Exploiting Holistic Computation: An evaluation of the Sequential RAAM

by

JAMES ALISTAIR HAMMERTON

A thesis submitted to the Faculty of Science
of The University of Birmingham
for the Degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
Faculty of Science
University of Birmingham
September 1998

Abstract

In recent years it has been claimed that connectionist methods of representing compositional structures, such as lists and trees, support a new form of symbol processing known as *holistic computation*. In a holistic computation the constituents of an object are acted upon simultaneously, rather than on a one-by-one basis as is typical in traditional symbolic systems. This thesis presents firstly, a critical examination of the concept of holistic computation, as described in the literature, along with a revised definition of the concept that aims to clarify the issues involved. In particular it is argued that holistic representations are not necessary for holistic computation and that holistic computation is not restricted to connectionist systems. Secondly, an evaluation of the capacity of a particular connectionist representation, the Sequential RAAM, to generate representations that support holistic symbol processing is presented. It is concluded that the Sequential RAAM is not as effective a vehicle for holistic symbol processing as it initially appeared, but that there may be some scope for improving its performance.

Acknowledgements

I would like to thank the following people:

- My supervisor, Peter Hancox, for his support, advice and guidance throughout my PhD. The other members of my thesis group, Russell Beale, Riccardo Poli and Ela Claridge for the advice, questions and suggestions they offered during progress reviews. Additionally I'd like to thank Russell for his comments on this thesis and both Russell and Riccardo for the informal discussions I had with them about my work.
- John Barnden for his interest in and discussions of my work.
- Stan Kwasny and Barry Kalman of Washington University, St. Louis, USA, for their valuable advice and help, for answering questions about their work and particularly for providing their simulator for training Sequential RAAMs with their training methods which is used in this work.
- The authors and maintainers of the PDP++ neural network simulator, and the members of the PDP++ discussion list for advice on the setting up and use of PDP++, which is also used extensively in this work for back-propagation training and the analyses presented here.
- The research students at the School of Computer Science, in particular the inhabitants of LG20, who provided a friendly atmosphere in which to work.
- My parents, family and friends for their support and advice and for helping me relax and enjoy myself when it was needed.

The work presented in this thesis was supported by a Research Studentship from the School of Computer Science, University of Birmingham.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The connectionist approach to AI

1.1.1 The connectionist revival

In 1986, Rumelhart and McClelland published the now famous "Parallel Distributed Processing" (Rumelhart and McClelland, 1986) collection of papers, marking the revival of connectionism as a methodology for Artificial Intelligence (AI). Since then, connectionism has become an established field within AI, and has mounted a challenge to the traditional symbolic approach. Connectionism has been applied to subjects such as natural language processing (Sharkey and Sharkey, 1992; Miikkulainen, 1993), common-sense reasoning (Sun, 1992), the modelling of memory and language (Levy et al., 1995) and various other areas of AI and Cognitive Science. Many connectionist researchers believe that connectionism may be able to replace the symbolic approach, although more recently interest has grown in hybrid approaches (Sun and Bookman, 1995).

1.1.2 The connectionist/symbolic debate

Ever since the revival of connectionism, an ongoing debate has been raging as to whether connectionism can or should replace the symbolic paradigm that dominates AI. Proponents of the symbolic paradigm launched an attack on connectionism questioning whether it was even capable of providing a viable architecture of the mind. For example, Fodor and Pylyshyn (1988) famously argued that connectionist architectures were not capable of pro-

viding the compositionality and systematicity they believed were essential in an architecture of cognition, and which symbolic methods provide naturally. They further argued that were a way to be found to endow connectionist systems with symbol processing capabilities, it would merely involve a reimplementation of classical symbol processing and therefore offer nothing new (except perhaps in providing an account of how symbol processing can be implemented in a "neural" architecture).

In response to the attack by Fodor and Pylyshyn, connectionists have produced a considerable body of research in which neural networks have tackled "symbolic" tasks or have been endowed with symbol processing capabilities. For example Pollack's Recursive Auto-Associative Memory (RAAM) (Pollack, 1990), Touretzky's BoltzCONS (Touretzky, 1986), Smolensky's Tensor Product representations (Smolensky, 1990) and Plate's Holographic Reduced Representations (Plate, 1994; Plate, 1995) all provide methods for representing symbolic structures such as lists and trees within connectionist networks. Considerable connectionist research has focussed on the traditionally symbolic field of natural language processing, with parsing (Miikkulainen, 1993; Henderson, 1994; Berg, 1992), script processing (Miikkulainen, 1994), grammatical inference (Maskara, 1993; Zeng, Goodman and Smyth, 1994), machine translation (Chandola and Mahalanobis, 1994) and language generation (Ward, 1994) amongst the areas that have been tackled.

1.1.3 Connectionist symbol processing and holistic computation

In response to the charge that the above responses merely involve connectionist implementations of classical symbolic methods, it has been argued that some of these methods (in particular the RAAM, Tensor Products and HRRs) support a new form of symbol processing known as *holistic computation*. In traditional symbolic methods, symbolic structures such as parse trees are operated on atomistically, symbol by symbol. With holistic computation it is claimed that these symbol structures can be acted upon holistically, without search, with the potential for performing complex structure sensitive operations more efficiently. For example, Chalmers performed active to passive transformations holistically (Chalmers, 1990a; Chalmers, 1990b). Holistic computation has an important role in the connectionist/symbolic debate as it seems to offer a form of symbol processing which traditional

symbolic approaches apparently do not support, and provides the basis for the most promising refutations of the arguments of Fodor and Pylyshyn yet. Indeed its role in the debate between connectionists and symbolists has been made explicit in the work of Chalmers and more recently in the work of Niklasson and Sharkey (Niklasson and Sharkey, 1997), who demonstrated how the RAAM can develop a compositional representation which supports systematic holistic operations, using a network trained to perform logical inferences on RAAM representations to implement de Morgan's laws (e.g. converting logical implications into the equivalent logical disjunctions).

1.2 Motivations

This thesis is above all a contribution to connectionism. The approach is a computer science/engineering approach rather than one grounded in psychology. No claims as to the psychological or biological plausibility of the work presented here are being made, as the work is an investigation of the computational capabilities of neural networks. More specifically it is an investigation into holistic computation, in particular the holistic processing of symbolic structures, and this section will explain the motivations behind the work presented in this thesis, and why this topic was chosen.

1.2.1 Background motivations

The main background motive for this work is the author's belief that connectionism is a valuable approach to AI. For this author, connectionism's value lies as much in its challenge to symbolic approaches and the work in connectionist and symbolic AI that this challenge has inspired, as in the explicit adherence to computational architectures inspired by the organisation of neurons in the brain. The author's desire to evaluate and explore the potential contributions of connectionism follows naturally from this belief in the value of connectionism. The work presented in this thesis is thus an explicit attempt to evaluate and explore one of the contributions of recent connectionist research.

1.2.2 Why a thesis on holistic computation?

As indicated above, holistic computation has played an important role in the connectionist responses to Fodor and Pylyshyn's attack. The importance of the debate between connectionists and symbolists about the status of connectionism is considerable, as it is about the very viability of connectionism as an approach to AI and Cognitive Science. To someone who believes, as this author does, that connectionism is a valuable approach, an attack on its viability requires serious consideration. Holistic computation appears to this author to be the most promising basis for a refutation of this attack and is thus of considerable interest. It should also be noted that holistic computation may be the most important contribution yet that connectionism has made, as it offers the ability to perform complex structure-sensitive operations without search, and thereby the potential for more efficient means of reasoning. As it also appears to be something which classical systems cannot do, it provides a reason for preferring connectionist approaches to symbolic approaches.

In the light of the importance of holistic computation indicated above, the author believes that the time is ripe for a critical examination of the concept in order to clarify precisely what it is, to investigate how easily connectionist systems can exploit it, and to investigate the potential for moving beyond the simple, "in principle" tasks that characterise much of the work that has been done thus far. The work presented in this thesis is intended to make a start at such a critical examination of holistic computation, to see how well a particular connectionist representation, the Sequential RAAM, (Pollack, 1990; Kwasny and Kalman, 1995) can develop representations that support holistic transformations on symbolic structures and to analyse the behaviour of the SRAAM to better understand what allows it to develop such a representation and whether it is likely to scale-up to more complex tasks than the simple "in principle" tasks that many researchers have demonstrated thus far. Such a critical look at holistic computation and connectionist systems' capacity to exploit it is necessary if it is to develop from being an interesting idea, to something which can be exploited in the development of practical systems.

Finally for this section it should be noted that this work is *not* intended to be a comprehensive refutation of the arguments of Fodor and Pylyshyn but rather to investigate holistic computation because of its importance to the debate their paper initiated and because of its importance to connectionism

otherwise. Those interested in the debate itself are referred to the papers by Smolensky and by Fodor *et al.* from the collection of papers in MacDonald and MacDonald (1995), the work of Robert Hadley (1992), and the work of Lars Niklasson, Noel Sharkey and Timothy van Gelder (Niklasson and Sharkey, 1997; Niklasson and van Gelder, 1994; van Gelder, 1990).

1.3 Aims

The aims of this thesis are as follows:

1. To clarify the nature and definition of holistic computation by answering the following questions:

    - What is holistic computation?
    - In the light of Balogh's critique of Chalmers' work (Balogh, 1994), does the work which claims to demonstrate holistic computation in fact do so?
    - Are distributed or holistic representations necessary for holistic computation?
    - Is holistic computation unique to connectionism?

2. To investigate how well the representations generated by the Sequential RAAM can support the holistic processing of symbol structures as follows:

    - An attempt to recreate Chalmers' (1990a) experiment using the Sequential RAAM.
    - Analysing the behaviour of the SRAAM in the above experiment to investigate whether the technique can scale up and whether it can be improved.

These aims embody the motivations mentioned above, by providing a start to the more critical investigation of holistic computation argued for in Section 1.2.2, a look at how easily a particular connectionist representation can support holistic symbol processing and an analysis of the technique to indicate whether it can easily be used to go beyond the "in principle" tasks of much earlier work.

1.4 Overview of thesis

- Chapter 2 surveys the literature on connectionist representations and holistic computation and argues that much of the literature uses confused notions of holistic computation. It then proposes and defends a new definition of holistic computation, and re-evaluates earlier work in light of the new definition, in particular Balogh's critique of Chalmers' work (Balogh, 1994). This chapter serves a starting point to the critical evaluation of holistic computation argued for in Section 1.2.2.

- Chapter 3 describes an attempt to recreate Chalmers active-passive transformations with the Sequential RAAM taking place of the RAAM that Chalmers used. The aim is to investigate whether the SRAAM can generate representations that can support holistic symbol processing at least as easily as the RAAM.

- Chapters 4 and 5 present an analysis of the Sequential RAAM's behaviour in trying to learn the task in Chapter 3, in order to determine the nature of its solutions, whether the technique can scale up and whether it can be improved. This analysis will indicate whether the SRAAM can be used for more than the "in principle" tasks used in much of the literature.

- Chapter 6 summarises what has been achieved by the work presented in the earlier chapters, and evaluates the work and its implications for connectionism, symbolic approaches, exploiting parallelism and the implications for AI generally. It then discusses possible areas of future work.

Chapter 2

Holistic Computation & Connectionist Representations

2.1 Introduction

This chapter[1] presents a brief discussion of distributed representations and then reviews the literature on holistic computation, arguing that the concept has been only vaguely defined in the literature leading to some confusion about what exactly it is and what makes it possible. The new definition presented here aims to clarify the nature of holistic computation. Finally the need to investigate how best to exploit holistic computation with neural nets is noted, and the contribution of the work in Chapters 3, 4 and 5 to this investigation is explained with reference to a set of questions about holistic computation and neural networks.

2.2 Distributed representations and holistic computation

Since the 1980s' revival of connectionism, networks that employ *distributed representations* (Hinton et al., 1986; van Gelder, 1991), as opposed to the *localist representations* (Feldman and Ballard, 1982; Waltz and Pollack, 1985; Selman and Hirst, 1985; Cottrell, 1985) that are also commonly used, have been a notable feature of much connectionist work. Indeed many of the repre-

[1] Chapter 2 is based substantially on the work presented in Hammerton (1998)

sentational techniques discussed in Section 2.3 are descendents of the earlier work on distributed representations. Where localist representations would employ an individual unit to represent some object, distributed representations employ a pattern of activity over a set of units to do so. Distributed representations are also divided into symbolic and subsymbolic distributed representations (see Sharkey (1991)). With symbolic distributed representations the individual elements may stand for features of the represented object, where with the subsymbolic distributed representations the individual elements do not code for specific features. The appeal of distributed representations lies in natural or easily achieved properties such as the more efficient use of the available units, support for generalisation, support for learning representations and the creation of content addressable or associative memories. Another aspect of distributed representations is that they can involve all the units used by them in representing the objects that can be encoded with them, with each unit being involved in the representation of all objects and all objects being represented across all units. When this occurs the term "holistic representation" is often used (see Section 2.4). For this reason distributed representations and holistic computation have been discussed as if where one occurs the other occurs as well, or as if they are intrinsically linked.

As argued in Section 2.6 and Section 2.7, the author believes that holistic computation does *not* require distributed representations, and that it is possible to perform holistic computations on localist representations under some circumstances. The holism that many researchers have had in mind regarding distributed representations, is holistic *representation* rather than holistic *computation*. Whilst holistic representation can facilitate holistic computation, holistic computation does not require holistic or distributed representations. For this reason and also because the focus of this thesis is on holistic symbol processing[2] which has only appeared since the late 1980s, the author feels a detailed discussion of the earlier literature on distributed representations is of limited relevance to the issues of this thesis, and thus refers interested readers to Sharkey's review of connectionist representations (Sharkey, 1991) for a more detailed discussion of the earlier literature than is presented here. Discussions of the nature of distributed representation can be found in (Hinton et al., 1986; van Gelder, 1991) and Chapter 3 of Balogh's thesis (Balogh, 1994) (the framework Balogh proposes is used in Section 2.4).

[2]See Section 2.7 for the distinction between this and holistic computation.

2.3 A survey of holistic computation

In Section 2.6 it is claimed that holistic computation requires representations where all the information necessary to perform an operation is directly accessible (that is accessible without search) to the processes operating upon it. Many connectionist representations, being of fixed length and making information about the constituents of the object directly accessible to other neural nets, meet these conditions.

This section reviews the literature on holistic computation and those connectionist representations that support it.

2.3.1 RAAM-based holistic computation

The bulk of the literature on holistic computation employs the Recursive Auto-Associative Memory (RAAM) (Pollack, 1990) to produce representations of symbolic structures which are then operated on by another neural network (Chalmers, 1990a; Ho and Chan, 1995; Sharkey and Sharkey, 1992). Much of the rest of the literature employs derivatives of the RAAM in a similar manner (Chrisman, 1991; Kwasny and Kalman, 1995; Sperduti, 1993a).

The RAAM is one of the earliest and most promising techniques for representing compositional structure. A ternary RAAM is depicted in Figure 2.1. In operation, the bottom half of the RAAM forms an encoder and the top half of the RAAM forms a decoder. To encode a tree, one starts by placing the leaves of a leaf-node of the tree in the banks of inputs and feeding the activation forwards to the hidden-layer. The hidden layer pattern can then be used in one of the banks of inputs at a later stage either alongside other hidden-layer patterns or the leaves of other leaf-nodes. Repeating this process, eventually a single hidden-layer pattern is produced which represents the entire tree. This can then be fed into the decoder, which produces the child nodes of the root node at its outputs. These can either be terminals or further hidden-layer patterns. If the latter are produced, the hidden layer patterns are fed back into the hidden-layer and the activation propagated forwards again to produce more nodes on the outputs. This continues until all the leaf nodes have been recovered. The RAAM architecture can be modified to have more banks of inputs and outputs, thus allowing N-branching trees where N is the number of banks. Training of the RAAM involves presenting the symbols and relevant hidden layer patterns to be encoded at each step as both the input and target output patterns.

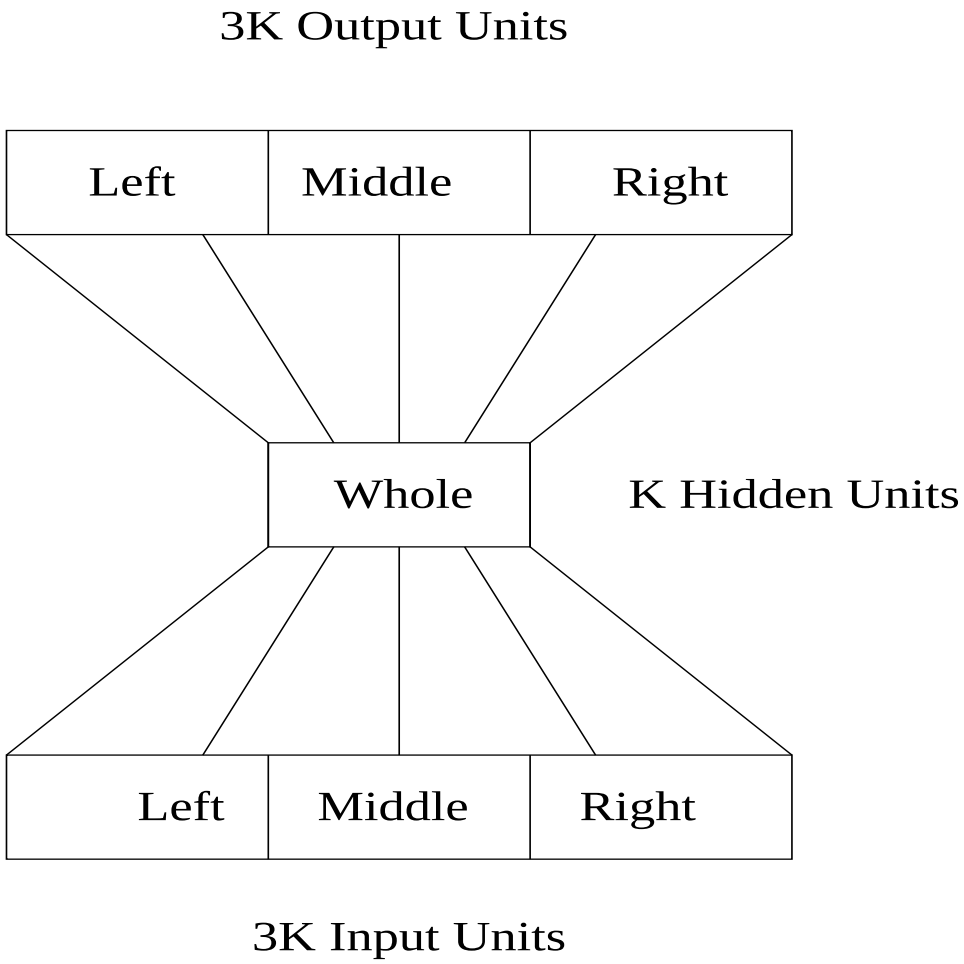

Figure 2.1: A ternary RAAM, capable of encoding ternary trees.

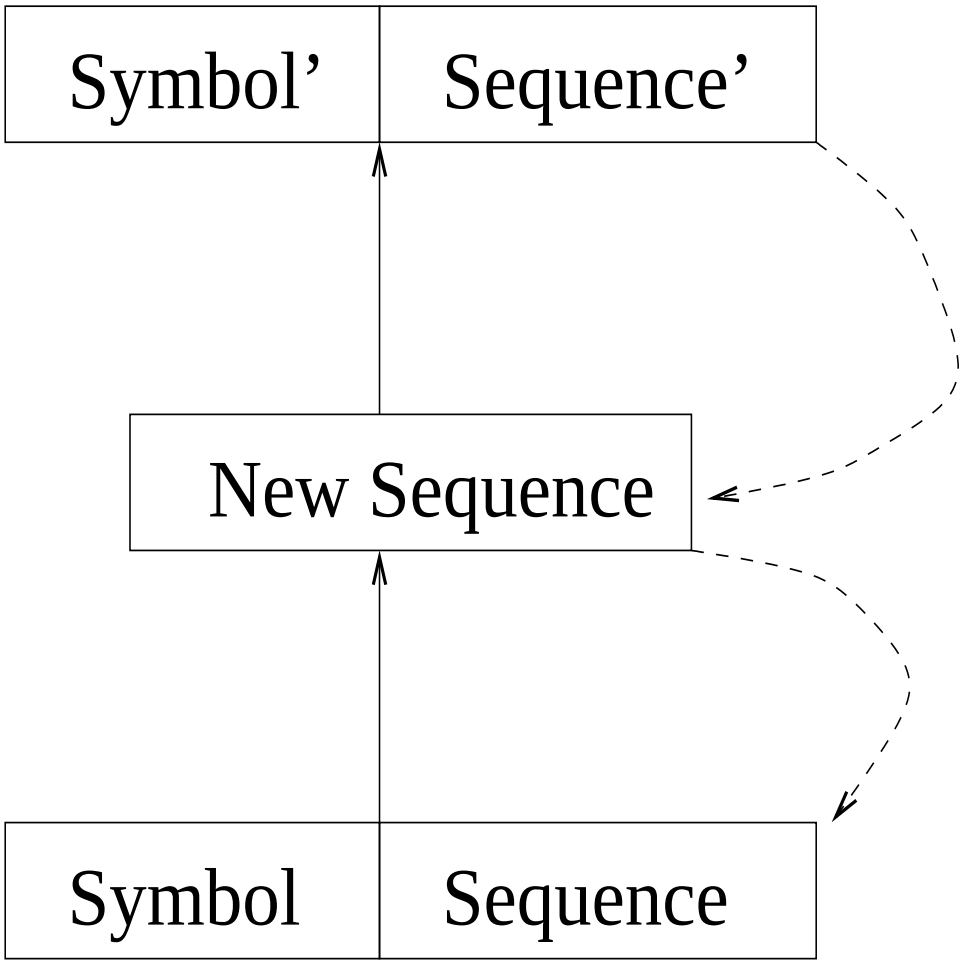

Figure 2.2: The SRAAM. Capable of encoding trees.

