* possible sequences of symbols, but only the ones that belongs to the domain. The network does typically not even need to correctly predict all symbols of the sequences in the domain, as some subparts of the sequence may be inherently unpredictable. The temporal XOR problem is one such example where only every third symbol is at all predictable (Elman, 1990). This means that the rules extracted from the network need only incorporate the sequences and subsequences that the network will encounter in the domain. If the network is for example trained to predict events that results from the behavior of a autonomous robot it would not be reasonable to extract rules for actions that would never be carried out in certain situations, e.g. the event 'drive-forward' should not occur if the robot is in the state 'wall-ahead' and is successfully avoiding obstacles.

We will use the notation $X \in \Sigma^*$ to refer to a sequence generated or sampled from the domain. The sequence is written as $x_0x_1x_2\dots x_n$. This domain specific input sequence will generate a sequence of states in the network which we will refer to as the X*-accessible set* from $\mathbf{s_0}$, or $\mathcal{A}_0^X$. $\mathcal{A}_0^X \subset \mathcal{A}_0^{\hat{I}}$ is more formally defined as

$$\mathbf{s}_{n+1} = \delta(\mathbf{s}_n, \mathbf{i}_n) \tag{D.3}$$

where $n \geq 0$ and $\mathbf{i}_n$ corresponds to x_n (remember that $\hat{I}$ and Σ are isomorphic and $X \in \Sigma^*$). And

$$\mathcal{A}_0^X = \{s_0, s_1, s_2, \dots, s_n\} \tag{D.4}$$

where n is the length of sequence X.

From the information about states gathered through the processing of the domain, a state machine of some kind, emulating the network, can be generated. The typically indeterministic data from the network must be processed in some way to lead to a deterministic discrete machine (e.g. (Tiňo & Šajda, 1995)) or the extracted state machine can in itself be stochastic (e.g. (Tiňo & Köteles, 1999)).

D.3 Experiments

The sets $\hat{\mathcal{A}}_0^X$ and $\hat{\mathcal{A}}_0^{X_B}$ are both subsets of $\mathcal{A}_0^{\hat{I}}$ but cover different aspect and generate different rule sets. In this paper we will experimentally investigate the relation between $\hat{\mathcal{A}}_0^X$ and $\hat{\mathcal{A}}_0^{X_B}$, i.e. the difference between the domain sampling and breadth first search approaches of RE in terms of the visited states.

In these experiments we have chosen to limit the tasks to be pure prediction tasks, i.e. the task for a network is to predict the next symbol in a sequence generated by a grammar and *not* to classify incoming strings. Another prerequisite is also that the networks are perfect, i.e. they never predict predictable symbols of the domain incorrectly. This in order to prevent illegal rules to be caused by an erroneous network, but instead to be indicators of flaws of the extraction procedure itself.

D.3.1 The Networks

Three prediction domains have been considered in this paper, two regular grammars and one context free. (Casey, 1996) showed that from an RNN effectively implementing a regular grammar, a finite state machine consistent with the RNN can be extracted. For the context free grammar, we assume that some limited version of it can be extracted.

- The simplest is the temporal XOR-problem, suggested in (Elman, 1990), where every third binary symbol is determined by an XOR operation of the two preceding symbols which are random.
- The next grammar, the "6-letter grammar", was created by Elman (Elman, 1990) to test a language with more than two symbols and that required some deeper memory in the network. The sequence from the grammar consists of the subsequences **ba**, **dii** and **guuu** concatenated in random order[2]. Consequently, only the vowels are predictable.
- The third domain was $\mathbf{0}^n\mathbf{1}^n$, a context free language. n was in these experiments $1 \leq n \leq 10$ and varied in random order with the generated strings concatenated into a single sequence. In this language, only the **1**'s and the first **0** is predictable. The full grammar, with n unlimited, cannot be represented in any finite state machine, but since we only require the rules to correctly predict the training set it is possible to view this as a regular grammar (although this may be complicated if the network has actually learned to generalize to longer sequences).