The Sequential RAAM (or SRAAM) is a variation of the RAAM trained to encode sequences of symbols rather than to encode a tree. Figure 2.2 depicts the SRAAM. The inputs/outputs each have two sections: the symbol to be encoded or that has been decoded, and the hidden layer pattern representing the remainder of the sequence. To encode a sequence, the encoder (bottom half of the SRAAM) is used. The first symbol and empty sequence value (e.g. all zeroes), are placed in the input and context layer respectively and the activation propagated to the hidden layer. The hidden layer pattern is then copied back into the context layer and the next symbol placed into the input layer. The activation is then propagated forwards again and the process repeated until all symbols have been presented. The final hidden layer pattern produced is thus the encoding for the complete sequence. The top half of the network is the decoder. Decoding proceeds by taking the encoding of a complete sequence and placing it into the hidden layer. The activation is propagated forwards to produce the last symbol added to the sequence and the encoding of the rest of the sequence. The encoding of the rest of the sequence is then fed into the hidden layer and the process repeated until the empty sequence and first symbol of the original sequence are produced. Training proceeds in a similar manner to the RAAM, with the symbol and relevant hidden layer pattern presented to both the inputs and target outputs at each step.

The RAAM has been used to represent phrase structure trees in connectionist parsers (Reilly and Sharkey, 1992; Sharkey and Sharkey, 1992; Ho and Chan, 1995) and to represent logical terms for various inference tasks (e.g. converting implications to their equivalent disjunctions (Niklasson and Sharkey, 1997)). The SRAAM has been used as an alternative to the RAAM for representing phrase structure trees by Kwasny & Kalman (Kwasny and Kalman, 1995) who claimed more straightforward training (e.g. use of a slightly modified Simple Recurrent Network (SRN) training algorithm) and better generalisation as advantages over the RAAM. Ho and Chan have also employed the SRAAM to represent input sentences and phrase structure trees (Ho and Chan, 1996).

The RAAM and the SRAAM are variations of essentially the same architecture. This architecture has also been extended to derive new techniques and used as the inspiration for an analytically derived technique. Sperduti (1993a) extended the RAAM architecture to encode labelled graphs, naming the technique Labeling RAAM (LRAAM), and this has been used in the classification of terms (Sperduti, Starita and Goller, 1995). Callan and

Palmer-Brown (1997) derived a technique called the Simplified RAAM, or (S)RAAM, analytically by noting that auto-associative networks essentially perform a form of principal components analysis (PCA) on their inputs and then basing the technique on recursive application of PCA. The (S)RAAM achieves training times of minutes compared to the hours the RAAM normally take. However it does depart somewhat from the standard connectionist paradigm in that it is not based on error minimisation; it does not employ units connected by links which have weights on them, and the learning component in it is fairly minimal. In the experiments reported by Callan and Palmer-Brown the (S)RAAM achieves 100% generalisation and it is possible to predict what sort of structures it will generalise to, as they are linear combinations of those it has been trained on.

Holistic computation with the RAAM and its derivatives

Chalmers (1990a) demonstrated the possibility of holistic computation by training a ternary RAAM to encode ternary trees representing active and passive sentences. The vocabulary consisted of five proper nouns and five verbal units (each in a passive or active form), giving a total of a possible 125 active and 125 passive sentences. The network was trained to take an active sentence as input and transform it into its passive equivalent (for example "John loves Helen" becomes "Helen is loved by John"), and another network was trained to perform the reverse transformation. Chalmers claimed that the experiment was a demonstration of the transformation of symbol structures without the need to decompose them into their constituents. He also claimed that the representations developed by the RAAM were distributed and, in reference to the explicit tokening of constituents in classical symbol structures, that *"In the distributed representations formed by RAAMs, there is no such explicit tokening of the original words"* (from the reprint in Sharkey (1992), page 54). The implication appears to be that the lack of explicit tokening in the representation and its distributed nature indicate that the transformation is accomplished holistically. Later work based on the RAAM that has sought to demonstrate holistic computation has included Niklasson and van Gelder's application of holistic computation to the task of deducing logical disjunctions from their equivalent implications (and the reverse deductions) (Niklasson and van Gelder, 1994).

The SRAAM has also been used in demonstrations of holistic computation. Blank et al. (1992) explored holistic computation using the SRAAM

in a similar but more systematic way than that used by Chalmers. They first trained a SRAAM to encode and decode sentences of 2 or 3 words in length generated by a simple grammar and then trained several networks to perform holistic operations. These operations fell into three categories: *detectors*, that simply detected features of sentences (e.g. that the subject and object were identical); *decoders*, that attempted to extract constituents directly from the hidden layer representation directly without first going through the iterative decoding process usual to the SRAAM, and finally, *transformers* which changed the sentence in some way, e.g. from *X chases Y* into *Y flees X*. Once trained, the transformers achieved 100% generalisation for sentences which had been in the SRAAM's training set but not the transformation network's training set, and 75% for sentences which were in neither training set. In spite of the small testing set (4 sentences novel to both the SRAAM and the transformer, and 4 sentences which were novel only to the transformer), this work is significant for demonstrating a range of possible operations for holistic computation. However the small size of the training sets and the simplicity of the tasks leave unanswered questions as to whether such techniques could scale up to real world applications. More recently, Kwasny and Kalman (1995) trained a network to detect the presence of certain symbols within a sequence encoded by an SRAAM, obtaining a 90% success rate. They also tried to see if a subsequence of symbols could be detected within a sequence encoded by an SRAAM, but only obtained a 60% success rate. These mixed results suggest further work would be needed in order to determine whether the SRAAM is well suited to holistic computation. Since Kwasny and Kalman reported faster, more straightforward training and a higher rate of generalisation than with the RAAM this may be worth investigating.

Callan and Palmer-Brown's demonstrations of the (S)RAAM included a reproduction of Chalmers' experiment, in which the (S)RAAM obtained 100% generalisation, compared to the 83% achieved in Chalmers' experiment using RAAMs. Given the speed of training of the (S)RAAM, and its impressive performance, it seems to be a strong candidate for use as a vehicle for holistic computation. However, as noted above, it departs from many aspects of typical connectionist techniques especially in not employing the networks of simple processing units that connectionists postulate as being the architecture of cognition.

Chrisman's dual-ported RAAM and confluent inference

Chrisman (1991) extended the architecture of the RAAM/SRAAM by introducing the dual-ported RAAM. (Figure 2.3 depicts a dual-ported SRAAM which exemplifies the extension.) The idea is that the left-hand side is used to encode the SRAAM representations of the untransformed structure and the right-hand side to encode the representations of the transformed structure. When training, each side is trained separately until there is a complete encoding for the untransformed structure and the transformed structure. The error is then propagated from the outputs of one side to the inputs of the other. The end result is that the SRAAM encodings of the untransformed structure and the transformed structure are identical, since the same hidden layer is shared by both SRAAMs. Chrisman described the inference performed by the dual-ported RAAM as *confluent inference*. He drew a distinction between confluent inference and the inference done by the transformation network in Chalmers' experiment, which he termed *transformational inference*. The crux of the distinction lay in Chrisman's observation that transformational inference has representations which are developed independently of the holistic inferences which will be performed in them. He suggested that superior performance can be achieved by allowing the representations to be influenced by the inference task to be performed.

Chrisman presented confluent inference as a form of holistic computation. In his work, he demonstrated confluent inference by training the dual-ported RAAM with a simple set of English sentences and their Spanish equivalents and then performing a simple translation task. It would be possible to reproduce Chalmers' active/passive experiment by training one side to encode the active sentences and the other to encode the passive sentences. The network would then be capable of performing both the active to passive and passive to active transformations.

While confluent inference is certainly an interesting concept, it is questionable whether the transformation is being performed holistically. The dual-ported RAAM develops a representation that can be decoded into either the untransformed object or the transformed object depending on which decoder is used. However once the object to be transformed is encoded, *no transformation is required* to obtain the encoding of the transformed object. This is because the representation of the object is identical to the representation of the transformed object. Confluent inference can be used where an explicit transformation phase is required (see for example the work of Niklas-

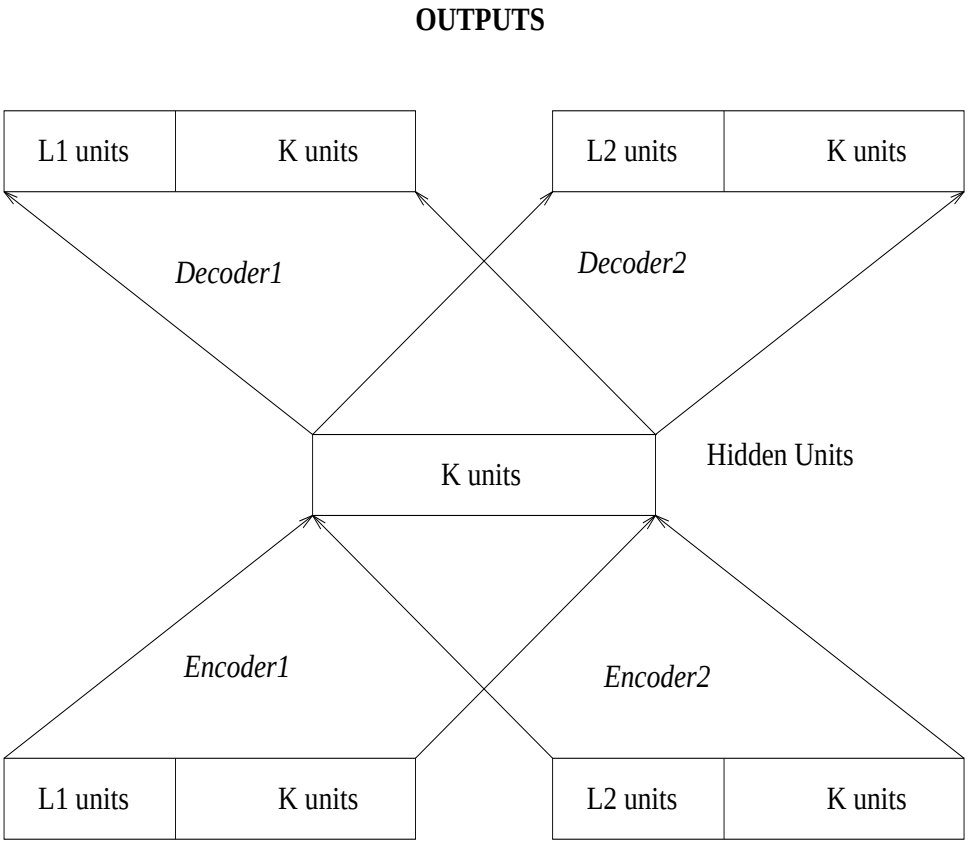

Figure 2.3: A dual-ported RAAM.

son and van Gelder (1994)), but does not seem to involve the same processes under which holistic computation has been described in other work. Confluent inference is an interesting way of training a network to perform a task and has even been used for parsing (Ho and Chan, 1995; Ho and Chan, 1996) but no transformation of the encoding of a structure is done. The transformation task is instead learned by the encoders and decoders, and performed in the iterative encoding and decoding of the structures. This suggests the transformation of the symbol structure is not being performed holistically, though the encodings developed might be capable of being operated upon holistically. Confluent inference thus seems to be a method of learning representations that incorporate specific information for specific purposes, rather than a method for performing holistic transformations.

2.3.2 Holistic computation and other connectionist representations

Whilst the RAAM or its derivatives are the most commonly used representations in the literature on holistic computation there are other representations capable of supporting holistic computation. This section looks at two of the most promising of these representations.

Holographic Reduced Representations(HRRS)

HRRs (Plate, 1991; Plate, 1993; Plate, 1994) are distinct from the RAAM and its derivatives in this review in as much as they are hand-designed methods of representing structures in fixed-width vectors[3]. The important point is that the representation of a few or a large number of inputs would remain of fixed size and would not expand with the expansion in the number of inputs.

HRRs have not been the subject of practical experiments in the same way as the RAAM and the SRAAM, but it is reasonable to consider their potential for holistic computation from a more theoretical viewpoint. Firstly, HRRs employ circular convolution to "bind" two vectors together and superposition (vector addition) to store multiple bindings within a single vector. Given vectors $\mathbf{a}$ and $\mathbf{b}$ of size n, the circular convolution of these vectors is:

$$\mathbf{c} = \mathbf{a} \otimes \mathbf{b}$$

[3]The (S)RAAM is perhaps intermediate between HRRs and the RAAM in this respect

Where:

$$c_j = \sum_{k=0}^{n-1} a_k b_{((j-k) \mod n)}$$

Circular correlation can be used as an approximate inverse to circular convolution if the base vectors have elements independently distributed with a mean of zero and a variance of $1/n$. Given vectors a and b the circular correlation $\mathbf{c}$ of the two is given by:

$$\mathbf{c} = \mathbf{a} \oplus \mathbf{b}$$

Where:

$$c_j = \sum_{k=0}^{n-1} a_k b_{((k+j) \mod n)}$$

Given a binding of two vectors, one can extract one vector by computing the circular correlation of the other vector with the binding. Given a superposition of bindings one can extract one vector from one of the bindings with noise added by correlating the superposition with the other vector in the appropriate binding. Alternatively, circular correlation of one vector with another is equivalent to circular convolution of the involution of that vector with the other, i.e. $\mathbf{a} \oplus \mathbf{b} = \mathbf{a}^* \otimes \mathbf{b}$, where $\mathbf{a}^*$ is the involution of $\mathbf{a}$, and $a_j^* = a_{(-j \mod n)}$. With large vectors (e.g. 512 elements) this noise is negligible. Also convolving a vector with its involution yields an approximate convolutive identity vector. To represent the sequence "a b c d", one would create a set of position vectors $\mathbf{p_i}$[4], where $i = 1, 2$ or 3, and create the vector:

$$\mathbf{t} = \mathbf{a} + \mathbf{p_1} \otimes \mathbf{b} + \mathbf{p_2} \otimes \mathbf{c} + \mathbf{p_3} \otimes \mathbf{d}$$

Note that all these vectors have the same number of elements. To extract $\mathbf{b}$ from $\mathbf{t}$ you would convolve $\mathbf{t}$ with $\mathbf{p_1^*}$, the involution of the vector $\mathbf{p_1}$. As for vector $\mathbf{a}$, $\mathbf{t}$ happens to be $\mathbf{a}$ with noise added.

With HRRs, it is possible to create vectors which (when convolved with the vector representing some structure) will move constituents around, replace them, or remove them, thus allowing structure transformations to occur

[4]The position vectors are usually formed by taking a base vector $\mathbf{p}$ and using convolutive powers of $\mathbf{p}$, e.g. $\mathbf{p_2} = \mathbf{p} \otimes \mathbf{p}$.

as the result of a single convolution operation. In other words, holistic computation is possible. For example, consider transforming $\mathbf{t}$ to the vector:

$$\mathbf{s} = \mathbf{a} + \mathbf{p_1} \otimes \mathbf{d} + \mathbf{p_2} \otimes \mathbf{c} + \mathbf{p_3} \otimes \mathbf{b}$$

In order to do this, $\mathbf{t}$ is convolved with the vector:

$$\mathbf{u} = \mathbf{p_3} \otimes \mathbf{p_1}^* + \mathbf{p_1} \otimes \mathbf{p_3}^*$$

This is because:

$$\mathbf{t} \otimes \mathbf{u} = \mathbf{a} + \mathbf{p_3} \otimes \mathbf{p_1}^* \otimes \mathbf{p_1} \otimes \mathbf{b} + \mathbf{p_2} \otimes \mathbf{c} + \mathbf{p_1} \otimes \mathbf{p_3}^* \otimes \mathbf{p_3} \otimes \mathbf{d} + \mathit{noise}.$$

This reduces to:

$$\begin{aligned} \mathbf{t} \otimes \mathbf{u} &= \mathbf{a} + \mathbf{p_3} \otimes \mathbf{b} + \mathbf{p_2} \otimes \mathbf{c} + \mathbf{p_1} \otimes \mathbf{d} + \mathit{noise} \\ &= \mathbf{a} + \mathbf{p_1} \otimes \mathbf{d} + \mathbf{p_2} \otimes \mathbf{c} + \mathbf{p_3} \otimes \mathbf{b} + \mathit{noise} \end{aligned}$$

Note that $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ need not be base vectors: they could represent arbitrary structures, such as entire noun phrases and verb phrases. Thus an arbitrarily complex transformation can be performed with a single convolution. Since a structure can be transformed through a single convolution with an appropriate transformation vector, it is claimed that HRRs support holistic computation in a fairly direct manner. If the transformation only requires the extraction of constituents from known positions, the movement of constituents at known positions and/or the replacement of constituents with others that can be known in advance without iterative search, then the transformation vector can be constructed once and used with all the appropriate structures without the need for further change.

Weber's TCO representations

Weber (1992) claims perhaps the most impressive demonstration of holistic computation, employing a derivative of the HRR based on an operation he calls the Term Coding Operation (TCO). This operation combines vectors together using the circular convolution that Plate uses, but in a manner that allows the vectors to be much smaller: 32 elements in the work Weber presents. Using this operation Weber builds up representations for logical terms, which he then uses to train a feed-forward network employing error

backpropagation to perform the logical unification of the terms. The terms to be unified are presented as inputs, and the outputs include the unified term, the most general unifier and any uninstantiated variables. Weber tested this unifier in a Prolog system, and it correctly performed over 65,000 different unifications. The unifier is however only a heuristic in that it is not proven correct – indeed there is a limit to the size of the terms that can be encoded in a TCO representations, thus it cannot unify completely arbitrary terms. However it seems to be able to unify correctly within the range of terms that can be represented by the 32 element vectors Weber uses, and may have application in areas where strict correctness is not necessarily required, such as natural language processing. Impressive as this is, the TCO representation is like the HRR in that it is a hand-designed representation, and thus does not employ connectionist learning to create the encodings for structures in the way that a RAAM does.

2.3.3 Conclusions

The distinctive aspect of the work presented above is the ability to transform or make inferences about the encoding of a structure directly, without the necessity of searching a structure to locate constituents or to extract constituents from the encoding before processing. This ability seems to be the hall mark of holistic computation. Many of the examples given in the literature have been limited and raise the question as to whether they are capable of being scaled-up. The more promising examples employ representations that depart in some way from normal connectionist techniques. For example the (S)RAAM, HRRs and TCOs all seem to employ representations which throw away some aspects of the normal connectionist approach. The (S)RAAM does not employ error minimisation or networks of interconnected units whilst HRRS and TCOs, which equally do not employ networks of interconnected units, do not employ learning of any sort as well. This raises the question of whether representation which are learned could support more sophisticated examples of holistic computation, such as the term unification that Weber demonstrated.

Confluent inference seems to offer some promise too; however it is questionable whether it can be regarded as holistic computation. In spite of this reservation, confluent inference seems to have potential as a tool for developing representations that support holistic computation. Much of the work presented above is open to the criticism that there has been a lack of analysis

of the *nature of the representation* developed. HRRs are, by design, a highly distributed representation and Chrisman, and Blank *et al.* conducted some limited investigation into the nature of their representations. The significance of the concern with the nature of representations stems from the view that the RAAM (and other methods) develop distributed representations that do not contain explicit tokening of constituents and it is this that allows holistic computation to take place.

The work reviewed above has not directly addressed the question of how distributed RAAM representations are. A direct analysis, conducted by Imre Balogh, of the representations developed by the RAAM used in Chalmers' experiments demonstrated that the representation was a strongly localist representation with explicit tokening of the constituents of the structure being represented, thus casting doubt on whether Chalmers did demonstrate holistic computation. This analysis is discussed in the next section.

2.4 Balogh's analysis of Chalmers' experiments

Balogh (1994) decided to tackle directly the question of whether the RAAM representations in Chalmers' experiments were distributed. In Chapter 3 of his thesis, Balogh defines what it means for a representation to be distributed: A structured representation needs two types of information, information about its constituents and information about the structure itself. A localist representation is where the basic constituents are totally circumscribable, non-overlapping entities. Each atom has a characteristic function which when given a position returns the activity at that position and, in a localist representation, this function can either be 0 or 100% indicating the total absence or total presence of the atom at that position and only one atom will be allowed to have a value of 100% at that position. This would be a representation that involved explicit tokening of its constituents. For a distributed representation, values between 0% and 100% are allowed, and more than one atom could have a non-zero value for that position. From this idea Balogh distinguishes four types of representation: Singleton Localist (SL) where a single unit represents a single atom; Circumscribable Localist, where more than one unit can represent a single atom but each unit only participates in representing one and only one atom; Circumscribable Distributed, where only a subset of units is used to represent an atom but these units may also play a part in representing other atoms; and Total Distributed, where all units

play a part in representing all atoms. To be sure these are not clean distinctions but blur into each other along a continuum from Singleton Localism to Total Distributedness. Totally distributed representations are also termed "holistic" representations[5]. This is because, in a holistic representation, one cannot isolate any part of the object being represented by isolating a subset of the representational units: altering a single unit in a holistic representation will involve making changes to *all* the constituents of the object being represented – the constituents of the object are not explicitly tokened in the holistic representation. Making systematic changes to the object being represented requires operating on the whole representation. Thus any changes to a holistic representation, whether to a single representational unit or to multiple units, will act upon all the constituents of the object being represented *simultaneously* and thus will act holistically upon that object.

Balogh proposed a way of testing whether any connectionist representation was distributed: lesion the units within the representation and note the errors that arise when the representation is decoded. With a distributed representation, lesioning units (e.g. setting their values to zero) one at a time should make little difference to the decoding of the representation and each unit should contribute equally to the error. With a localist representation, on the other hand, lesioning a unit might make an error in one single atom, while lesioning another unit wouldn't affect that atom but would cause an error with another atom. Lesioning increasing numbers of units in a distributed representation should see a gradual, smooth increase in the error rate showing graceful degradation, whereas in a localist representation, the error would increase in jumps. When Balogh applied this methodology to Chalmers' experiment he found that the experiment's RAAM representations exhibited localist properties: i.e. the error rates for different atoms varied considerably as each unit was lesioned. Further analysis showed that the representation was mainly Circumscribable Localist. Groups of units identified the contents of each position in the RAAM representation (i.e. they explicitly tokened the constituents of the sentence), and a separate unit indicated whether the RAAM represented an active or passive sentence. (There was some overlap for a couple of the units in determining the values of one of the slots.) Moreover, Balogh showed how the representations can be directly manipulated in a manner similar to that of classical symbolic representations. He suggested that holistic computation was not occurring in Chalmers' experi-

[5] *cf.* the concept of a *superpositional representation* described in van Gelder (1991)

ment. Balogh also argued that his results should apply to any RAAM-style network, such as those used in Chrisman's work and Blank's work.

Balogh's conclusion that holistic computation was not occurring depended on a simple and intuitive assumption; one that was not made explicit in Balogh's argument but which can be inferred from a detailed reading of the thesis. This assumption was that holistic computation requires holistic representations. Certainly with a holistic representation, any operation on a holistic representation acts on all the constituents of the object being represented *by the very nature of the representation.* One cannot change an isolated constituent of the object being represented by changing an isolated section of the representation. The fact that the representations developed by Chalmers' RAAMs are apparently not holistic thus leads one naturally to question whether holistic computation was really demonstrated in Chalmers' work, and further to question whether the other RAAM-based work demonstrated it either. Indeed these are the conclusions that Balogh draws. However, in addition to not making his argument explicit, Balogh does not explicitly state what he means by holistic computation. Indeed, as shall be seen, the literature on holistic computation generally describes it in rather vague terms.

It could be that holistic computation can be achieved without holistic representations but without an adequate definition for it, this cannot be judged with certainty. The next section will look at what has been said about holistic computation and the subsequent sections will present and defend a new definition of holistic computation proposed by the author. It will be argued that: firstly, Balogh was wrong (but this was because he made the same assumptions as the connectionists he criticises) and that Chalmers did demonstrate a form of holistic computation and, secondly, that holistic computation (whilst having been discovered by work with connectionist systems) is not restricted to connectionist systems.

Finally for this section, it should be noted that Balogh's analysis would not necessarily yield the same results with other RAAM-based work. It could be that the simplicity of the task in Chalmers' experiment allowed the RAAM to develop a localist solution to it, and that more complicated tasks might force the RAAM into a more distributed encoding, or an encoding that lacks explicit tokening of the constituents. Furthermore, the SRAAM is trained rather differently from the general RAAM in that one is encoding a sequence rather than a tree structure and thus there is no indication to the network of how to partition the hidden layer representation. This could also

lead to distributed representations since more and more symbols are being compressed into a fixed-width vector, without any indication of partitioning. Indeed Chrisman tested his SRAAM representations for micro-features (that is individual units that encode specific information) and found that whilst there was a slight tendency for micro-features to occur with small groups of closely related sentences, they did not occur consistently over the training set. This suggests that a mainly distributed representation was being created which occasionally included some information encoded in a more localist fashion. Thus, whilst Balogh's work is useful in clarifying the issues, demonstrating a way of analysing the representations developed by the RAAM, and showing that Chalmers' experiment did not develop holistic representations, more work is needed to determine firstly, whether this necessarily shows that holistic computation did not occur and secondly, whether the conclusions apply to other RAAM-based work.

2.5 Descriptions of holistic computation in the literature

2.5.1 Chalmers' descriptions

Chalmers (1990a), described holistic computation, and indicated what he thought the nature of the RAAM representations was, as follows:

> *"Connectionism offers the opportunity to operate on compositional representations holistically without first proceeding through the step of extraction"*. (page 54/47) [6]
>
> *"Here we will model the process of syntactic transformation by operating directly on the compressed distributed representations of sentences and without passing through any stages of composition or extraction"*. (page 62/55).

From this we can see that Chalmers regarded holistic computations as transformations which can be performed without the need to extract constituents and compose them into the transformed symbol structure. However,

[6] Chalmers' paper has been reprinted in *Connectionist Natural Language Processing:Readings from Connection Science* (Sharkey, 1992).The first page number is from the original paper, the second from the reprint.

he described the process in terms of what is happening to the representation rather than what is happening to the object being represented. He did not say how the process works "holistically". One can infer that he assumed it works holistically because he (mistakenly if Balogh is correct) believed the representations to be holistic themselves: however this was not made explicit. The description of holistic computation is vague and does not distinguish between how the representation is processed and how the object being represented is processed.

2.5.2 Chrisman's descriptions

Chrisman also described what he thought holistic computation (which he also refers to as holistic inference) entailed (page numbers from Chrisman (1991)):

> *"This form of inference occurs in a* gestalt *fashion by deriving a solution to the inference problem directly from a representation of structured data without decomposing, locating or accessing its constituent elements."* (page 346).
>
> *"Inferences that use a representation directly without accessing its compositional structure are said to perform holistic inference."* (page 349).
>
> *"By harnessing the emergent microstructure in these distributed representations, holistic inference maps directly from the representation of a problem to the representation of an answer, in a gestalt fashion without accessing the constituent elements or relations within the data."* (page 363).

What Chrisman appears to be saying here, is that a holistic inference acts on the reduced descriptions (i.e. the representations) of compositional structures directly without needing to decode them into the representations of the constituents of the structures. However there is vagueness and confusion within these quotations. There is vagueness because when Chrisman states that the processing occurs in a "gestalt fashion" it is not clear what he is referring to. Does he mean that all the constituents of the representation are processed simultaneously, or that all the constituents of the structure being represented are processed simultaneously? This is important in light of

the distinction between holistic and non-holistic representations, yet Chrisman does not say. There is confusion when Chrisman says that holistic inferences use a representation directly without accessing its compositional structure. Taken literally this means that the representation is transformed without accessing the constituents of the representation, which cannot be the case. However this probably not what Chrisman meant. The point Chrisman might be trying to convey (which is in line with the context of the quotations) is this: With the reduced descriptions created by the RAAM, the compositional structure of the *representation* need not bear a direct relationship to the compositional structure of the *object being represented* (e.g. the RAAM representation could be holistic) and, since changes are effected by making changes to the constituents of the representation, a holistic inference proceeds without directly accessing the compositional structure of the represented object. This point would have been clearer if Chrisman distinguished the processing of the representation and the processing of the object being represented.

2.5.3 Blank, Meeden and Marshall's descriptions

Blank et al. (1991) (published as Blank et al. (1992)) give perhaps the clearest description of what is meant by holistic computation in the following quotation:

> *If a composite structure in a symbolic system, say a Lisp list, were to have its fourth element tested for some criterion, one must first remove first three elements of the list to get to the fourth. In fact to do anything to the list involving the elements of that list, one must first decompose it. Thus, symbolic list structures can only be operated on* atomistically. *Since many AI practitioners have exclusively used concatenative data structures, the need for this initial deconstruction step seems to be a natural consequence of building data structures. In the subsymbolic paradigm however an operation can act* holistically *on the entire symbol structure... In this way, a subsymbolic operation can, in one step, perform a complex function without decomposing the representation of a symbol structure into its constituent parts.* (page 5)

A key point here is their description of a holistic operation as one that acts holistically on an entire symbol structure, meaning that the operation

acts on all the constituents of the structure simultaneously. More than the other descriptions, this one avoids confusing what is happening to the representation with what is happening to the object being represented. However, a question is begged as to what constitutes a single-step operation. One could suggest that a transformation of a vector is a single-step operation, but one could counter that it is made up of the operations of changing each element individually and thus it is not a single-step operation. Further one could even argue that changing an individual element is not a single-step operation at the level of what the microprocessor does to alter the individual element. With a holistic representation however, this would not matter — changing one element of a holistic representation involves changes to the whole object being represented (i.e. to the whole symbol structure). With a non-holistic representation it does make a difference though, since if a single element of the representation corresponds to a single constituent of the structure then changing elements one at a time would not involve acting on all the elements of the structure simultaneously. Parallel processing of all the elements of the vector on the other hand would. In fact this seems to suggest what the core idea of holistic computation should be seen as — the ability of a computational process to act on all the constituents of an object simultaneously and without the need to perform an iterative search to access those constituents.

2.5.4 Conclusions

The descriptions of holistic computation in the literature are vague and sometimes confusing. They do not adequately distinguish between the processing of the representation and the processing of what is being represented. They often describe the processes involved in terms of what happens to the representation or, worse, in terms which mix or confuse what happens to the representation with what happens to what is being represented. They do not explicitly make it clear whether the holism arises through the representation being holistic (though this is implied in much of the work) or through the use of a parallel architecture (this possibility is not even discussed). However something can be gleaned about what the authors of these descriptions are trying to get at. It seems holistic computation involves processes acting on all the constituents of an object simultaneously without the need to use iterative search/decoding to access those constituents. With these insights, a revised definition of holistic computation can be put forward.

2.6 Defining Holistic Computation: A proposal

> *A holistic computation is any computational process that can act on all the constituents of an object simultaneously, without requiring search to access those constituents.*

This definition[7] of holistic computation makes explicit what is being claimed, which is that in a holistic computation all the constituents of the object being processed are acted upon simultaneously without the need to search in order to access the constituents. The advantage of using a holistic computation is that since it eliminates the search required to access the constituents of a complex structure, it offers the potential for more efficient forms of processing that may also exploit parallelism effectively.

One way of achieving holistic computation has already been indicated. Using a holistic representation (e.g. HRRs) allows one to do precisely what this definition describes: act upon all the constituents of the structure simultaneously without searching to locate those constituents. Another way of achieving holistic computation is to devise a representation which makes all the information required for the transformation directly accessible (that is accessible without needing search) to the process operating on the representation and to use parallel processing of that representation to act on all the elements of the representation simultaneously. An important distinction arises here, which is holistic processing of a representation versus holistic processing of the object being represented. Holistic processing of the object being represented can be achieved either by processing (holistically or otherwise) a holistic representation of it, or by holistically processing the representation. Thus it is not necessary to have a holistic representation if one wishes to perform holistic computations.

Why should the above definition be preferred to earlier definitions? The main reason for doing so, is for *conceptual clarity*. By separating out holistic computation from holistic representation (and confluent inference), a clearer picture of what a particular system is doing (or can do) emerges. If the goal is to build a system which can perform structure sensitive operations on

[7]Hammerton (1998) defined a holistic computation as *any computational process that can act on all the constituents of an object simultaneously without needing to perform a search to locate or access those constituents.* The definition presented here is a more concise formulation.

symbolic structures without employing search (with the potential to be more efficient), the revised definition will indicate a wider and more accurate range of options for how this can be achieved. Earlier definitions, by confusing how the representation is being processed with how the object being represented is being processed, make it less clear what options are open to the system designer who wishes to exploit holistic computation. This is because whether the representation is holistic or not does not determine whether holistic computation is possible or not: yet the designer might believe they have to use a holistic representation because this is implied in much of the literature. What does matter is that all the information needed to perform the transformation is directly accessible to the computational process that is performing the transformation. Thus a fixed-length localist representation on a parallel machine can facilitate holistic computation, just as a holistic representation on either a sequential or parallel machine can.

2.7 Implications of the proposed definition

One implication has already been mentioned, that holistic computation does not require holistic or distributed representations. Further implications include:

- Many types of neural network perform holistic computation. A trivial form of holistic computation is supported by feedforward and simple recurrent networks. When the activation is propagated from the input layer to the output layer in feedforward networks, the input vector is transformed holistically into the output vector, since the network operates on the elements of the input vector in parallel. Similarly the input and context vectors in an SRN are combined and transformed holistically into the output vector of the SRN. The main focus of the work on holistic computation discussed above is not therefore on holistic computation per se, under the proposed definition, but on *holistic symbol processing*, i.e. making inferences about and/or manipulating symbol structures holistically. The main focus of this thesis is also on holistic symbol processing, as it is the harnessing of holistic computation for the processing/manipulation of symbol structures that is the real innovation provided by RAAMs, HRRs and other connectionist representations. The term "holistic symbol processing" will thus be used

through the rest of this thesis to refer to the holistic processing of symbol structures. Regarding the title of this thesis and later usage of the term "holistic computation", it should be noted that holistic symbol processing is a way of exploiting the holistic computations performed by neural networks on their input vectors.

- By itself, confluent influence only involves a trivial form of holistic computation. In the earlier discussion of confluent inference it was argued that the transformation was not performed holistically because once an encoding was produced no transformation was required to obtain the encoding of the transformed object and that the work of the transformation was performed during encoding and decoding. Thus in the light of the proposed definition of holistic computation, holistic symbol processing is not occurring although holistic computation is occurring in the trivial form described above, i.e. the transformation of the input vector to the output vector at each stage of encoding or decoding is an operation that is holistic with respect to the input vector. Of course if a form of confluent inference that involves an explicit transformation phase is used then holistic symbol processing may be occurring as a result.

- Chalmers *did* demonstrate holistic computation. This is obviously true for the trivial form of holistic computation mentioned above, but it is also true that Chalmers demonstrated holistic symbol processing. When Balogh suggested that holistic symbol processing was not occurring, he was assuming that by demonstrating that the representation developed by Chalmers' network was not holistic he was also demonstrating that sentences were not being transformed holistically. From the above arguments it can be seen that this is not so. The transformation network operated on each element of the RAAM representations in parallel and thus operated holistically in the required sense, or rather it would have been if implemented for real rather than being simulated on a sequential machine. However, since connectionists posit a highly distributed parallel architecture as being the architecture of the mind, the sequential implementation of such an architecture is beside the point. Thus Chalmers did demonstrate holistic symbol processing.

- Holistic computation is not unique to connectionism. Consider permuting an array on a parallel machine. If the machine has N processors,

where N is the size of the array, then each processor can be assigned an element in the array which it then moves to its new position. By doing this, the array would be acted upon holistically: every element of the array would be being processed simultaneously, without any searching being required to locate them. This is similar to the trivial form of holistic computation mentioned above. If the array was actually representing a list, then it could also be a simple example of holistic symbol processing.

- What is "new" about the connectionist examples of holistic computation discussed above is holistic symbol processing. The connectionists' innovation was to devise representations that allowed complex structures to be processed holistically. This innovation does have costs. Accuracy is lost in storing symbol structures in vectors leading to a "fuzzy" limit on the size of the structure that can be stored in any given vector. If the size of the structures that can be stored needs to be increased, then a new neural network needs to be constructed and trained to process the vectors representing the structures or, if using HRRs, new base vectors need to be created.

2.8 Classical representations and holistic computation

Traditional AI systems have almost exclusively used classical symbol structures such as lists and trees, and have not supported holistic symbol processing. One question which is raised is whether systems using such structures *could* support holistic symbol processing. At first sight the answer seems to be "no". Such systems operate on the symbol structures by finding and changing a symbol one at a time, and since the representations used are not holistic representations, then on a sequential machine there is thus no possibility of supporting holistic symbol processing. With standard implementations of the symbol structures it would not be possible on a parallel machine either.

All the above points relate to implementations of the symbol structures that closely reflect their high-level specification. For example, a list would be implemented using pointers and dynamic memory allocation each time an element is added or removed and, in order to get at an element, a search

which follows the pointers from the head of the list is required. However in principle the implementation need not be such a direct reflection of the high-level specification. Again taking the example of a list, an intelligent compiler for a parallel machine could implement it as an array, if it could deduce its likely maximum size from the code. During any periods when the list remains the same size, but where the elements of the list are changed, the compiler could parallelise the changes and thus the implementation could support holistic transformations of the list, given enough processors.

Thus it seems that implementations of at least some classical symbol structures could support holistic symbol processing. However, the scope for doing this is limited. It is not obvious, for example, how one could manipulate a graph or a tree holistically, unless the compiler were to implement the symbol structure using a holistic representation such as HRRs. However the compiler would then need to know the size of the largest structure likely to be used and the size of the representation that will hold such a structure. These of course need to be known anyway when using HRRs, but the programmer would be in a better position to know this information than the compiler, and it is unlikely that the compiler could get quite the same efficiency as the programmer in this respect. Also, a neural network can learn to do some types of transformation which a compiler would not be able to specify. For example, a neural net can learn a mapping between the representations of, for instance, sentences in two different languages: something which cannot be done using HRRs and other "hard wired" representations without performing a search of the sentences to locate the words/phrases and then finding their replacements.

The conclusion then is that whilst there may be some scope for implementing classical symbol structures in a manner which supports holistic symbol processing, the scope for doing so is limited, and in any case it is more efficient to use representations which support holistic symbol processing directly rather than make conversions.

2.9 Are computations on holistic representations always holistic?

This chapter claims that all computations on a holistic representation are necessarily holistic computations performed on the represented object. One

might object that, on a machine with virtual memory, accessing one of the constituents of a holistic representation might incur a search of page tables and/or the disk to find the relevant page of memory and, because the access involved search, the operation is not holistic according to the definition above. This would be incorrect, since the definition states that the computation is done without requiring search to access the constituents *of the object*, whereas the search here is to access the constituents of the representation. The above argument confuses the processing of the representation with the processing of the object. That is the object being represented is being processed holistically, even though the representation is not.

It might also be objected that, as with traditional symbol structures, the implementation need not be the same as the high-level specification and thus an implementation of a holistic representation would nevertheless be non-holistic. The most obvious example here would be implementing an HRR vector as a linked list of elements, but it would still be the case that the search for an element of the list would not be a search for a constituent of the object being represented. Of course one could convert the holistic representation to a classical symbol structure, by extracting the structure from the representation but that involves effectively abandoning the use of a holistic representation.

2.10 Conclusion: On exploiting holistic computation

2.10.1 Summary

This chapter has proposed a revised definition of holistic computation which focuses on the ability to process all the constituents of an object simultaneously. It has been argued that this definition offers greater clarity than previous definitions on the grounds that it does not confuse the processing of an object with the processing of its representation and, consequently, gives a clearer and more accurate picture of what is required to perform holistic symbol processing. It has further been argued that, under the revised definition, holistic computation is not unique to connectionism, that it is the holistic processing of symbol structures that is "new" and that Chalmers did demonstrate holistic symbol processing, contrary to Balogh's conclusions. It has demonstrated that holistic symbol processing can be achieved using a

holistic representation, or via parallel processing of all the elements of a fixed-length non-holistic representation. Both possibilities have been discovered by connectionists, and whilst there may be some scope for holistic *implementations* of classical symbol structures, they do not typically support holistic symbol processing. However, neither form of holistic symbol processing is unique to connectionism, although connectionist systems seem well suited to processing the representations that make them possible.

It has also been argued that to regard confluent inference as a class of holistic symbol processing is mistaken and confuses two different phenomena. Confluent inference seems to be a way of ensuring that the information required for certain tasks is directly accessible in a representation and may facilitate holistic symbol processing with that representation. However by itself confluent inference does not involve holistic symbol processing. It is nevertheless worthy of further study.

2.10.2 Exploiting Holistic Computation

In Chapter 1 it was argued that there is a need to evaluate holistic computation more critically and to investigate how easy it is to exploit holistic computation with neural networks. The work in this chapter contributes to both of these issues; the first by evaluating the concept of holistic computation as described in the literature and suggesting a new definition to achieve greater clarity than has hitherto been demonstrated, and the second by making it clearer just exactly what it is that we are trying to exploit.