These domains were chosen to test the effect of the number of symbols and language class separately. All languages have predictable and unpredictable parts of the generated sequences and the networks are all trained to predict the next symbol. 100 networks were trained on each domain until they were deemed to perfectly predict the predictable parts of the sequences. The architecture chosen was simple recurrent networks (SRNs) (Elman, 1990) with two hidden nodes. For the regular language backpropagation through time (BPTT) was used to train

[2]In our experiments we used 'one-hot' encoding to represent the symbols to the network, i.e. six bits were used of which each one encodes only one symbol. Elman used a quite different non-orthogonal encoding based on phonological properties of the letters.

them and since BPTT had problems on the context free language an evolutionary hill-climbing algorithm was used for that instead.

D.3.2 Evaluation criteria

The primary objective of the experiments was to assert the degree of excess computational power used by rule extraction through breadth first in the selected domains. For all networks, we tested RE through breadth first search and sampling for varying values of q to see the effects of the quantization level on various aspects.

We chose to measure $|\hat{\mathcal{A}}_0^{X_B}|$ and $|\hat{\mathcal{A}}_0^{X}|$ and will here present the ratio, $|\hat{\mathcal{A}}_0^{X_B}|/|\hat{\mathcal{A}}_0^{X}|$, i.e. the relative difference in number of bins visited through RE and through processing of domain respectively. Also, the proportion of substrings in X_B that are at all possible in the domain which the network is trained on is measured. If the breadth first RE for example tests the sequence **00011110** on a correctly predicting $\mathbf{0}^n\mathbf{1}^n$-network starting from the initial state in the network, it is a symbol-sequence that never occurs in the true domain and should therefore be considered an obsolete sequence.

The performance of the extracted machines was also monitored to determine whether correct rules were extracted. The termination point for the breadth first RE, i.e. when the extracted machine is consistent with the data, was also tested in order to see if and when the algorithm would terminate.

D.4 Results

In Figure D.1 we show an example of how RE can be illustrated in the state space of the RNN predicting the 6-letter sequence. In this example it can be seen how RE through breadth first search finds many states irrelevant for predicting within the domain.

In Figure D.2 the ratios of visited bins and of syntactical substrings generated in the RNN by breadth first search RE in comparison to domain interaction are shown. It is clear that breadth first RE generated the biggest amount of irrelevant tests on the 6-letter networks. This is probably due to the fact that after each symbol in the 6-letter sequence, typically only one of six symbols will occur in the domain whereas all six symbols will be tested by the RE.

It should be mentioned that the RE algorithm terminated quite rapidly; for XOR within $q = 3$ to $q = 10$, for the 6-letter grammar within $q = 2$ to $q = 8$. But for $\mathbf{0}^n\mathbf{1}^n$ at least $q = 21$ and for 15% of the networks, the algorithm did not terminate at all. 84% of the XOR networks seemed to stabilize in terms of extracting equivalent machines. Only 2% of the 6-letter sequence stabilized. 5% of the $\mathbf{0}^n\mathbf{1}^n$ actually also stabilized. These numbers are not fully certain however, since the number of states in the minimized automata could continue increasing for higher quantization levels.

D.5 Discussion and Conclusions

We have shown that the degree to which breadth first RE requires excessive computational resources seems to be related to the number of symbols in the language