There is still a long way to go before the questions of how easy it is and how best to exploit holistic computation in neural nets are answered. For example it is not clear from the literature just how far connectionist representations which are learned, such as the RAAM, can be pushed. The most reliable techniques for producing representations that support holistic symbol processing have thus far been analytically/mathematically derived techniques such as HRRs and the (S)RAAM, raising a question as to whether learned techniques can compete.

In order to investigate the questions of how easy it is to exploit holistic computation with neural networks and how best to go about it, the various techniques which support holistic computation will need to be evaluated in order to determine their limitations, what it is that allows them to support holistic symbol processing, what capacity there is for them to "scale up" to larger or more complicated tasks, etc. On this issue, the literature on holistic

computation and the techniques which support it is lacking in the following ways:

- With one exception, none of the literature explicitly sets out to evaluate the capacity of a technique for supporting holistic symbol processing except in so far as it demonstraes some task being performed holistically. That one exception is Niklasson and Bodèn's study of the RAAM (Niklasson and Bodèn, 1994). They explicitly set out to determine what sort of information was captured in RAAM encodings and therefore was available for holistic symbol processing. They found that information about the structure of a tree was available, whilst information about constituents was not always available in the encodings produced by the RAAM (this was often the case for deeply embedded information). This work is an example of the sort of study which needs to be done to evaluate the capacity of connectionist systems to exploit holistic computation, but it does not address questions about what a RAAM can and cannot learn, whether confluent inference might solve the problem of encoding constituent level information in the RAAM encodings of symbol structures or whether the RAAM's learning can be improved, for example.

- Most of the literature simply presents a representational technique and/or demonstrates its use for a particular task without, for example, trying to push the technique to its limits, determine what those limits are or analysing the performance of the technique for clues to its strengths and weaknesses.

- Even where some effort has been made to understand how a particular technique works, it has seldom gone beyond a cluster analysis of the representations. An exception to this, and probably the most thorough analysis of a technique, other than those derived mathematically, is that done on the Labeling RAAM which includes a formal analysis of its properties. This example may have yielded a lot of information relevant to evaluating the capacity of the LRAAM to support holistic symbol processing, although it was not the explicit aim of the analysis to do so. The other notable exceptions however have tended to concentrate solely on the nature of the representations developed, without looking at the limitations of the technique for learning to produce those representations. Balogh's work (Balogh, 1994), Blank *et*

al.'s work (Blank, Meeden and Marshall, 1992) and the work of Niklasson, Sharkey and Van Gelder (Niklasson and Sharkey, 1997; Niklasson and van Gelder, 1994) on systematicity fall into this category. This is of course useful information in evaluating the capacity for supporting holistic transformations but does not offer a complete picture.

The work in Chapters 3, 4 and 5 will make a contribution to the evaluation of the capacity of connectionist networks for holistic symbol processing, and to thus address some of the criticisms made of the literature above. The main purpose of this work is to evaluate the SRAAM's ability to generate representations that support holistic symbol processing via an attempt to recreate Chalmers' experiment with the SRAAM (thus providing direct comparison with the RAAM) and an evaluation of the performance of the SRAAM in doing so, both in terms of the representations produced and in terms of the nature of the solutions the SRAAM finds. As well as analysis of the representations developed, a wider analysis of the solutions produced, involving the weights and encoding/decoding trajectories will be presented, as well as consideration of what solutions should exist in principle. This evaluation of the SRAAM will form the rest of the contribution of this thesis towards the evaluation of holistic computation and the investigation of how best to exploit it, as argued for in Chapter 1.

Chapter 3

Syntactic Transformations with the Sequential RAAM

3.1 Purpose of this work

The motivation for the work in this chapter is to contribute to the investigation into how effectively connectionist systems can support holistic computation argued for in Chapter 1. The work presented here and in Chapters 4 and 5 contributes to this investigation by attempting to evaluate the ability of the Sequential RAAM to generate representations that support holistic symbol processing. This evaluation is made on the basis of an attempt to recreate Chalmers' experiment but with the RAAM replaced by the SRAAM. This chapter presents the results of this attempt to recreate Chalmers' experiment and Chapters 4 and 5 present an analysis of the SRAAM's performance in trying to learn to encode and decode the sentences used in Chalmers' experiment.

3.2 Why the SRAAM was chosen

Any of the representations discussed in Chapter 2 could have been chosen for this work as they all support holistic transformations to some degree. The SRAAM was chosen in preference to the other representations discussed in Chapter 2 for various reasons:

- The extent to which it can support holistic symbol processing is not clear from the literature. The more complicated tasks it has been used

for, such as Chrisman's (1991) translation of simple English sentences into Spanish, or Ho and Chan's (1996) parser, employ confluent inference rather than holistic symbol processing. The work of Blank et al. (1992) does involve holistic symbol processing but the tasks are simpler than those for which the RAAM has been used. Kwasny and Kalman (1995) train their SRAAM to encode and decode more complicated structures, but their attempts to perform holistic operations on those structures met with ambiguous results. Thus there is a real question as to whether the SRAAM is as capable of supporting holistic symbol processing as other representations.

- Kwasny and Kalman's work suggested that the SRAAM had various advantages over the RAAM which if combined with effective support for holistic symbol processing would suggest the SRAAM is a better vehicle for exploiting holistic computation than the RAAM. Their work suggested that the SRAAM is easier to train, offers greater flexibility in the range of structures that can be represented and offers higher levels of generalisation. The SRAAM is *easier to train* because the training does not require an external memory to store intermediate patterns and is in fact a simple modification of the Simple Recurrent Network (SRN) training. It is *more flexible* because arbitrarily branched trees can be represented as opposed to the fixed-branching required by the RAAM. The SRAAM achieved *greater levels of generalisation* on the tasks Kwasny and Kalman trained it with than Chalmers' RAAM achieved. Kwasny and Kalman reported 100% generalisation when their SRAAM was trained on 30 trees out of a total of 183 generated by a simple grammar. After training, the SRAAM could encode and decode all 183 trees without error. By comparison, Chalmers' RAAM, when trained on 80 trees (out of 250 possible) and tested on 80 novel trees, made errors when encoding and decoding 13 of the 80 novel trees. These results are not totally clear cut because Chalmers used back-propagation to train his RAAM whilst Kwasny and Kalman employed conjugate gradient training and they recomputed the error derivatives to take into account the feedback loops that exist in the SRAAM's training, hence the difference could be attributed to the training techniques rather than the network architectures used. However Kwasny and Kalman's results do suggest that when trained with Kwasny and Kalman's methods the SRAAM does have these advantages. If these

advantages could be combined with effective support for holistic symbol processing then the SRAAM would stand as a worthy alternative to the RAAM for holistic tasks.

- The SRAAM is closer to standard connectionist models than either the (S)RAAM (which boasts very quick training times) or HRRs, both of which appear to support holistic symbol processing at least as effectively as the RAAM. Neither (S)RAAM or HRRs involve error minimisation, nor do they utilise networks of units with weighted connections between them, as most standard connectionist models do. The SRAAM by contrast can be trained using standard techniques, such as back-propagation or conjugate gradient, and explicitly employs a network of units with weighted connections.

- The SRAAM representations employed in Kwasny and Kalman's work were smaller than with HRRs, and in one case smaller even than the representations developed by RAAM in Chalmers' experiment, suggesting that they should be reasonably practical to use.

- Finally, the LRAAM's operation is rather different to that of the other methods due to the practice of turning it into a bi-directional associative memory and the production of reduced descriptions of structures to be used in holistic transformations is not as natural with the LRAAM as it is with other methods. Furthermore there has already been extensive investigation into its properties (Sperduti, 1993a; Sperduti, 1993b), which include a formal analysis of it. Thus it is already well understood.

3.3 Recreating Chalmers' experiment

3.3.1 Chalmers' experiment

Chalmers' experiment involved a simple task, the active to passive transformations of simple sentences, which the RAAM performed fairly well. Chalmers' RAAM learned to encode and decode all the training sequences and the bulk of the testing sequences, and the transformation network was able to learn the task well. Indeed the errors that arose during the transformation task were shown to be due to errors in encoding and decoding,

rather than in the transformation. Thus it was clear that the RAAM was creating representations that supported holistic symbol processing fairly well. For these reasons, it seems instructive to try to recreate the experiment, using the SRAAM in place of the RAAM. The original experiment thus forms a benchmark against which to compare the SRAAM's capacity to generate representations that support holistic transformations. By representing the sentences as simple sequences, rather than trees, the SRAAM can be trained to encode and decode the sentences and then another network can be trained to perform the transformation on the resulting representations. It was also decided that both back-propagation and Kwasny and Kalman's training methods[1] would be used for training the SRAAM to provide a comparison between the two and determine whether the SRAAM architecture or Kwasny and Kalman's training methods were responsible for the SRAAM's apparent superiority to RAAM.

3.3.2 Experimental procedure

In the experiments reported below, all the sequences are of the form "john love helen" or "helen is love by john", as in Chalmers' experiment. There are five possible nouns ("john", "helen", "chris", "diane" and "michael"), five possible verbs ("love", "kill", "chase", "hug" and "hit") and the words "is" and "by", yielding 125 possible sentences of the form "john love helen" and another 125 possible passive equivalents. 130 sentences (65 active/passive pairs) were used in the training set and the remaining 120 sentences (60 pairs) were used as the testing set. Appendix E lists the training and testing sentences.

In Chalmers' experiment the representation of each symbol involved 13 units, with 2 units set to the value of 1 and the rest to 0 and this scheme is used for various training runs reported here. The scheme is illustrated in Table 3.1 and will subsequently be referred to as the "2 unit representation". Some of the networks trained here, including those trained by Kwasny and Kalman's simulator, use the hyperbolic tangent activation function outputting values between -1 and 1. For these networks, the "0"s in the symbol patterns are replaced with "-1"s. In addition to the training runs involving the 2 unit representation, various runs were done with patterns where only 1 unit out of the 13 is set to 1, to see if this simpler scheme would help training

[1]Kwasny and Kalman generously provided their simulator for use with this work.

john	0 1 0 0 0 0 1 0 0 0 0 0 0
michael	0 1 0 0 0 0 0 1 0 0 0 0 0
helen	0 1 0 0 0 0 0 0 1 0 0 0 0
diane	0 1 0 0 0 0 0 0 0 1 0 0 0
chris	0 1 0 0 0 0 0 0 0 0 1 0 0
love	0 0 1 0 0 0 1 0 0 0 0 0 0
hit	0 0 1 0 0 0 0 1 0 0 0 0 0
betray	0 0 1 0 0 0 0 0 1 0 0 0 0
kill	0 0 1 0 0 0 0 0 0 1 0 0 0
hug	0 0 1 0 0 0 0 0 0 0 1 0 0
is	0 0 0 0 1 0 1 0 0 0 0 0 0
by	0 0 0 0 0 1 1 0 0 0 0 0 0
nil	0 0 0 0 0 0 0 0 0 0 0 0 0

Table 3.1: The 2 unit representation of the symbols used in Chalmers' data. For hyperbolic tangent networks, replace "0"s with "-1"s.

— the patterns are all orthogonal which may reduce interference. Table 3.2 illustrates this representation. As before, "0"s are replaced with "-1"s for the networks employing the hyperbolic tangent function. This representation will subsequently be referred to as the "1 unit representations".

All the training runs for back-propagation indicate the best performance achieved after trying various combinations of values for the momentum and learning in the range 0.01 to 0.9, and training for upto 10000 iterations. With Kwasny and Kalman's simulator the step size is determined adaptively so these parameters do not exist. The simulator attempts to train until the outputs are within in a small tolerance of the target outputs (a tolerance of 0.01 was used here), or until convergence is achieved otherwise.

3.4 Results

3.4.1 Learning the main data set

Table 3.3 shows the results of various attempts to train an SRAAM to encode and decode 130 sequences drawn at random from the set of sequences derived from Chalmers' data, using the 2 unit representations. Likewise Table 3.4 shows the equivalent training runs involving the 1 unit representation. In

john	1 0 0 0 0 0 0 0 0 0 0 0 0 0
michael	0 1 0 0 0 0 0 0 0 0 0 0 0 0
helen	0 0 1 0 0 0 0 0 0 0 0 0 0 0
diane	0 0 0 1 0 0 0 0 0 0 0 0 0 0
chris	0 0 0 0 1 0 0 0 0 0 0 0 0 0
love	0 0 0 0 0 1 0 0 0 0 0 0 0 0
hit	0 0 0 0 0 0 1 0 0 0 0 0 0 0
betray	0 0 0 0 0 0 0 1 0 0 0 0 0 0
kill	0 0 0 0 0 0 0 0 1 0 0 0 0 0
hug	0 0 0 0 0 0 0 0 0 1 0 0 0 0
is	0 0 0 0 0 0 0 0 0 0 1 0 0 0
by	0 0 0 0 0 0 0 0 0 0 0 1 0 0
nil	0 0 0 0 0 0 0 0 0 0 0 0 0 0

Table 3.2: The 1 unit representation of the symbols used in Chalmers' data. For hyperbolic tangent networks, replace "0"s with "-1"s.

both cases, the networks were also tested on the 120 remaining sequences from Chalmers' data. The column headings for both tables are described below:

- The "Method" column indicates what training method was used. The "BP SSE" training method refers to standard error back-propagation, with units with sigmoid activation functions giving outputs in the range 0 to 1, and using the sum of squares error (SSE) function. The "BP KKE" method employs error back-propagation with the hyperbolic tangent activation function giving the range -1 to 1 and the Kwasny and Kalman's error (KKE) function. Kwasny and Kalman claimed these modifications would lead to faster convergence (Kalman and Kwasny, 1994). The "KK" training method refers to Kwasny and Kalman's training method, based on conjugate gradient training and with the error derivatives recomputed to take into account feedback effects. This method also employs hyperbolic tangent activations and the KKE function.

- The "Network" column indicates the network architecture used, giving the number of inputs, the size of the hidden layer and the number of outputs for each case.

Method	Network	Iterations	Error	% train	E/seq Tr
BP SSE	25-12-25	1350	98.87	0.8%	1.80
BP SSE	52-39-52	1520	71.28	42.3%	0.96
BP KKE	25-12-25	570	492.80	3.8%	2.65
BP KKE	52-39-52	320	243.71	47.7%	0.59
KK	25-12-25	DE:8310 FE:80587	1.23	46.9%	0.53
KK	52-39-52	DE:6950 FE:71384	0.04	85.3%	0.15

Method	Network	% test	E/seq Te
BP SSE	25-12-25	1.7%	1.83
BP SSE	52-39-52	21.7%	1.19
BP KKE	25-12-25	0.0%	2.85
BP KKE	52-39-52	50.8%	0.64
KK	25-12-25	45%	0.55
KK	52-39-52	87.5%	0.12

Table 3.3: Various attempts to learn to encode and decode 130 sequences from Chalmers' data set, using the 2 unit representation.

- The "Iterations" column indicates the number of epochs used in training for back-propagation, and the number of derivative (DE) epochs and functional epochs (FE) for Kwasny and Kalman's training method.
- The "Error" column indicates the minimum network error reached during training.
- The "% train" column indicates the percentage of the training sequences correctly encoded and decoded once the network is trained.
- The "% test" column indicates the percentage of the testing sequences correctly encoded and decoded.
- The "E/seq Tr" column gives the number of encoding and decoding errors per sequence in the training set, i.e. the number of incorrect symbols when a sentence is encoded and then decoded — if the decoded sequence is of incorrect length, this counts as one error.
- The "E/seq Te" column gives the number of encoding and decoding errors per sequence in the testing set.

Method	Network	Iterations	Error	% train	E/seq Tr
BP SSE	25-12-25	2400	123.05	3.1%	1.81
BP SSE	52-39-52	1820	131.38	32.3%	0.92
BP KKE	25-12-25	120	535.85	0.0%	2.37
BP KKE	52-39-52	270	362.83	23.8%	0.90
KK	25-12-25	DE:7679 FE:92470	0.81	31.5%	0.85
KK	52-39-52	DE:7153 FE:71545	0.06	50.0%	0.50

Method	Network	% test	E/seq Te
BP SSE	25-12-25	0.0%	2.09
BP SSE	52-39-52	17.5%	1.22
BP KKE	25-12-25	0.0%	2.40
BP KKE	52-39-52	15.8%	1.16
KK	25-12-25	20.8%	0.88
KK	52-39-52	50.0%	0.50

Table 3.4: Various attempts to learn to encode and decode 130 sequences from Chalmers' data set, using the 1 unit representation.

As can be seen in both tables, it was not possible to obtain perfect encoding and decoding of the training set with hidden layer sizes of up to 39 units with either back-propagation or Kwasny and Kalman's training. Various points can be noted about this data, however:

- The error levels in the testing set were generally higher than in the training set. However, for networks trained with Kwasny and Kalman's method, the error levels in the testing set were closer to the error levels in the training set than was the case for networks trained with back-propagation and in one case the error in the testing set was lower than in the training set for Kwasny and Kalman's method. This is suggestive of a greater power of generalisation in Kwasny and Kalman's training than in back-propagation.

- Kwasny and Kalman's training consistently outperforms back-propagation, obtaining lower network error, a higher percentage of correctly encoded/decoded sentences and a lower number of encoding and decoding errors per sequence.

- The use of the hyperbolic tangent activation function and Kwasny and Kalman's error function improved the performance of back-propagation

somewhat with the 2 unit representation of symbols, but was worse with the 1 unit representation. Convergence was faster than with standard back-propagation, as Kwasny and Kalman suggested.

- The 2 unit representation generally lead to better performance than the 1 unit representation, especially with the 39 unit networks, contrary to the expectations of the author. It appears that the fact that in the 2 unit representation some units code for whether the symbol is a noun/verb or "by" or "is" may have helped the network to learn which sort of symbol to produce at any given point in time. To test this hypothesis, SRAAMs with 39 hidden units and employing sigmoidal and hyperbolic tangent activations[2] were trained by back-propagation using a randomized 2 unit representation, i.e. where 2 of the 13 units are on but these are randomly chosen for each pattern and thus do not indicate what type of symbol the pattern represents. For sigmoidal units, the lowest error of 214.61 was reached after 773 iterations. The average error per sequence on the training set was 1.18 and 20% of the training sequences were encoded and decoded without error. On the testing set the average error per sequence was 1.44 and 12.5% of the training sequences were encoded and decoded correctly. For the SRAAM employing hyperbolic tangent units, the minimum error of 888.52 was reached after 91 iterations. The average error per sequence was 1.57 and 7.69% of the training sequences were encoded and decoded correctly. Thus the errors for this randomized representation were higher than for either of the representations used above, confirming that the use of the units to indicate which sort of symbol a pattern represents improved the training.

A direct comparison of the training times between back-propagation and Kwasny and Kalman's method is not valid as the iterations mean different things in each method. Some iterations of Kwasny and Kalman's method involve computing derivatives as in back-propagation, whilst others employ a heuristic that approximates using the derivatives. The network error reported is not the calculated uniformly either. Kwasny and Kalman's training and the BP KKE method employ the Kalman-Kwasny error function (Kwasny and Kalman, 1995):

[2]There was not enough time to train the Kwasny and Kalman networks with this representation.

$$E = \sum_p \sum_i \frac{(t_{pi} - a_{pi})^2}{1 - a_{pi}^2}$$

where p runs over all the patterns; i runs over the output units; t_{pi} is the target output for pattern p, unit i; and a_{pi} is the actual output for that pattern and unit. This leads to higher error values than the normal sum of squares measure used in standard back-propagation for the same data. This means the lower errors reported by Kwasny and Kalman's training method are of greater significance.

Attempts to extend back-propagation's training beyond the point of lowest error, resulted in the error increasing gradually and then oscillating around the 400-800 level with standard back-propagation and increasing dramatically and then oscillating chaotically around a very high value (typically 1000-2000) with back-propagation modified with KKE and hyperbolic tangent activation. With Kwasny and Kalman's training method, attempts to extend the training beyond the point at which it stops in the above tables result in the simulator terminating training almost immediately, suggesting it has found the lowest error it is capable of finding, without being restarted from scratch at least.

3.4.2 Transformations on imperfectly learned data

Several training runs came close to learning the sequences perfectly, including all those employing Kwasny and Kalman's method (except the 1 unit representation with the 12 unit hidden layer combination) and the BP KKE method with 39 unit hidden layer and the 2 unit representation. This suggests that in these training runs, most of the information needed to represent these sequences was being captured successfully. If so it might be possible to perform the active to passive transformations without inducing any extra errors in the encoding/decoding process. Were this to be the case, one could conclude that the SRAAM encodings can capture useful information that can be used for performing structural transformations, and given enough resources would thus support holistic symbol processing.

To test this hypothesis, a standard 3 layer feed-forward net was trained, using back-propagation, to map between the active and passive representation of half the pairs of training sequences learned in these training runs. The trained network (hitherto known as the transformation network) was then

tested on the entire set of sequences, i.e. the active sequences were encoded and fed into the transformation network and the resulting encoding was decoded and checked against the relevant passive sentence. Back-propagation was used for training the transformation network in order to provide direct comparison with Chalmers' experiment, where the transformation network was also trained with back-propagation. Any difficulties in learning the task could then be directly traced to the nature of the representation learned by the SRAAM (with whichever training method), rather than to differences in the training method employed to learn the transformation. All the training runs involved stopping the network once the sum squared error fell below 0.04.

The results of this experiment are given in Table 3.5. The "SRAAM" column indicates which of the SRAAMs trained above was used to produce the encodings for the sequences which the transformation network operated on. BPSSE39Un2 refers to the SRAAM with 39 hidden units trained by standard back-propagation, on the 2 unit representations. BBKKE39Un2 refers to the SRAAM with 39 hidden units trained by back-propagation using Kwasny and Kalman's error function and hyperbolic tangent activation on the 2 unit representations. KK12Un2 refers to the SRAAMs with 12 hidden units trained using Kwasny and Kalman's method on the 2 unit representations. KK39Un1 and KK39Un2 refer to the SRAAM with 39 hidden units trained using Kwasny and Kalman's method using 1 and 2 unit representations respectively. The "Network" column indicates the architecture of the transformation network, e.g. the first one has 39 inputs and outputs and 19 units in the hidden layer. "Epochs" gives the number of iterations it took to train the transformation net. The "Induced errs TR" column gives the number of extra incorrect symbols on average after decoding, introduced by the active-passive transformation when using the sequences that the transformation network was trained on. The "Induced errs TE" column gives the same figure for the sequences the transformation network was not trained on. (Recall that the training and testing sets are formed by splitting the 130 sequences the SRAAMs were trained on in half.) From this table the following can be noted:

- With the exception of KK39Un2 and KK12Un2, all the transformation networks induced an average of only 0.03 or 0.06 errors per transformation on the training set.
- KK39Un1 achieves the best result on the testing set.

SRAAM	Network	Epochs	Induced errs TR	Induced errs TE
BPSSE39Un2	39-19-39	1740	0.03	0.44
BPKKE39Un2	39-19-39	2240	0.06	0.53
KK12Un2	12-6-12	4227	0.36	0.50
KK39Un1	39-19-39	162	0.06	0.13
KK39Un2	39-19-39	219	0.85	1.31

Table 3.5: Results of training & testing transformation network on complete set of sequences

- The networks trained on the 39 unit SRAAMs produced by back-propagation introduced 0.44 or 0.53 errors per transformation on the testing set.
- Networks KK39Un2 and KK12Un2 did not perform as well as the rest on either the training or testing sets. In the case of KK12Un2 this is perhaps understandable due to the smaller hidden layer size. In the case of KK39Un2 it is a surprising result. KK39Un2 had achieved the best encoding/decoding performance of all, yet the errors introduced are greatest here.

From these results it seems that the 39 unit SRAAM encodings produced by KK39Un1 managed to capture most of the information needed to perform the active/passive transformation on them successfully. To test this conclusion further, KK39Un1 was trained until the sum squared error was below 0.01. This took 756 epochs (including the 126 to get below the SSE of 0.04). The trained network resulting from this was able to perform the transformation without inducing any extra errors on either the training sequences or the testing sequences. This confirms that if a SRAAM could be trained to perform the encoding and decoding perfectly, then a transformation network could be trained to operate on those encodings without inducing error. All that would be needed to obtain such performance would be to increase the size of the hidden layer until it is large enough for either back-propagation or Kwasny and Kalman's training method to find an error-free solution (the declining error with increased size of hidden-layer confirms this). In other words, given enough resources, the SRAAM can generate representations that support holistic symbol processing.

The performance of the networks operating on SRAAM encodings produced by back-propagation is not as good as KK39Un1. Although their per-

formance on transforming the training set is on a par with that of KK39Un1, this performance does not generalise well to the testing set. Indeed, when BBKKE39Un2 was trained until the SSE was less than 0.01 (requiring 9142 iterations in total), whilst no errors were introduced in the testing set, an average of 0.47 errors per transformation were introduced on the training set: not much improvement on the same network trained to a SSE of just under 0.04. Likewise when BBSSE39Un2 was trained until the SSE was under 0.01, requiring 100083 iterations in total, it added no errors on the training set but induced 0.44 errors per transformation on the training set. Hence it seems the SRAAM encodings produced by back-propagation do not support holistic symbol processing to the same degree as those produced for KK39Un1.

KK12Un2's performance is worse than BP39SSEUn2, BPKKE39Un2 or KK39Un1. This is not surprising given that the same amount of information is being stored in 12 units as in 39 units, which will lead to lower tolerance of noise.

This leaves KK39Un2. Here the errors are much higher than with any of the other networks, which is puzzling. One can only deduce that the encodings were much less robust to the noise introduced by the transformation network than in the other cases. Since this set of encodings was produced from the best performing SRAAM this is rather odd. Perhaps in learning to encode and decode a higher proportion of the sequences than the other networks, KK39Un2 had to create a fragile representation for some reason. As with KK39Un1, an attempt to train the transformation network until the SSE was under 0.01 was made to see if the performance could be improved to the point where no errors were introduced. This took 636 iterations, resulting in 0.51 errors per sequence on the training set and 0.75 errors per sequence on the testing set, again confirming that these representations are less tolerant to noise than the others.

3.4.3 Discussion

The foregoing sections have yielded the following findings:

- In order to learn to encode and decode 130 of Chalmers' sentences without error, the SRAAM requires more than 39 hidden units, whether one is using back-propagation or Kwasny and Kalman's conjugate gradient based training method.

- Kwasny and Kalman's method consistently produces better encoding/decoding results than does standard back-propagation, hence their claimed improvements over the RAAM could have been due to their training method rather than the use of the SRAAM *per se*.
- Chalmers' original 2 unit representation for the symbols in each sequence yields better encoding/decoding results than the 1 unit representation which one might have thought to be simpler to use as each symbol is orthogonal to the others.
- One can train a 3-layer feedforward network to perform the active to passive transformation on the encodings produced by the SRAAM without any errors being introduced on top of those produced by the encoding and decoding process itself, suggesting that the encodings would support holistic symbol processing if given enough resources to achieve perfect encoding and decoding.

These findings suggest that:

- The SRAAM is more difficult to train than the RAAM, calling into question the claims made for the superiority of the SRAAM by Kwasny and Kalman (1995).
- Although holistic symbol processing may be supported when the SRAAM is given enough resources to encode and decode a set of sequences perfectly, the representations developed by the SRAAM will be much larger than those developed by a RAAM for the same task.
- Much of the improved performance Kwasny and Kalman claimed for the SRAAM may have actually been due to their training methods, and they might find that the RAAM performs even better still when trained with their methods rather than back-propagation.

For these reasons it appears that the SRAAM, when trained with back-propagation or Kwasny and Kalman's methods, may not be as effective a vehicle for exploiting holistic computation as other techniques, such as the RAAM or the (S)RAAM, because it is more difficult to train and requires larger representations. However whether this is due to inherent properties of the SRAAM's architecture and operation, or whether some modification to the training methods could improve the performance of the SRAAM is yet to be seen.

3.5 Comparison with earlier work

The SRAAM has been used successfully in other work, such as the work of Blank et al. (1992), Ho and Chan (1996) and Chrisman (1991).

Blank *et al.*'s experiments include an experiment comparable to the work presented here in complexity, except that all the sentences are the same length, being 3 symbols long. Thus their experiment is simpler in this respect, but comparable in other respects, to the experiment presented here. They trained a SRAAM, using standard back-propagation, to encode and decode 100 sentences, of the form "X verb Y". There were 27 total possible symbols in these sentences, which were represented by 27 units, of which one was set to "1" and the rest to "0" depending on which word was being represented. The SRAAM's hidden layer was 30 units. After 21000 iterations of training, the SRAAM could encode and decode all 100 sentences perfectly and also was able to encode and decode 80 sentences perfectly out of 100 novel sentences presented to it. This is in contrast to the inability to learn the sequences exhibited in the work presented here. There are three possibilities to consider in explaining this. Firstly, and this is most probably part of the reason for the difference, is the fact that since the sentences used by Blank *at al.* are all 3 symbols long, there is less embedding and fewer hidden-layer states are produced per sequence than in the experiments reported here. Here on average the sequences are 4 symbols long and there are on average 4 levels of embedding. Half the sequences are 5 symbols in length, half 3. Secondly, Blank *et al.*'s networks were trained for far more iterations than used here (21000 compared to 1000 to 4000 or so here). However in order to see whether such long training runs might make the difference, a training run of 30000 iterations was performed. It failed to improve on the results presented earlier, and displayed the chaotic behaviour described above after reaching its minimum error. Also it should be noted that with Kwasny and Kalman's method all 65 of the active sequences were encoded and decoded without error, though this was not the case for the back-propagation runs. Thirdly, perhaps Blank *et al.* used different values for the momentum and learning rate. However they do not mention what values they used, and the results of the experiments here are the best achieved with several sets of values ranging from 0.01 to 0.9, which covers the values normally used in the literature. It appears that the most likely explanation is therefore that the extra complexity of the experiments here is the most probable reason for the difference in performance.

Both Chrisman's and Ho and Chan's work involve more complicated tasks. Chrisman translated a set of English sentences into Spanish, and Ho and Chan performed syntactic parsing using confluent inference. At first sight it might seem that the results presented here are also in conflict with these examples, however in Ho and Chan's work the representations are much larger than those used here, being 210-220 units in length. Also both Chrisman and Ho and Chan employ confluent inference to create their representations which could explain part of the difference in performance, and neither's work employs holistic transformations, in that neither actually involves an explicit transformation being done. It is therefore unclear to what extent holistic symbol processing are supported in either case. Thus these results could be quite compatible with Chrisman's and Ho and Chan's work, because that work involves either larger representations or the use of confluent inference, all of which would affect the success or otherwise of what is being attempted.

It is puzzling as to why Kwasny and Kalman obtained better results than those presented here even though they used smaller representations for equally or more complicated tasks. They obtained very good encoding and decoding performance (in contrast to the results here), including 100% generalisation, but had mixed results for holistic symbol processing. For example, their first experiment involved learning a set of 30 sequences involving a total of 930 individual symbols. Whilst the number of sequences was smaller than here, the length was much greater (31 on average, compared to 4 here) and the total number of symbols was greater (930 compared to 520 here). It is especially puzzling that worse performance was obtained for apparently simpler tasks, in the light of the fact that their code was used for some of the training runs here. It may be that the peculiar properties of the tasks involved are behind this, but it is not clear why this would be so.

3.6 Conclusion: The SRAAM's limitations

The SRAAM's performance on learning the sentences from Chalmers' experiment has been shown in this chapter to be inferior to the performance of the RAAM on the same task. This suggests that the SRAAM may be more difficult to train and require larger representations than the RAAM, for the same tasks. Furthermore, it can be noted that other techniques have been used for more complicated tasks than the encoding of simple active and passive

sentences used here (see the techniques discussed in Chapter 2 for examples). On this basis it would be tempting to conclude that the SRAAM is not as good a vehicle for holistic computation as other learned methods, such as the RAAM, and leave it at that. However to do so would leave the question of why the SRAAM performed poorly on this task unanswered. The work presented in this chapter does not address this question. It is an important question for several reasons:

- An understanding of why the SRAAM failed may yield insights on how to improve the SRAAM and overcome the problems exhibited here. It may be for example that some simple modification to the training methods could allow the SRAAM to learn to encode and decode the sequences with a small hidden layer, thus making it more effective as a vehicle for exploiting holistic computation.

- The RAAM, SRAAM and other derivatives of the RAAM are all based on the use of hidden layer activation patterns as inputs and target outputs to create compositional representations. For this reason an understanding of why the SRAAM failed and what the limitations of the SRAAM are may indicate how to improve the performance of RAAM-style techniques generally and may also indicate what limitations exist for RAAM-style techniques.

- More generally, an understanding of the limitations of the SRAAM may offer lessons on how to create connectionist representations capable of supporting holistic symbol processing.

For these reasons, Chapters 4 and 5 present an analysis of the SRAAM aimed at understanding why it failed and indicating what limitations of the SRAAM lay behind this failure.

Chapter 4

Analysis of the Sequential RAAM Part 1: Analysis of the weights

4.1 Purpose

In Chapter 3, an attempt was made to recreate Chalmers' experiment using the SRAAM in place of the RAAM. It was found that whether using back-propagation or Kwasny and Kalman's training method, the SRAAM could not learn to encode and decode a training set of 130 sequences perfectly with a hidden layer size of up to 39 units, 3 times larger than for the RAAM used in Chalmers' experiment. Further experimentation demonstrated that holistic transformations of encodings for the imperfectly learned sequences were possible without inducing extra errors on top of those in the encoding and decoding processes, suggesting that given enough resources a SRAAM could generate representations that would support holistic symbol processing. The purpose of the work presented in this chapter and Chapter 5 is to try and find out why the SRAAM failed to learn this task. This is important in order to determine whether the failure is due to inherent problems with the SRAAM or whether some modification to the SRAAM training might allow it to learn the task with smaller hidden layers. This chapter will address the questions of how the solutions relate to the weight-space, e.g. whether the solutions exist, if they do exist whether they are easy to find and whether the partial solutions actually found by the SRAAM are sensitive to weight

changes. Chapter 5 will address the questions of how the hidden-layer states produced in encoding and decoding are organised in hyperspace.

4.2 Possible causes of failure

When a neural network fails to learn a task there are many possible reasons why this might be the case. These can summarised as follows:

- The solution does not exist.
- The solution is very difficult to find.
- There is a problem with the training method.
- The right set of parameters has not been used with the training method in question.

One of the aims of the work in this chapter and Chapter 5 is to indicate which of these reasons is the correct one. The first and the last are worth eliminating before the others are considered. The rest of this section deals with these two possibilities.

4.2.1 Does the solution exist?

More precisely the question is whether the solution exists for the sizes of network which were used in the last chapter, namely SRAAMs with hidden layers of 12 to 39 units. There are some reasons for thinking it might not exist, as follows:

- The limitations of 2 layer networks. When you split the SRAAM into the encoder and decoder these are both 2-layer networks and thus suffer from the well-known limitations of 2-layer networks[1]. In a 2-layer network, the outputs are derived by multiplying the inputs by the weight matrix and then applying the activation function. For each individual output unit, if the inputs are not linearly separable, the unit will not be able to discriminate between them. If this were the problem however, it would also affect the RAAM, since precisely the same situation

[1]These limitations are explored by Minsky and Papert (1969) and in Rumelhart and McClelland's volumes(Rumelhart and McClelland, 1986).

holds for the RAAM's encoder and decoder. Also there is more flexibility than with normal 2-layer networks since both the inputs to the encoder and decoder and the outputs from them are not fixed – only part of them is. Hence the network can alter the inputs and outputs as well as the weights in order to find the solution.

- The number of hidden layer states generated during encoding and decoding is too large for the network to discriminate. As each sequence is encoded, a set of hidden layer patterns is produced, 1 for each symbol in the sequence. It may be that this number is simply too high for the network to discriminate between them all. For the training set of 130 sequences used in Chapter 3 a total of 520 hidden layer patterns is generated – 4 on average per sequence. However, suppose the hidden layers were restricted to binary patterns, then even the smallest hidden layer used in Chapter 3 can distinguish more than 520 patterns. This was 12 units in size, leading to 4096 possible binary patterns that can be generated in those units. It does provide an explanation for the greater difficulty of training the networks compared to the RAAM however. The RAAM in Chalmers' experiment only has 2 hidden layer states on average per sequence due to the different method of encoding, thus only 260 states would be produced for the set of sequences used in Chapter 3. Thus the SRAAM needs a greater power of discrimination than the RAAM for the same number of sequences.

- The amount of embedding of information is too great. As more and more symbols are added to a sequence, the first symbol to be added gets more and more deeply embedded. In the sequences here, there are either 3 or 5 levels of embedding depending on whether the sequence is an active or a passive sentence. It is worth noting that with Chalmers' RAAM, the maximum amount of embedding was 2 levels (though more symbols would be embedded at a particular level), again another reason why the SRAAM could be more difficult to train.

Some reasons for thinking the solution does not exist are discussed above. They are reasons for thinking the training of the SRAAM might be more difficult than that of the RAAM. However to get any further with the question of whether these factors are behind the failure of the training here requires performing analyses of the SRAAMs in order to find out how they are developing the representations of the sequences they are learning to encode and

decode. For the moment however a proof that a solution does exist for the task of encoding and decoding the sequences used here using any SRAAM with a hidden layer of 20 or more units will be presented. This proof says nothing about whether some other form of solution might exist for smaller SRAAMs.

4.2.2 Proof the solution exists for 20 or more hidden units

The "shifting" architecture

Figure 4.1 gives the general form for the weights of a working SRAAM. The context layer, hidden-layer and the distributed part of the output layer are each split into five banks. **I** indicates the identity mapping, thus the mapping from the context to the hidden-layer copies the contents of banks 1 to 4 in the context layer into banks 2 to 5 in the hidden-layer. Likewise the mapping from the hidden-layer to the distributed output layer copies the contents of banks 2 to 5 in the hidden-layer back into banks 1 to 4 in the distributed output. **T** is the mapping of the symbol to its encoding in bank 1. **S** is the reverse mapping.

Putting the details of how the necessary mappings are achieved to one side for the moment, the operation of the encoder and decoder halves of the SRAAM can be considered:

- The encoder. The context layer is set to the value for the "empty" sequence, e.g. all zeros. Then the first symbol is presented to the input layer. The activation is propagated forward, mapping the symbol into bank 1 of the hidden-layer, and banks 1 to 4 of the context layer to banks 2 to 5 of the hidden-layer. Then the hidden-layer is copied into the context layer and the next symbol presented to the input layer. This process repeats itself until all 5 banks in the hidden layer are full (i.e. the encoder can encode sequences of up to 5 symbols).

- The decoder. This starts with the final hidden-layer pattern produced from the encoder being put into the hidden-layer. The activation is then fed forward, producing the last symbol to be encoded on the symbolic output and the values of banks 2 to 5 of the context layer are copied to banks 1 to 4 of the distributed output layer. Bank 5 of the distributed output layer is set to the "empty" value, e.g. all zeros.

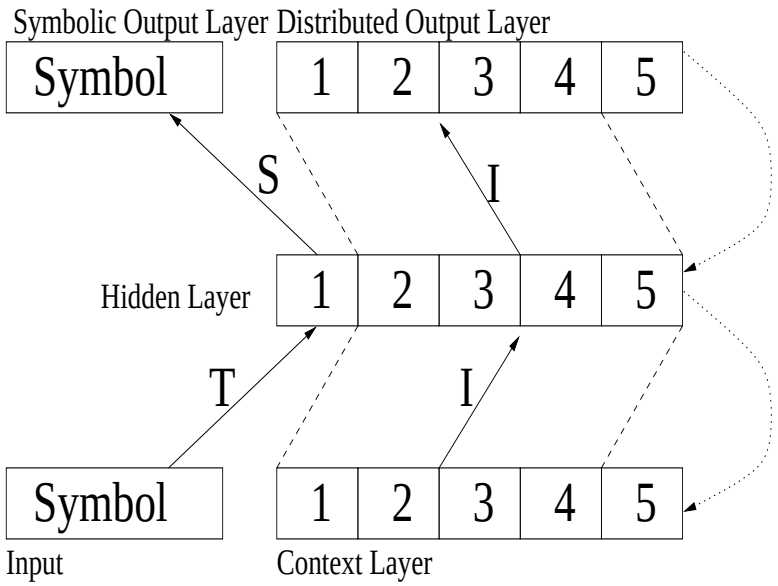

Figure 4.1: A possible solution for implementing a working Sequential RAAM.

> Then the distributed output layer is copied back into the hidden-layer. This process is repeated until all the banks of the distributed output layer are set to zero. Again this can decode sequences of length 5 or less.

One can extend this architecture with more banks in the context hidden and distributed output layers to create encoders and decoders for longer sequences.

The solution employing linear units

Here it is shown how the above solution can be implemented using units employing the linear activation function. The solutions for units employing the sigmoidal activation function and the hyperbolic tangent activation functions are based on this. The simplest way of implementing the above architecture with linear activation is simply to set the weight matrix for **I**, **S** and **T** to the relevant identity matrices, and set all other weights in the network to zero. Thus the mapping from the input and context layer to the hidden-layer simply copies the symbol to bank 1 of the hidden-layer and banks 1 to 4 of the

context layer to banks 2 to 5 of the hidden layer. Likewise the mapping from the hidden-layer to the symbolic and distributed output layers copies bank 1 to the symbolic output and banks 2 to 5 of the hidden-layer to banks 1 to 4 of the distributed output layer. Because all other connections are zero, bank 5 of the output layer receives no input and is also thus zero (since we are using linear units).

Given that the symbols are represented by x units, then the hidden layer of this SRAAM will consist of $5x$ units. Since the solution can be extended to larger and larger hidden-layers by adding more and more banks, we can get a general solution; for a set of sequences of maximum length L, with the symbols represented in x units, we have a solution involving a hidden-layer of Lx units. The distributed output and context layers are of course the same size. If there are no restrictions on how to represent the symbols then only 1 unit is needed to represent them and the integers 1 to M can be used to code for each symbol, where M is the number of possible symbols that can appear in the sequences. Thus a general solution exists for linear units that employ L hidden units, where L is the length of the longest sequence.