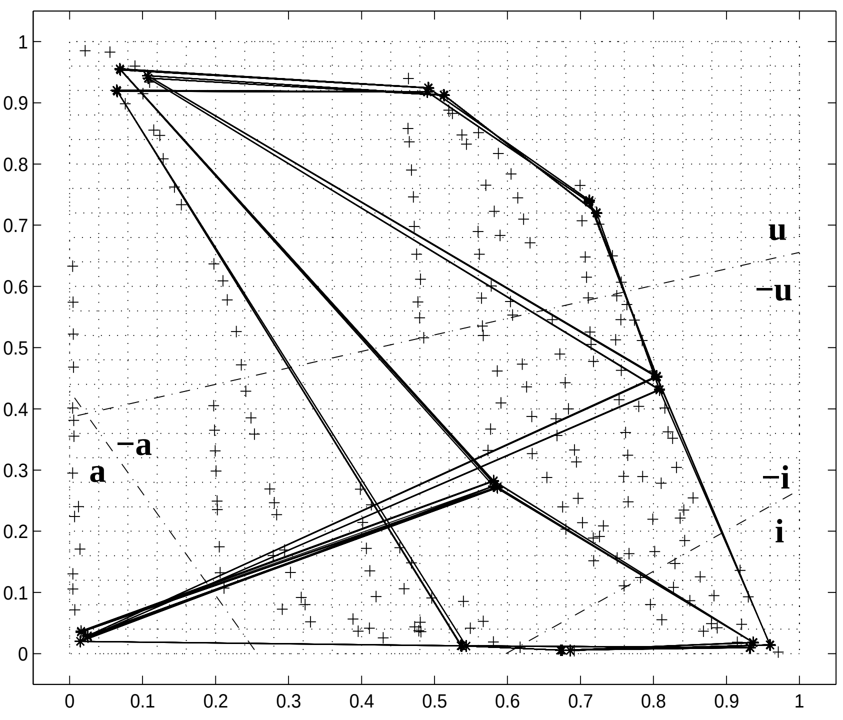
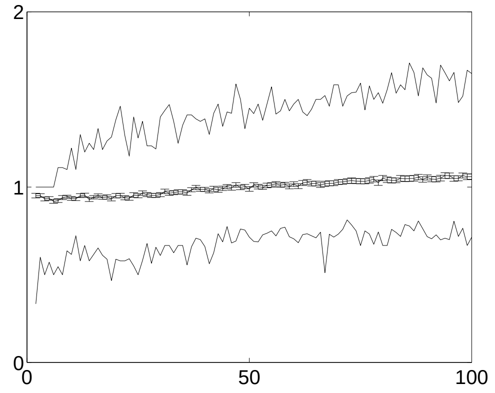


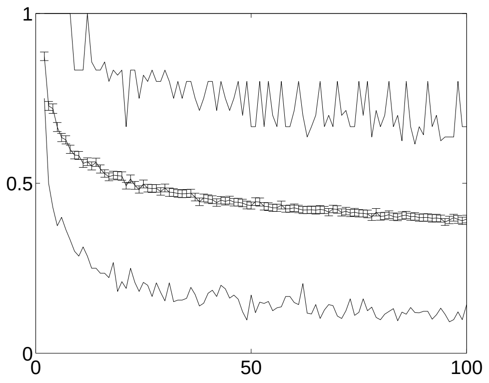
Figure D.1: The internal activation of a network performing prediction in the 6-letter sequence. The lattice corresponds to the discretization with the state divided into 25^2 bins in this example. The diagonal dotted lines are the hyperplanes, defining the borders within the state space for which symbol that is predicted. The hyperplanes divides the state space into the '**u**'-region on the upper half, the '**a**'-region on the lower left side and the '**i**'-region on the lower right side. The rest of the state space corresponds to no valid symbol; the center area with all output nodes set to zero and a small area on the center left side with the '**a**' and '**u**'-node active simultaneously. The states visited through the breadth first RE are denoted '+' and the states visited through processing of the domain are denoted '$*$' and are connected to show the order of the states visited.

for networks trained to predict symbolic sequences. The ratio of, for the domain, relevant "questions" (in form of sequences) "asked" to the network also was very low for the grammar with six symbols, and for the context free grammar.

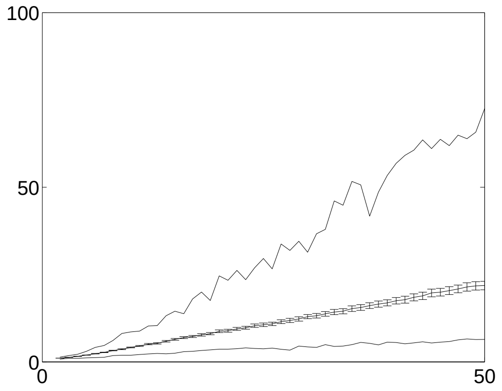
Blair and Pollack (Blair & Pollack, 1997) suggested to use the state count of the extracted machine to determine whether the network is effectively implementing "regular" or "non-regular" automaton. If the state count is growing indefinitely with q, they proposed to use this as an indicator that the underlying RNN is non-regular. But the results presented here suggest that, for prediction tasks, regularity of the network can not be tested as suggested in (Blair & Pollack, 1997) since the number of states generated from networks predicting sequences of the regular languages was almost always growing indefinitely although the networks were predicting all symbols of the language perfectly. The percentage of networks for which the RE stabilized did also not correlate with the language class. The termination criterion of the RE algorithm was however satisfied much earlier for regular than for context free prediction networks. But this could also be due to the larger number of states needed to model the strictly *regular language* $\mathbf{0}^n\mathbf{1}^n$ with $1 \leq n \leq 10$. This should however be investigated further to give more insight into