The sigmoid activation solution with binary symbols

Now the above solution is adapted for units employing the sigmoid activation function. Recall that we have a set of sequences of maximum length L built up out of any of M symbols. As can be seen from Figure 4.2, the sigmoid function squashes its input into the range $(0, 1)$. Because of this, the identity mapping can only be approximated. Because hidden-layer patterns are recycled in both the encoder and decoder, the error in this approximation will build up over the encoding and decoding of a sequence, and may well destroy the information within it. If however the identity mapping can be approximated to an arbitrary degree of closeness, then the error can be kept low enough to allow encoding and decoding to occur. If the hidden-layer patterns that are being passed through the identity mappings are restricted to vectors consisting of either "0"s or "1"s, this can be achieved. The weights can be constituted by taking the identity matrix and making some modifications. Firstly the matrix is multiplied by some value β. Secondly the bias weights are set to $-\frac{\beta}{2}$. Thus if the value of the input unit copied is "0", the relevant output unit will compute:

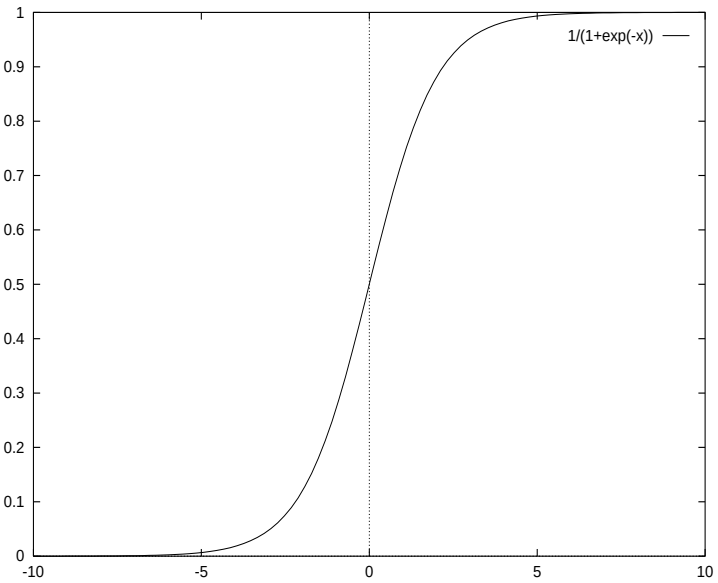

Figure 4.2: The sigmoid function

$$\lim_{\beta \to \infty} \text{sigmoid}(-\frac{\beta}{2}) = 0$$

If the value of the input unit being copied is "1" then the relevant output unit will compute:

$$\lim_{\beta \to \infty} \text{sigmoid}(\frac{\beta}{2}) = 1$$

By setting β to a suitably large value the "0"s and "1"s can be copied to an arbitrary degree of accuracy.

Now consider the patterns used to represent the symbols. The above considerations allow the use of patterns consisting solely of "1"s and "0"s since the identity mapping for these can be approximated to arbitrary precision. Thus for M symbols, the number of units in each bank is $\lceil \lg(M) \rceil$[2] units, and the hidden-layer is thus $L \times \lceil \lg(M) \rceil$ units in size. This is all of course assuming that the symbols are simply copied into and out of bank 1 of the hidden layer as with the linear solution presented above. Thus we have a general solution; for M symbols, the sigmoid activation function and

[2]For the purposes of this proof, $\lg(M) = \log_2(M)$.

sequences of length L, one can construct an SRAAM with $L \times \lceil \lg(M) \rceil$ units in the hidden, context and distributed output layers that will encode and decode those sequences without error, by using binary patterns to represent the symbols. This is assuming that the degree of precision required can be matched by the machine. (In practice this is unlikely to be a problem.) With Chalmers' sequences there are 12 possible symbols (excluding "nil") and a maximum length of 5. 4 units is the minimum necessary to represent all the symbols with binary patterns, thus the hidden-layer for this solution is 20 units in size, well within the range used in Chapter 3.

The above solution assumes we can choose an arbitrary set of patterns to represent each symbol. However, with Chalmers' experiment and the work in Chapter 3, the patterns for representing the symbols were pre-chosen and the size of the patterns representing the symbols was 13 units, thus if we are restricted to identity mappings with these patterns, the solution requires 65 hidden units. The next section shows how the 1 unit representation from Chapter 3 can be mapped to a representation requiring only 4 units and binary patterns.

The sigmoid activation solution with the 1 unit representation

Using the identity mapping to map the input symbol into bank 1 of the hidden-layer and then to map this bank into the symbolic output requires hidden-layers made up of banks of units of the same size as the input layer. This is inflexible if the patterns representing the symbols are prechosen for some reason. Instead the patterns representing the symbols could be mapped onto a smaller representation to reduce the size of the network. Such a mapping must be a reversible mapping and in order for the identity mapping to work, the smaller representation must also consist solely of values on one or other extreme of the sigmoid function. Table 4.1 illustrates how the orthogonal patterns from the 1 unit representation can be mapped onto a set of binary patterns of 4 units.

That the mapping from the 1 unit representation to the binary representation can be implemented with sigmoidal units can be seen as follows. This mapping corresponds to the mapping **T** in Figure 4.1. The pattern for each symbol can thus be considered as the pattern presented at the input layer with the corresponding binary pattern being the pattern of activation that results at the first bank of the hidden layer. In the first 4 rows, a "+" indicates that the value of the unit in the input layer is added to the value

unit 4	+	+	+	+	+	+	+	+	0	0	0	0							
unit 3	+	+	+	+	0	0	0	0	+	+	+	+							
unit 2	+	+	0	0	+	+	0	0	+	+	0	0							
unit 1	+	0	+	0	+	0	+	0	+	0	+	0							
john	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
michael	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
helen	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
diane	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
chris	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
love	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1
hit	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
betray	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
kill	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0
hug	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0
is	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0
by	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0

Table 4.1: The mapping from the 1 unit representation to a binary representation.

of the corresponding unit in the first bank of the hidden layer. A "0" indicates that the value of the unit in the input layer is discarded (by setting the connection weight to zero). Note that because the final unit in the 1 unit representation is always zero it can be ignored. The third column indicates the new patterns for each symbol that result from this encoding. Thus the value of each unit in the internal binary patterns can be calculated as the sum of all the units in the original representation with a "+" next to them. The value of each unit u in the first bank of the hidden-layer can thus be approximated with the sigmoid function as follows:

$$u = \text{sigmoid}(\beta \times \sum_{k \in P} (a_k) - \alpha)$$

P is the set of input units with a "+" next to them for hidden-layer unit u. β is the value put on the weights between the units in P and the unit u in the first bank of the hidden-layer. All the other connections from the input layer to the unit u are set to zero. $-\alpha$ is the value put on the bias weights for the units in the first bank of the hidden-layer. a_k is the activation of the relevant unit k in P. When the value of the summation is zero, then:

$$u = \lim_{\alpha \to \infty} \text{sigmoid}(-\alpha) = 0$$

This is because as α becomes larger u travels further down the lower extreme of the sigmoid function. If $\beta = 2\alpha$ and the summation is equal to 1 (i.e. one and only one of the original units with a "+" next to it is on, as is the case with the patterns considered here) then:

$$u = \lim_{\alpha \to \infty} \text{sigmoid}(\alpha) = 1$$

This is because $u = \text{sigmoid}(\beta - \alpha)$ and $\beta - \alpha = \alpha$, so as α is increased, u travels further and further along the upper extreme of the sigmoid. Thus the mapping can be approximated to an arbitrary degree with sigmoidal units by choosing suitably large α and β.

Now the mapping from the binary patterns back to the original patterns needs to be considered. This corresponds to the mapping **S** in Figure 4.1. For each binary pattern, which is of length $\lceil \lg(M) \rceil$, where M is the number of symbols, in this case 12, there will be a corresponding output unit, u, that should be on if and only if that binary pattern is present in the first bank of the hidden-layer. So u should thus compute:

$$u = \text{sigmoid}(\beta(\sum_{k \in P} a_k - \sum_{j \in Q} a_j + \alpha - (\lceil \lg(M) \rceil - 0.9)))$$

Where P is the set of units from the first bank of the hidden-layer that should be "1"; Q is the set of units from the first bank that should be "0", and α is the number of units that should be "0". Note that if all the units have the correct value then adding the first summation and α yields $\lceil \lg(M) \rceil$. The second summation will come to zero and the output unit computes the following:

$$u = \text{sigmoid}(\beta(\lceil \lg(M) \rceil - \lceil \lg(M) \rceil + 0.9)) = \lim_{\beta \to \infty} \text{sigmoid}(0.9\beta) = 1$$

If any of the units has the wrong value then:

$$\sum_{k \in P} a_k - \sum_{j \in Q} a_j \leq \lceil \lg(M) \rceil - 1$$

Thus the unit computes:

$$u = \lim_{\beta \to \infty} \text{sigmoid}(\beta(-\gamma + 0.9)) = 0$$

Where:

$$\gamma = \lceil \lg(M) \rceil - (\sum_{k \in P} a_k - \sum_{j \in Q} a_j) \geq 1$$

Hence $-\gamma + 0.9 < 0$. β is the value of the weights connecting the units that should be "1" to u; $-\beta$ is the value of the weights connection the units that should be "0" to u. $\beta(\alpha - \lceil \lg(M) \rceil + 0.9)$ is the value of the bias weight for the output unit u. Hence the reverse mapping can be implemented with sigmoidal units.

Note that this scheme can be extended for any number of orthogonal patterns. Each additional unit in the new representation doubles the number of orthogonal patterns that can be represented using binary patterns. Thus for M symbols represented by orthogonal patterns, this scheme will yield a new representation involving $\lceil \lg(M) \rceil$ units and so the size of hidden-layer required when the symbols are represented by orthogonal patterns is $L \times \lceil \lg(M) \rceil$ units as in the previous section. In the case of the symbols and

unit 5									+					
unit 4								+	0					
unit 3							+	0						
unit 2						+	0							
unit 1			+	0										
john	0	1	0	0	0	0	1	0	0	0	0	0	0	1*1**
michael	0	1	0	0	0	0	0	1	0	0	0	0	0	1*0**
helen	0	1	0	0	0	0	0	0	1	0	0	0	0	1**1*
diane	0	1	0	0	0	0	0	0	0	1	0	0	0	1**0*
chris	0	1	0	0	0	0	0	0	0	0	1	0	0	1***1
love	0	0	1	0	0	0	1	0	0	0	0	0	0	0*1**
hit	0	0	1	0	0	0	0	1	0	0	0	0	0	0*0**
betray	0	0	1	0	0	0	0	0	1	0	0	0	0	0**1*
kill	0	0	1	0	0	0	0	0	0	1	0	0	0	0**0*
hug	0	0	1	0	0	0	0	0	0	0	1	0	0	0***1
is	0	0	0	0	1	0	1	0	0	0	0	0	0	*11**
by	0	0	0	0	0	1	1	0	0	0	0	0	0	*01**

Table 4.2: The mapping from Chalmers' representations to internal ternary representations.

sequences used in Chalmers' experiment, this yields a SRAAM with 20 hidden units, well within the range used. Thus we know that a solution for the 1 unit representation exists for hidden layers of ≥ 20 units.

The sigmoid activation solution with Chalmers' patterns

With Chalmers' patterns a different scheme is used. Table 4.2 illustrates this. This yields a new representation of 5 units in size, and thus a hidden-layer of 25 units. The "*"s in column 3 indicate where the internal pattern has the value 0.5 in one of its units. "0.5"s can be copied as well as "1"s and "0"s because sigmoid$(0) = 0.5$. If we set the bias weight for a unit to $-0.5H$, then the unit can compute sigmoid$(x - 0.5H)$ where x is the net input. If $x = 0.5H$ then the unit outputs 0.5. If H is the value used to multiply the

identity matrix as above, then when 0.5 is copied from 1 layer to another, the relevant output unit computes:

$$\text{sigmoid}(0.5H - 0.5H) = 0.5$$

Hence "0.5"s can be copied exactly by sigmoidal units whilst retaining the approximate identity function for "0"s and "1"s. The first 5 rows of Table 4.2 indicate how to construct the internal patterns from the original patterns as before. A "+" indicates that a unit from the original pattern is added to the value of the unit in the internal pattern. A "0" indicates where the internal unit is set to zero. A blank space indicates the unit takes on the value 0.5. Thus a "+" or a "0" indicates that one of two units is activated in the original patterns, otherwise neither of them are active. The patterns this yields are illustrated in the third column as before.

Firstly, the mapping from the original pattern to the internal pattern is considered. Aside from unit 5, each unit in the internal pattern takes inputs from 2 units in the original pattern – one is a unit, u_1 with a "+" next to it and the other, u_2, with a "0" next to it. If neither of these units is on, the unit should output the value 0.5. If u_1 has the value 1, then we wish the unit to output 1, and if u_2 has the value 1 the unit should output 0. This can be achieved by setting the bias weight for the unit to 0, the weight on the connection from u_1 to α and the weight on the connection from u_2 to $-\alpha$. If u_1 and u_2 are both zero the unit outputs sigmoid(0) = 0.5 due to the bias weight being set to 0. If $u_1 = 1$ and u_2=0, the unit outputs:

$$\lim_{\alpha \to \infty} \text{sigmoid}(\alpha) = 1$$

If $u_2 = 1$ and u_1=0, the unit outputs:

$$\lim_{\alpha \to \infty} \text{sigmoid}(-\alpha) = 0$$

Thus the mapping can be performed to arbitrary precision with sigmoidal units by setting α to a suitably large value. Unit 5 only has 1 unit contributing to it, which is a unit with a "+" next to it. The same set up as for the other units is used, discarding the unit u_2 from consideration. Thus the bias for unit 5 will be set to 0 and the connection from the relevant unit in the original representation has a weight set to α. This scheme works so long as, for each group of original units contributing to an internal unit, only 1 unit is on at any time, or none of the units is on as is the case here.

Now the mapping from the internal patterns back to the original patterns is considered. Each internal pattern contributes to 2 units in the original patterns (o_1 and o_2). These units must be zero if the internal unit is set to 0.5. If the internal unit is set to 1 then $o_1 = 1$ and $o_2 = 0$. If the internal unit is set to 0 then $o_1 = 0$ and $o_2 = 1$. This can be achieved by having:

$$o_1 = \text{sigmoid}(\beta_1 u - \alpha_1)$$
$$o_2 = \text{sigmoid}(-\beta_2 u + \alpha_2)$$

Where $-\alpha_1$ is the bias weight for o_1 and β_1 is the weight for the connection between the internal unit u and o_1. α_2 is the bias weight for o_2 and $-\beta_2$ is the weight for the connection between the internal unit u and o_2. If $\alpha_1 = \frac{3}{4}\beta_1$ and $\beta_2 = 4\alpha_2$ then if $u = 1$:

$$o_1 = \lim_{\alpha_1 \to \infty} \text{sigmoid}(\frac{\alpha_1}{3}) = 1$$
$$o_2 = \lim_{\alpha_2 \to \infty} \text{sigmoid}(-3\alpha_2) = 0$$

If $u = 0.5$:

$$o_1 = \lim_{\alpha_1 \to \infty} \text{sigmoid}(-\frac{\alpha_1}{3}) = 0$$
$$o_2 = \lim_{\alpha_2 \to \infty} \text{sigmoid}(-\alpha_2) = 0$$

If $u = 0$:

$$o_1 = \lim_{\alpha_1 \to \infty} \text{sigmoid}(-\alpha_1) = 0$$
$$o_2 = \lim_{\alpha_2 \to \infty} \text{sigmoid}(\alpha_2) = 1$$

Which allows the required behaviour of o_1 and o_2 to be approximated to arbitrary precision by setting $\alpha_1, \alpha_2, \beta_1$ and β_2 to suitably large values. Thus the mapping can be performed both ways. This can be generalised to any set of patterns composed of N pairs of units only 1 of which is on in any particular pattern, yielding a new representation of N units, and thus at least halving the size of the hidden-layer required from the corresponding SRAAM. The presence of redundant units will allow greater reductions.

We can obtain a smaller size of patterns than this as Table 4.3 shows. Essentially units 2 and 5 in Table 4.2 are combined. To see that this works,

unit 4								+	0					
unit 3							+	0						
unit 2					+	0					+			
unit 1		+	0											
john	0	1	0	0	0	0	1	0	0	0	0	0	0	1*1*
michael	0	1	0	0	0	0	0	1	0	0	0	0	0	1*0*
helen	0	1	0	0	0	0	0	0	1	0	0	0	0	1**1
diane	0	1	0	0	0	0	0	0	0	1	0	0	0	1**0
chris	0	1	0	0	0	0	0	0	0	0	1	0	0	11**
love	0	0	1	0	0	0	1	0	0	0	0	0	0	0*1*
hit	0	0	1	0	0	0	0	1	0	0	0	0	0	0*0*
betray	0	0	1	0	0	0	0	0	1	0	0	0	0	0**1
kill	0	0	1	0	0	0	0	0	0	1	0	0	0	0**0
hug	0	0	1	0	0	0	0	0	0	0	1	0	0	01**
is	0	0	0	0	1	0	1	0	0	0	0	0	0	*11*
by	0	0	0	0	0	1	1	0	0	0	0	0	0	*01*

Table 4.3: A smaller mapping from Chalmers' representations to internal ternary representations.

firstly note that internal units 1, 3 and 4 in this representation are the same as in the previous representation and the mapping is performed in the same manner in both directions. However internal unit 2 takes input from or contributes to the values of 3 units in the original representation, depending on which direction the mapping is going.

Taking the mapping from the original representation to the internal representation first, the value of unit 2 can be computed as follows:

$$u = \text{sigmoid}(\beta(\sum_{k \in P} a_k - \sum_{j \in Q} a_j))$$

Where P is the set of units with a "+" next to them and Q is the set with a "0" next to them (which only contains 1 member here). Since at any time only 1 of the units in P and Q will be on, the unit will compute either:

$$u = \lim_{\beta \to \infty} \text{sigmoid}(\beta) = 1$$

If the unit is in P. Or:

$$u = \lim_{\beta \to \infty} \text{sigmoid}(\beta(-1)) = 0$$

If the unit is in Q. Or:

$$u = \text{sigmoid}(\beta(0)) = 0.5$$

If no units are on. Thus the bias weight is set to 0, and the weights from the units with a "+" next to them are set to β and the weights from the units with a "0" next to them are set to $-\beta$.

The mapping from internal representation back to the original representation works as before except for the 3 units o_1, o_2 and o_3 that contributed to unit 2 above.

$$o_1 = \text{sigmoid}(\beta(u_2 + u_3 - 1.75))$$
$$o_2 = \text{sigmoid}(\beta(u_3 - u_2 - 0.75))$$
$$o_3 = \text{sigmoid}(\beta(u_2 - u_3 - 0.25))$$

Where β is the value of the weights from u_2 and u_3 to o_1 and the weights from u_3 to o_2 and u_2 to o_3 and $-\beta$ is the value of the rest of the connections and -0.5β is the value of the bias weight on o_2, -0.25β is the bias weight

on o_3 and -1.75β is the value of the bias weight on o_1. If u_2 and u_3 are both set to "1" then:

$$o_1 = \lim_{\beta \to \infty} \text{sigmoid}(0.25\beta) = 1$$
$$o_2 = \lim_{\beta \to \infty} \text{sigmoid}(-0.75\beta) = 0$$
$$o_3 = \lim_{\beta \to \infty} \text{sigmoid}(-0.25\beta) = 0$$

If $u_3 = 1$ and $u_2 = 0$ then:

$$o_1 = \lim_{\beta \to \infty} \text{sigmoid}(-0.75\beta) = 0$$
$$o_2 = \lim_{\beta \to \infty} \text{sigmoid}(0.25\beta) = 1$$
$$o_3 = \lim_{\beta \to \infty} \text{sigmoid}(-1.25\beta) = 0$$

If $u_2 = u_3 = 0.5$ then:

$$o_1 = \lim_{\beta \to \infty} \text{sigmoid}(-0.75\beta) = 0$$
$$o_2 = \lim_{\beta \to \infty} \text{sigmoid}(-0.75\beta) = 0$$
$$o_3 = \lim_{\beta \to \infty} \text{sigmoid}(-0.25\beta) = 0$$

If $u_2 = 1$ and $u_3 = 0.5$ then:

$$o_1 = \lim_{\beta \to \infty} \text{sigmoid}(-0.25\beta) = 0$$
$$o_2 = \lim_{\beta \to \infty} \text{sigmoid}(-1.25\beta) = 0$$
$$o_3 = \lim_{\beta \to \infty} \text{sigmoid}(0.25\beta) = 1$$

If $u_2 = 0.5$ and $u_3 = 1$ then:

$$o_1 = \lim_{\beta \to \infty} \text{sigmoid}(-0.25\beta) = 0$$
$$o_2 = \lim_{\beta \to \infty} \text{sigmoid}(-0.25\beta) = 0$$
$$o_3 = \lim_{\beta \to \infty} \text{sigmoid}(-0.75\beta) = 0$$

Which covers all the situations that occur with Chalmers' patterns. Thus the required mapping can be implemented with sigmoid units.

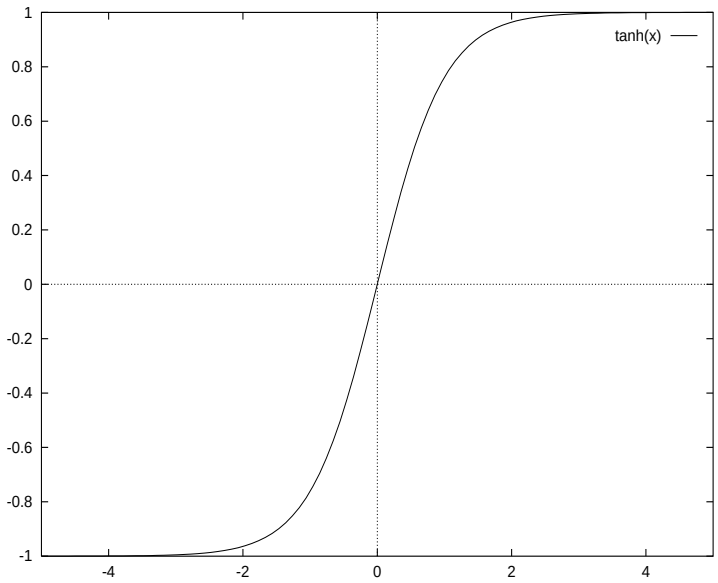

Figure 4.3: The hyperbolic tangent function

Hyperbolic tangent activation with arbitrary patterns

Now the solutions for the SRAAMs using hyperbolic tangent activations are derived. The hyperbolic tangent function has the same shape as the sigmoid function except that the range is (-1,1). Figure 4.3 illustrates this. As with the sigmoidal function, the identity mapping can be approximated to an arbitrary degree for the extremes of the hyperbolic tangent function, i.e. "1"s and "-1"s can be copied to a high degree of precision. Thus for a set of sequences of length L made up out of combinations of M symbols, an SRAAM can be constructed with $L \times \lceil \lg(M) \rceil$ hidden units as for the sigmoidal solution, by selecting patterns for the symbols consisting solely of "1"s and "-1"s. Because $\tanh(0) = 0$ the matrices for the identity mappings need only consist of the identity matrix multiplied by some large value H, whilst the bias weights are set to zero.

Hyperbolic tangent activation with the 1 unit representation

Table 4.4 illustrates the mapping from the 1 unit representations to the internal representations for the hyperbolic tangent units.

Firstly the mapping from the 1 unit representation to the bipolar repre-

unit 4	+	+	+	+	+	+	+	+	-	-	-	-	
unit 3	+	+	+	+	-	-	-	-	+	+	+	+	
unit 2	+	+	-	-	+	+	-	-	+	+	-	-	
unit 1	+	-	+	-	+	-	+	-	+	-	+	-	

john	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
michael	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1
helen	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	1	1
diane	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1
chris	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	-1	1
love	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	1
hit	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	1
betray	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	1
kill	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	1	1	1	-1
hug	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	1	1	-1
is	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	1	-1	1	-1
by	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	1	-1

Table 4.4: The mapping from the 1 unit representation to a bipolar representation for hyperbolic tangent units.

sentation is considered. In the first 4 rows a "+" indicates that the original unit's value is added to the internal unit's value. For units 1 and 2, a "-" indicates that the input unit's value is subtracted from the value of the internal unit. For units 3 and 4, the "-" indicates that twice the value of the input unit is subtracted from the internal unit. This is because there are twice as many "+"'s as there are "-"s, so doubling the values of the "-"s is needed to balance things out. Note that this would not be necessary if there were 16 ($= 2^4$) patterns to encode. Thus for units 1 and 2 to calculate the value we simply added all the inputs with a "+" and subtract all the inputs with a minus. This can be approximated with the hyperbolic tangent as follows:

$$u = \tanh(\beta(\sum_{k \in P} a_k - \sum_{j \in Q} a_j))$$

P is the set of input units with a "+" next to them and Q is the set with a "-" next to them. a_k is the activation of unit k. If the units in P and Q all have the same value (e.g. all -1) then $u = 0$. However with the orthogonal patterns 1 and only 1 unit will be set to 1. If that input unit is in P then:

$$\sum_{k \in P} a_k = \sum_{j \in Q} a_j + 1$$

The internal unit will thus compute:

$$u = \lim_{\beta \to \infty} \tanh(\beta) = 1$$

If the input unit is in Q then:

$$\sum_{k \in P} a_k = \sum_{j \in Q} a_j - 1$$

The internal unit will thus compute:

$$u = \lim_{\beta \to \infty} \tanh(-\beta) = -1$$

Hence units 1 and 2 have the desired behaviour. Now consider units 3 and 4. They compute:

$$u = \tanh(\beta(\sum_{k \in P} a_k - 2\sum_{j \in Q} a_j))$$

As before if the values in P and Q are the same then $u = \tanh(0) = 0$, thus with the 1 unit representation if the input unit that is "1" is in P then:

$$\sum_{k \in P} a_k = 2 \sum_{j \in Q} a_j + 1$$

And:

$$u = \lim_{\beta \to \infty} \tanh(\beta) = 1$$

On the other hand if the input unit is in Q then:

$$\sum_{k \in P} a_k = 2 \sum_{j \in Q} a_j - 2$$

And:

$$u = \lim_{\beta \to \infty} \tanh(-2\beta) = -1$$

Which is the desired behaviour for units 3 and 4. Hence the mapping from the 1 unit representation to the internal representation can be implemented with hyperbolic tangent units.

Now the reverse mapping is considered. The proof is similar to the proof for the sigmoidal version. For each bipolar pattern there is an output unit u that should be on only when the pattern is present in bank 1 of the hidden-layer. The activation of u can be computed thus:

$$u = \tanh(\beta(\sum_{k \in P} a_k - \sum_{j \in Q} a_j - \lceil \lg(M) \rceil + 0.5))$$

Where P is the set of units from bank 1 of the hidden-layer that should be "1" and Q is the set of units that should be "-1". If the units all have the correct values then the two summations will combine to produce $\lceil \lg(M) \rceil$ and the above reduces to:

$$u = \lim_{\beta \to \infty} \tanh(\beta(0.5)) = 1$$

If any of the units in bank 1 has the wrong value then the summations will combine to produce at most $\lceil \lg(M) \rceil - 1$. Thus the net input to the output unit u will be at most $\beta(-0.5)$, and:

$$\lim_{\beta \to \infty} \tanh(\beta(-0.5)) = -1$$

If net input is less than $\beta(-0.5)$ the activation will simply be closer still to -1. β is the value of the weights on the connections from the units in P to relevant output unit, whilst $-\beta$ is the value of the weights on the connections from the units in Q. The bias weight is set to 0.5. Thus by choosing suitably large β the mapping from bipolar patterns back to the 1 unit representation can be implemented in hyperbolic tangent units.

Hyperbolic tangent activation with Chalmers' patterns

The mapping from the 2 unit representation to an internal representation for hyperbolic tangent units can be performed by adapting either of the solutions presented for the sigmoidal solution. Here the mapping to the equivalent 4 unit representation is presented to illustrate the adaptations needed. Table 4.5 illustrates the mapping. In the rows for each hidden unit, the "0"s show where the hidden unit outputs the value "0" regardless of the value of the corresponding input unit, the "+"s show where the unit takes the value "1" if the corresponding input unit is "1" and the "-" shows where the hidden unit outputs the value "-1" if the corresponding input unit is "1".

Taking the mapping from the 2 unit representation to the internal representation first, internal units 1, 3 and 4 compute the following:

$$u = \tanh(\beta(u_1 - u_2))$$

Where u_1 is the input unit that should be "1" if the hidden unit is to output "1", u_2 is the input unit that should be "1" if the hidden unit is to output "-1", β is the value of the weight connecting u_1 to u, $-\beta$ is the value of the weight connecting u_2 to u and 0 is the value of the bias weight. If $u_1 = 1$ and u_2=-1 then:

$$u = \tanh(\beta(1 - (-1))) = \lim_{\beta \to \infty} \tanh(2\beta) = 1$$

If $u_1 = -1$ and $u_2 = -1$ then:

unit 4	0	0	0	0	0	0	0	0	+	-	0						
unit 3	0	0	0	0	0	0	+	-	0	0	0						
unit 2	0	0	0	0	+	-	0	0	0	0	+						
unit 1	0	+	-	0	0	0	0	0	0	0	0						
john	-1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	1	0	1	0
michael	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	1	0	-1	0
helen	-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0	1
diane	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	1	0	0	-1
chris	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	1	1	0	0
love	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	0	1	0
hit	-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	0	-1	0
betray	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	0	0	1
kill	-1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	0	0	-1
hug	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	1	0	0
is	-1	-1	-1	-1	1	-1	1	-1	-1	-1	-1	-1	-1	0	1	1	0
by	-1	-1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	1	0

Table 4.5: The mapping from the 2 unit representation to a ternary representation for hyperbolic tangent units.

$$u = \tanh(\beta(-1 - (-1)) = \tanh(0\beta) = 0$$

If $u_1 = -1$ and $u_2 = 1$ then:

$$u = \tanh(\beta(-1 - 1)) = \lim_{\beta \to \infty} \tanh(-2\beta) = -1$$

Thus, for units 1,3 and 4, the mapping from 2 unit representation to the internal representation can be performed with hyperbolic tangent units be choosing a suitably large value of β. Hidden unit 2 would compute the following:

$$u = \tanh(\beta(u_1 - u_2 + u_3 + 1))$$

Where u_1 and u_3 are the 2 units which, if either are on hidden unit 2 should ouput "1", and u_2 is the unit which should be "1" if the hidden unit is to output "-1". Recall that only 1 of u_1, u_2 and u_3 will be on at any point. If *either* u_1 or u_3 is "1" then:

$$u = \tanh(\beta(1 + 1 - 1 + 1)) = \lim_{\beta \to \infty} \tanh(2\beta) = 1$$

If u_2 is "1" then:

$$u = \tanh(\beta(-1 - 1 - 1 + 1)) = \lim_{\beta \to \infty} \tanh(-2\beta) = -1$$

If all of u_1, u_2 and u_3 are "-1" then:

$$u = \tanh(\beta(-1 + 1 - 1 + 1)) = \tanh(0) = 0$$

β is the value of the weights connecting u_1 and u_3 to u whilst $-\beta$ is the value of the weight connecting u_2 to u. β is also the value of the bias weight for u. Hence the mapping can be implemented for hidden unit 2.

Now the reverse mapping is considered. For each of hidden units 1, 3 and 4, there are 2 output units, o_1 which should be "1" when the hidden unit is "1" and "-1" otherwise, and o_2 which should be "1" when the hidden units is "-1" and "-1" otherwise. These units should compute:

$$o_1 = \tanh(\beta(u - 0.5))$$
$$o_2 = \tanh(\beta(-u - 0.5))$$

Where u is the activation of the relevant hidden unit, β is the value of the weight connecting u to o_1, -0.5β is the bias weight, $-\beta$ is the value of the weight connecting u to o_2 and 0.5β is the bias weight for o_2. If $u = 1$ then:

$$o_1 = \lim_{\beta \to \infty} \tanh(0.5\beta) = 1$$

$$o_2 = \lim_{\beta \to \infty} \tanh(-1.5\beta) = -1$$

If $u = -1$ then:

$$o_1 = \lim_{\beta \to \infty} \tanh(-1.5\beta) = -1$$

$$o_2 = \lim_{\beta \to \infty} \tanh(0.5\beta) = 1$$

If $u = 0$ then:

$$o_1 = \lim_{\beta \to \infty} \tanh(-0.5\beta) = -1$$

$$o_2 = \lim_{\beta \to \infty} \tanh(-0.5\beta) = -1$$

Hidden unit 2 takes input from 3 units, and the values of the corresponding output units o_1, o_2 and o_3 need to be calculated using both hidden units 2 and 3 as follows:

$$o_1 = \tanh(\beta(u_2 + u_3 - 1.5))$$

$$o_2 = \tanh(\beta(u_3 - u_2 - 1.5))$$

$$o_3 = \tanh(\beta(u_2 - u_3 - 0.5))$$

Where β is the value of the weights from u_2 and u_3 to o_1 and the weights from u_3 to o_2 and u_2 to o_3 and $-\beta$ is the value of the rest of the connections and -0.5β is the value of the bias weights on o_2 and o_3 and -1.5β is the value of the bias weight on o_1. If u_2 and u_3 are both set to "1" then:

$$o_1 = \lim_{\beta \to \infty} \tanh(0.5\beta) = 1$$

$$o_2 = \lim_{\beta \to \infty} \tanh(-1.5\beta) = -1$$

$$o_3 = \lim_{\beta \to \infty} \tanh(-0.5\beta) = -1$$

If $u_3 = -u_2 = 1$ then:

$$o_1 = \lim_{\beta \to \infty} \tanh(-1.5\beta) = -1$$

$$o_2 = \lim_{\beta \to \infty} \tanh(0.5\beta) = 1$$

$$o_3 = \lim_{\beta \to \infty} \tanh(-2.5\beta) = -1$$

If $u_2 = u_3 = 0$ then:

$$o_1 = \lim_{\beta \to \infty} \tanh(-1.5\beta) = -1$$

$$o_2 = \lim_{\beta \to \infty} \tanh(-1.5\beta) = -1$$

$$o_3 = \lim_{\beta \to \infty} \tanh(-0.5\beta) = -1$$

If $u_2 = 1$ and $u_3 = 0$ then:

$$o_1 = \lim_{\beta \to \infty} \tanh(-0.5\beta) = -1$$

$$o_2 = \lim_{\beta \to \infty} \tanh(-2.5\beta) = -1$$

$$o_3 = \lim_{\beta \to \infty} \tanh(0.5\beta) = 1$$

If $u_2 = 0$ and $u_3 = 1$ then:

$$o_1 = \lim_{\beta \to \infty} \tanh(-0.5\beta) = -1$$

$$o_2 = \lim_{\beta \to \infty} \tanh(-0.5\beta) = -1$$

$$o_3 = \lim_{\beta \to \infty} \tanh(-1.5\beta) = -1$$

Which covers all the situations that occur with Chalmers' patterns. Thus the required mapping can be implemented with hyperbolic tangent units.

4.3 Could it be that the right set of parameters simply has not been tried?

The short answer to this question is yes. Consider back-propagation. There are two parameters controlling the steps taken in gradient descent; the momentum and the learning rate. These take any continuous value, in the range from 0 to 1. One simply cannot try every possible combination of values for these parameters as there is an infinite number of values each of the parameters can take. With both training methods used, there is also the initial set of weights chosen. Typically these are randomly distributed between $-x$ and x where x is a value less than 1. Again the choice of this distribution is arbitrary and one cannot exhaustively try every possibility.

However, if after trying a wide range of values for such parameters one fails to obtain success then simply trying more and more values is not an intelligent way of finding out which parameters would work or whether such a set of parameters exist. At this point an analysis of the dynamics of training, the hidden layer patterns produced, and a consideration of whether the solution exists and if it does how difficult it is to find becomes essential if an understanding is to be gained of why the training fails. Also a sufficiently powerful training method should find a solution wherever one starts, though it may take a long time. In the work reported in Chapter 3, different values for the momentum and learning rate in back-propagation and several different initial distributions of weights were tried without success, thus warranting an analysis of the training. Failure to find a solution after many tries with wide ranges of parameters suggests either the solution is difficult to find or the training method lacks the power to find it except from a favourable starting point. Both back-propagation and Kwasny and Kalman's method, which is based on conjugate gradient, can get stuck if there are a lot of local minima in the solution space – neither is guaranteed to find the global optimum.

The implication of the above is that because a wide range of parameters have been tried, a deeper analysis is warranted rather than to go on trying ever more sets of parameters. A solution that can only be found under a favourable set of parameters is perhaps indicative of an inherently difficult problem or a problem with the training method being used.

4.4 Determining the cause of failure

Section 4.2 mentioned several possible causes of failure and, for networks with hidden layers of 20 or more units proved that a solution does exist. Having eliminated one possible cause of failure the aim of the rest of the work in this chapter is to find out what the cause of failure is.

4.4.1 Determining the difficulty of finding a solution.

Could it be that the solution is simply too difficult to find? Both back-propagation and Kwasny and Kalman's method[3] are local optimisation methods that are vulnerable to being caught in local optima. If the error landscape is rugged (contains many local optima) it may be difficult for local optimisation methods to find the global optimum for a given task. Here two ways of determining how difficult an error-free optimum might be to find are used:

- Use of simulated annealing to train the SRAAM. Simulated annealing is normally used to train Boltzmann machines, but can be adapted for the networks used here. The idea is simple. A random change is made to the weights and then this change is accepted with a probability dependent on a variable known as the temperature and whether or not the change leads to a reduction in the error. By gradually reducing the temperature, the network will eventually settle into an optimum. If the temperature is reduced slowly enough, then it is guaranteed that simulated annealing will find the global optimum for a given task in a finite length of time. In practice this means that if, after several long training runs the network has failed to find the global optimum or at least an optimum where encoding and decoding are error-free, then one can conclude that an error-free solution is difficult to find. The version of simulated annealing used here is that presented in "Numerical Recipes in C" Press et al. (1992), which is based on the downhill simplex method. A simplex of N+1 points is successively transformed with a transformation accepted if it leads to lower error (subject to noise proportional to the temperature being added to the error). N is the number of weights and thus the simplex points represent different sets of weights to use in the network. The results of an attempt to train

[3]Kwasny and Kalman's method is based on conjugate gradient training

a 26-13-26 SRAAM to learn 60 sequences for 1.5 million iterations this way are presented in Section 4.5.1.

- Using a hand derived solution. In Section 4.2.1 a proof that a solution exists for SRAAMs with hidden layers of 20 units was presented. This can be used to indicate the difficulty of training as follows. One derives the weights that implement the solution. Then training runs are performed using these weights with increasing amounts of noise added until the training fails to find an error free solution. This will indicate how much of a deviation from the solution's weights is required before the training method fails. Note that it is possible for there to be other error free solutions in the vicinity of the hand-derived solution that the network might find. If the deviation is small it indicates that the error free solutions are findable only within a small area of the weight-space.Section 4.5.2 presents the results of doing this for networks with 20 and 39 hidden units, trained with both back-propagation and Kwasny and Kalman's method.

A cautionary note needs to be made here which is that both the above methods give results that are specific to the error function used in training. The error function is part of what determines the shape of the error landscape and thus how much impact local minima can have. The results of the above analyses will thus be specific to the error functions used in training the networks here. Two error functions are used in training, the standard sum of squares measure and the function used by Kwasny and Kalman in their work. Whether improved training could come about with a different error function is an important question, but devising an error function to do so would require knowledge of the problems that hinder training, thus requiring an analysis to be performed.

4.4.2 Analyses of the hidden-layer states

How the hidden-layer states are organised in hyperspace will determine the nature of the representations produced by the SRAAM, how sensitive they are to noise and how well separated the encoding and decoding trajectories are. Analyses of the hidden-layer states aimed at finding out how they are organised are presented in Chapter 5.

4.4.3 Analyses of the network weights

The analyses of the hidden-layer states should indicate whether there are any problems in the way the representations and trajectories cluster and evolve or in the nature of the representations developed by the SRAAM. However it would also be relevant to know whether problems exist in the nature of the overall solution developed by the SRAAM, i.e. the set of weights produced. For example, one indication of a local optimum is if all the weights are set to high values making it difficult for the network to move away from that point in weight-space. Also if a very small perturbation of the weights results in the encoding and decoding performance collapsing, it would suggest the network has found a solution that is sensitive to noise on the weights. In this chapter the following analyses are presented in order to determine whether such problems might exist with the SRAAM:

- Find the range, average and average magnitude of the weights at different points in training. Trends in these values may give clues as to why training fails. For example if the weights increase in magnitude with more and more training it may make it difficult for the network to move away from the area of weight-space it has reached, thus causing it to get stuck in a local optimum.

- Lesioning the weights. Here all the weights are perturbed by some percentage of their value (or by a small constant if the weight is zero), and the encoding and decoding performance of the network is subsequently noted. By varying the percentage perturbation, one can measure the sensitivity to noise in the weights. If only small perturbations lead to significant errors this indicates a sensitive solution has been found.

4.5 Results of the Analysis

This section present the results of the analyses discussed above.

4.5.1 Simulated Annealing

A 26-13-26 SRAAM was trained for 1.5 million iterations via simulated annealing. At the end of training all the points in the simplex had a sum squared error in the range 170 to 171, well above the error reached by Kwasny and

Kalman's training or back-propagation in Chapter 3. Initially this was intended to be a preliminary run before trying the full set of sequences and networks with hidden layers of up-to 39 units as used in Chapter 3. However this run took 2 weeks to perform on a twin 300 MHz processor SunUltra2 implying prohibitively long training runs for anything more complex than this.