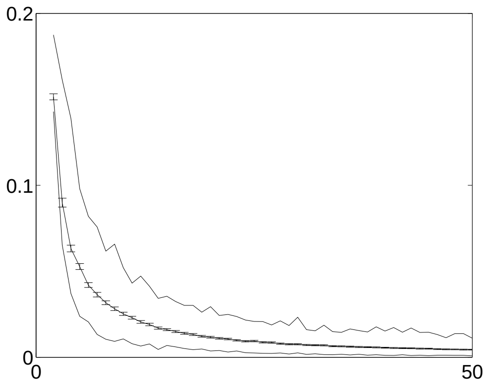
(a)



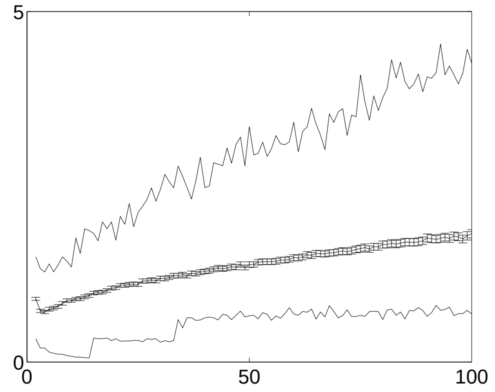
(b)



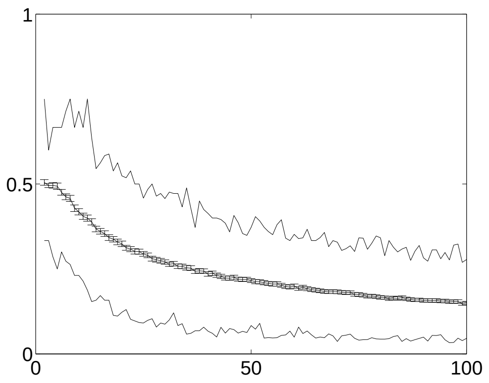
(c)



(d)



(e)



(f)

Figure D.2: The ratio $|\hat{\mathcal{A}}_0^{X_B}|/|\hat{\mathcal{A}}_0^{X}|$ is shown in the left column and the ratio of substrings in X_B possible in the domain is shown on the left side. (a) and (b) correspond to the XOR-language, (c) and (d) to the 6-letter language (observe that for this language q was at most 50) and (e) and (f) to the $\mathbf{0}^n\mathbf{1}^n$-language. The maximum, minimum, average and standard deviation of one hundred networks for each domain are shown.

if (and how) RE can be used to determine the underlying language class, which is judged to be "fool's gold" by Kolen (Kolen, 1993).

One can also argue that RE through search is, in some sense, less credible than through sampling since it requires the possibility of an external entity *setting* the state of the network. Sampling of the networks internal states generated in the context of its domain however generates stochastic machines that are harder to analyse (and to minimize, execute, compare etc.) than the finite automata generated by breadth first search RE.

We suggest that in most "real world domains", e.g. stock market prediction, the task is precisely to *predict* sequences of data with typically a magnitude of possible input patterns. According to our results, in these types of tasks it would be especially beneficial to use sampling rather than breadth first to extract rules.

But, to fully exploit the potential of RE through sampling and to ensure further development of these algorithms, new questions need to be asked. For example, the optimal quantization function for the state space should be sought. And to do that, we need to ask how to evaluate different quantization functions. Since an RNN (as defined here) is deterministic, one possibility could be to give higher scores to quantization functions generating "less stochastic" machines. Another difficulty that has not been investigated properly is how rule extraction from imperfect networks (very commonly found in real world domains) should be conducted. In a way, this has been implicitly touched in this paper, since we used partly unpredictable prediction domains, but this should be investigated in further detail.