Whilst the Simulated Annealing had failed to find a solution by this point, it might have left the simplex points in a region of hyperspace from which the solution could be found. Testing all the points to see if this was the case would have been prohibitively expensive involving 754 training runs so 10 points were selected, the best and worst plus 8 randomly selected points. If a solution allowing error-free encoding and decoding was easily findable from within the simplex, it was likely that a solution would be found from a selection of random points from the simplex. Back-propagation was used to train the networks from these points[4] In every case the error rose rapidly, indicating that the learning rate or momentum were too high, hence the training runs were run again with the momentum eliminated. A brief drop in the error to 169.99 or so was observed in the first few iterations before it rose rapidly again. A final repeat of the training was performed where the learning rate was reduced to 1.0×10^{-6}. Each run involved a gentle decline in the error to 169.99 or so, the lowest value reached being 169.986.

The network with the lowest error obtained from all of this was used to encode and decode the 60 sequences in the training set. None of the sequences was correctly encoded and decoded and the average error per sequence was 2.57. This suggests that none of the points were anywhere near a solution and that the simplex had yet to find the region of hyperspace where a solution exists for this network.

4.5.2 Using hand-derived solutions

Hand-derived solutions for networks employing 20 and 39 hidden units, sigmoidal and hyperbolic tangent units, and back-propagation training were implemented (based on the proofs in Section 4.2.2) and used with 0%, 1%, 10% and 50% noise as the starting points for several training runs. Table 4.6 summarises the results of this experiment. The columns are as follows:

[4]With Kwasny and Kalman's method the simulator would need to have been altered in order to take these weights in due to a different gain being used. It would also have taken much longer to train.

Network	Act	Rep	Noise	Err Start	Err Fin	Enc Start	Enc Fin
33-20-33	sig	1 unit	0%	14.96	8.53	0.00	0.00
33-20-33	sig	1 unit	1%	15.44	8.60	0.00	0.00
33-20-33	sig	1 unit	10%	36.68	10.45	0.14	0.00
33-20-33	sig	1 unit	50%	1220.98	114.49	3.18	1.40
33-20-33	sig	2 unit	0%	11.03	9.94	0.00	0.26
33-20-33	sig	2 unit	1%	12.63	10.02	0.35	0.25
33-20-33	sig	2 unit	10%	88.09	16.37	2.42	0.71
33-20-33	sig	2 unit	50%	1341.34	104.14	3.45	1.63
33-20-33	tanh	1 unit	0%	39.44	9.73	0.00	0.00
33-20-33	tanh	1 unit	1%	45.86	7.15	0.00	0.00
33-20-33	tanh	1 unit	10%	464.28	72.65	1.97	0.22
33-20-33	tanh	1 unit	50%	6960.61	845.90	4.52	2.48
33-20-33	tanh	2 unit	0%	22.32	10.16	0.00	0.03
33-20-33	tanh	2 unit	1%	29.62	11.36	0.54	0.02
33-20-33	tanh	2 unit	10%	506.72	46.52	3.60	0.22
33-20-33	tanh	2 unit	50%	8640.33	915.93	4.70	2.08
52-39-52	sig	1 unit	0%	15.26	5.97	0.00	0.00
52-39-52	sig	1 unit	1%	15.49	5.98	0.00	0.00
52-39-52	sig	1 unit	10%	31.36	7.02	0.00	0.00
52-39-52	sig	1 unit	50%	912.98	137.02	3.85	2.72
52-39-52	sig	2 unit	0%	11.70	5.99	0.00	0.45
52-39-52	sig	2 unit	1%	12.79	5.95	0.29	0.46
52-39-52	sig	2 unit	10%	69.88	8.25	2.78	0.68
52-39-52	sig	2 unit	50%	1013.93	85.52	4.81	2.09
52-39-52	tanh	1 unit	0%	82.31	6.36	0.00	0.00
52-39-52	tanh	1 unit	1%	92.77	9.38	0.00	0.00
52-39-52	tanh	1 unit	10%	919.91	120.87	1.07	0.42
52-39-52	tanh	1 unit	50%	14111.2	803.68	4.55	2.07
52-39-52	tanh	2 unit	0%	56.92	6.56	0.00	0.01
52-39-52	tanh	2 unit	1%	68.51	8.41	0.12	0.02
52-39-52	tanh	2 unit	10%	984.04	60.39	3.92	0.35
52-39-52	tanh	2 unit	50%	16181.6	654.157	4.92	2.26

Table 4.6: The results of using hand derived solutions plus varying amounts of noise as the starting point for training using back-propagation

- The "Network" column indicates the architecture of the SRAAM used.
- The "Act" column indicates the type of units used, either sigmoid (sig) or hyperbolic tangent (tanh).
- The "Rep" column indicates whether the 1 unit representation or the 2 unit representation was used.
- The "Noise" column indicates how much noise was introduced to the weights prior to training.
- The "Err Start" column indicates the network error at the start of training after the noise has been introduced. In the case of sigmoid units this is the sum squared error. In the case of the hyperbolic tangent units this is the error calculated via Kwasny and Kalman's function.
- The "Err Fin" column indicates the network error at the end of training.
- The "Enc Start" column indicates the encoding and decoding errors per sequence on the training set of 130 sequences, after the noise is introduced but before training begins.
- The "Enc Fin" column indicates the encoding and decoding errors per sequence at the end of training.

Several things can be observed with respect to these results:

- In all but 2 cases, 10% noise is sufficient to cause subsequent training to fail to find an error free solution again, and in every case 50% noise is sufficient.
- The solution using the 2 unit representation is less robust to noise than the solution using the 1 unit representation.
- Even without noise being added the trained network can deviate from the error-free solution as in the case of the solutions that use the 2 unit representation. This highlights the "moving target" nature of the training, where the target outputs change over time. Back-propagation does not take this change in the target outputs into account and thus seeks merely to minimise the error at each step of encoding and decoding, ignoring the accumulation of error during the encoding and

Network	Rep	Noise	Err Start	Err Fin	Enc Start	Enc Fin
33-20-33	1 unit	0%	77.26	0.24	0.00	0.00
33-20-33	1 unit	1%	103.05	0.27	0.42	0.00
33-20-33	1 unit	10%	2078	0.19	3.18	0.00
33-20-33	1 unit	50%	4.98×10^{10}	8.67	4.95	0.00
33-20-33	2 unit	0%	48.56	0.12	0.00	0.00
33-20-33	2 unit	1%	73.63	0.11	1.30	0.00
33-20-33	2 unit	10%	2377	0.02	4.30	0.00
33-20-33	2 unit	50%	1.13×10^{10}	0.66	4.96	1.32
52-39-52	1 unit	0%	145.07	0.19	0.00	0.00
52-39-52	1 unit	1%	198.95	0.56	0.89	0.00
52-39-52	1 unit	10%	5299	0.33	3.5	0.00
52-39-52	1 unit	50%	4.12×10^{14}	2.78	4.95	0.96
52-39-52	2 unit	0%	105.31	0.38	0.00	0.00
52-39-52	2 unit	1%	155.89	0.39	1.00	0.00
52-39-52	2 unit	10%	9288	0.02	4.10	0.10
52-39-52	2 unit	50%	5.64×10^{13}	0.75	4.96	1.50

Table 4.7: The results of using hand-derived solutions plus varying amounts of noise as the starting point for training using Kwasny and Kalman's method.

decoding process. Kwasny and Kalman's method has the error derivatives recomputed to take this into account.

- The sigmoid activation usually leads to lower encoding and decoding errors than the hyperbolic tangent activation for the same network size, representation and level of noise added.

- The amount of error induced by 50% noise is much greater than that induced by 10% noise.

With Kwasny and Kalman's method the activation function is hard-coded into the simulator so hand-derived solutions for networks employing hyperbolic tangent activation with 20 and 39 hidden units were implemented and then noise added as before for several training runs. The results are summarised in Table 4.7.

In contrast to the networks trained by back-propagation, the networks trained by Kwasny and Kalman's method typically manages to find an error-

free solution after 10% noise has been added and in some cases after 50% noise has been added. Even in the one case where the solution was not found again after 10% noise was added, the errors in encoding and decoding were lower than with back-propagation. This suggests that Kwasny and Kalman's method is better able to find an error-free solution again. Given that Kwasny and Kalman's method explicitly takes into account the changing output targets and the feedback within the network this is perhaps not surprising. However it begs the question as to why Kwasny and Kalman's method fails to find this solution when started from randomly distributed weights, since an error free solution is "findable" from a starting point consisting of the weights for the hand-derived solution with as much as 50% noise added and where the network error is very high.

Finally, it should be noted that in the case where error-free solutions are discovered, the data *does not indicate* whether or not the hand-derived solution was rediscovered or whether some other error-free solution was discovered. As this analysis was performed automatically and took some time to run, there was no time to perform the further analysis which would indicate this, although Section 5.4 does include an analysis of one of the networks which indicates it found a solution other than the original hand-derived solution.

4.5.3 Networks used for the analyses

This section details the networks that were used for the analyses of the weights presented here and the analyses of the hidden-layer states presented in Chapter 5. Because the solution is only known to exist for SRAAMs with hidden-layers of 20 or more units, networks of at least this size were used. For networks trained by back-propagation SRAAMs using 39 hidden units were used, as was the case in Chapter 3. For networks trained by Kwasny and Kalman's method, SRAAMs with 20 hidden units were used as the solution is known to exist for these, the training of these is much quicker than for SRAAMs with 39 hidden units (a few days compared to 2 weeks to learn 130 sequences) and the performance was close to that achieved with the 39 unit SRAAMs in Chapter 3.

Act Fun	Rep	Seqs	Err	Err/seq Tr	Err/seq Te	% Tr	% Te
sigmoid	1 unit	20	3.07	0.00	2.10	100.00%	0.00%
sigmoid	1 unit	40	37.70	0.90	1.45	35.00%	0.05%
sigmoid	1 unit	80	56.22	0.59	1.08	52.50%	21.25%
sigmoid	1 unit	130	131.37	0.92	1.22	32.31%	17.50%
sigmoid	2 unit	20	2.46	0.00	1.30	100.00%	15.00%
sigmoid	2 unit	40	9.56	0.28	0.63	72.50%	55.00%
sigmoid	2 unit	80	27.97	0.34	0.60	71.25%	50.00%
sigmoid	2 unit	130	131.37	0.96	1.18	33.85%	21.67%
tanh	1 unit	20	14.28	0.20	2.85	80.00%	0.00%
tanh	1 unit	40	98.35	0.75	1.85	45.00%	0.03%
tanh	1 unit	80	179.71	0.70	1.16	40.00%	21.25%
tanh	1 unit	130	362.70	0.90	1.16	23.85%	15.83%
tanh	2 unit	20	7.05	0.00	2.25	100.00%	0.00%
tanh	2 unit	40	40.74	0.20	1.05	80.00%	32.50%
tanh	2 unit	80	142.24	0.51	0.81	52.50%	36.25%
tanh	2 unit	130	243.17	0.59	0.64	47.69%	50.83%

Table 4.8: The encoding and decoding performance of various 52-39-52 SRAAMs trained by back-propagation to encode 20, 40, 80 and 130 sequences respectively.

Back-propagation networks

52-39-52 SRAAMs employing hyperbolic tangent units or sigmoidal units were trained using back-propagation to encode 20, 40, 80 and 130 active and passive sequences. Their performance at the end of training is summarised in Table 4.8. Each column is as follows:

- The "Act Fun" column indicates whether the network used sigmoidal activation or hyperbolic tangent (tanh) activation units. If the latter was used, the Kwasny-Kalman error function was also used as in Chapter 3. This leads to higher figures for the network error than sum of squares.
- The "Rep" column indicates whether the 1 unit representation or 2 unit representation was used (see Chapter 3).
- The "Seqs" column indicates the number of sequences in the training set. The testing set contains the same number of sequences, except when the number of sequences is 130 as there are then only 120 sequences left out of Chalmers' data set.
- The "Err" column give the value of the error function at the end of training.
- The "Err/seq Tr" column gives the encoding/decoding errors per sequence for the training set.
- The "Err/seq Te" column gives the encoding/decoding errors per sequence for the testing set.
- The "% Tr" column gives the percentage of sequences correctly encoded and decoded in the training set.
- The "% Te" column gives the percentage of sequences correctly encoded and decoded in the testing set.

As can be seen, increasing the number of sequences tends to increase the levels of error and move the performance on the training and testing sets closer together as one would expect.

Rep	Seqs	Err	Err/seq Tr	Err/seq Te	% Tr	% Te
1 unit	20	0.016	0.10	1.90	90.00%	5.00%
1 unit	40	0.013	0.50	0.58	50.00%	50.00%
1 unit	80	0.025	0.50	0.50	50.00%	50.00%
1 unit	130	0.024	0.58	0.58	50.00%	50.00%
2 unit	20	0.015	0.10	1.00	90.00%	35.00%
2 unit	40	0.006	0.40	0.50	60.00%	55.00%
2 unit	80	0.008	0.18	0.26	82.50%	73.75%
2 unit	130	0.012	0.36	0.37	64.62%	63.33%

Table 4.9: The encoding and decoding performance of various 33-20-33 SRAAMs trained by Kwasny and Kalman's method to encode 20, 40, 80 and 130 sequences respectively.

Networks trained via Kwasny and Kalman's method

Table 4.9 summarises the performance of the SRAAMs trained by Kwasny and Kalman's method on 20, 40, 80 and 130 sequences and using both the 1 unit and 2 unit representations. Generally speaking the performance here is better than that of the back-propagation networks, though the networks trained on 20 sequences did not achieve perfect encoding and decoding as they did with back-propagation.

4.5.4 Analyses of the weights

Networks trained via back-propagation

The networks trained by back-propagation had 1%, 10% and 50% noise added to their weights and the encoding and decoding performance was noted. This noise was added by perturbing the value of each weight randomly up or down by the relevant percentage of its value. The results are summarised in Tables 4.10 and 4.11. As can be seen in every case, 10% noise was sufficient to seriously degrade performance from the level reached at the end of training, and 50% noise was sufficient to cause complete failure in the encoding and decoding performance.

The average, average magnitude, minimum and maximum values of the weights were calculated at the end of training for all the networks covered in Table 4.8. The results are summarised in Table 4.12. There seems to be no particular trend associated with the increasing number of sequences learned,

Rep	Seqs	Noise	Err/Seq	% Seqs
1 unit	20	0%	0.00	100%
1 unit	20	1%	0.00	100%
1 unit	20	10%	1.75	40%
1 unit	20	50%	4.70	0%
1 unit	40	0%	0.9	35%
1 unit	40	1%	0.92	32.5%
1 unit	40	10%	2.05	10%
1 unit	40	50%	4.73	0%
1 unit	80	0%	0.59	52.5%
1 unit	80	1%	0.53	56.25%
1 unit	80	10%	2.63	3.75%
1 unit	80	50%	4.58	0%
1 unit	130	0%	0.92	32.31%
1 unit	130	1%	0.9	31.54%
1 unit	130	10%	1.55	11.54%
1 unit	130	50%	4.82	0%
2 unit	20	0%	0.00	100%
2 unit	20	1%	0.00	100%
2 unit	20	10%	0.50	65%
2 unit	20	50%	4.25	0%
2 unit	40	0%	0.27	72.5%
2 unit	40	1%	0.27	72.5%
2 unit	40	10%	0.65	40%
2 unit	40	50%	4.65	0%
2 unit	80	0%	0.34	71.25%
2 unit	80	1%	0.29	72.50%
2 unit	80	10%	0.97	26.25%
2 unit	80	50%	4.61	0%
2 unit	130	0%	0.96	33.85%
2 unit	130	1%	0.98	30%
2 unit	130	10%	1.11	23.08%
2 unit	130	50%	4.45	0%

Table 4.10: Effect of adding varying amounts of noise to the weights of the networks trained by back-propagation, using sigmoidal units.

Rep	Seqs	Noise	Err/Seq	% Seqs
1 unit	20	0%	0.2	80%
1 unit	20	1%	0.25	75%
1 unit	20	10%	1.55	0%
1 unit	20	50%	4.25	0%
1 unit	40	0%	0.75	45%
1 unit	40	1%	0.775	45%
1 unit	40	10%	1	37.5%
1 unit	40	50%	3.65	0%
1 unit	80	0%	0.7	40%
1 unit	80	1%	0.775	35%
1 unit	80	10%	1.5	6.25%
1 unit	80	50%	4.34	0%
1 unit	130	0%	0.9	23.85%
1 unit	130	1%	0.88	24.62%
1 unit	130	10%	1.61	6.15%
1 unit	130	50%	4.09	0%
2 unit	20	0%	0	100%
2 unit	20	1%	0.05	95%
2 unit	20	10%	1.4	5%
2 unit	20	50%	4.4	0%
2 unit	40	0%	0.2	80%
2 unit	40	1%	0.25	75%
2 unit	40	10%	0.95	15%
2 unit	40	50%	4.02	0%
2 unit	80	0%	0.51	52.5%
2 unit	80	1%	0.54	48.75%
2 unit	80	10%	0.86	30%
2 unit	80	50%	3.90	0%
2 unit	130	0%	0.59	47.69%
2 unit	130	1%	0.55	50.77%
2 unit	130	10%	1.18	14.62%
2 unit	130	50%	4.17	0%

Table 4.11: Effect of adding varying amounts of noise to the weights of the networks trained by back-propagation, using hyperbolic tangent units.

Act Fun	Rep	Seqs	Avg Wt	Min Wt	Max Wt	Avg Mag
sigmoid	1 unit	20	-0.18	-17.90	16.32	1.51
sigmoid	1 unit	40	-0.20	-43.92	25.51	1.80
sigmoid	1 unit	80	-0.15	-10.62	11.74	1.07
sigmoid	1 unit	130	-0.13	-25.67	24.38	1.56
sigmoid	2 unit	20	-0.12	-8.33	7.20	1.09
sigmoid	2 unit	40	-0.10	-6.33	6.43	0.81
sigmoid	2 unit	80	-0.08	-7.58	7.59	0.85
sigmoid	2 unit	130	-0.09	-5.67	7.63	0.83
tanh	1 unit	20	0.01	-2.87	3.71	0.47
tanh	1 unit	40	0.01	-2.19	2.48	0.34
tanh	1 unit	80	0.02	-2.29	2.55	0.37
tanh	1 unit	130	0.01	-2.30	2.60	0.37
tanh	2 unit	20	0.01	-3.38	3.60	0.49
tanh	2 unit	40	0.01	-4.62	3.43	0.35
tanh	2 unit	80	0.02	-5.64	2.92	0.39
tanh	2 unit	130	0.01	-6.87	4.96	0.45

Table 4.12: The average, minimum and maximum value of the weights for each network trained via back-propagation.

whilst the weights for the hyperbolic tangent networks are of somewhat lower magnitude than for the networks employing sigmoidal units. The networks employing sigmoidal units and the 1 unit representation had much larger weights than the rest.

For the networks trained on 20 sequences and 130 sequences, the average, average magnitude, minimum and maximum weight was calculated at roughly one third and two thirds through training and at the end of training to see if there was a trend during training. The results are summarised in Table 4.13. Generally speaking, the weights cover a wider range of values as training goes on and the average magnitude of the weights increases too, although an exception to this trend occurs with sigmoid units trained on 130 sequences and using the 2 unit representation.

Networks trained via Kwasny and Kalman's method

As with the back-propagation networks, the networks trained via Kwasny and Kalman's method had varying amounts of noise added and the performance

Act Fun	Rep	Seqs	Iters	Avg Wt	Min Wt	Max Wt	Avg Mag
sigmoid	1 unit	20	1500	-0.14	-13.67	11.87	1.08
sigmoid	1 unit	20	3000	-0.17	-16.29	14.19	1.34
sigmoid	1 unit	130	600	-0.13	-15.71	19.68	0.90
sigmoid	1 unit	130	1200	-0.12	-25.54	21.32	1.37
sigmoid	2 unit	20	1000	-0.10	-5.71	5.99	0.84
sigmoid	2 unit	20	2000	-0.11	-6.95	6.37	0.99
sigmoid	2 unit	130	500	-0.08	-4.95	6.90	0.75
sigmoid	2 unit	130	1000	-0.00	-0.25	0.25	0.13
tanh	1 unit	20	2600	0.01	-2.31	2.36	0.32
tanh	1 unit	20	5000	0.01	-2.71	3.44	0.41
tanh	1 unit	130	100	0.01	-1.88	1.84	0.26
tanh	1 unit	130	200	0.02	-2.26	2.40	0.34
tanh	2 unit	20	1500	0.01	-3.10	3.20	0.37
tanh	2 unit	20	3000	0.01	-3.09	3.53	0.44
tanh	2 unit	130	100	0.01	-4.26	2.81	0.28
tanh	2 unit	130	200	0.01	-6.43	3.87	0.36

Table 4.13: The average, minimum and maximum value of the weights for the networks trained on 20 and 130 sequences at roughly one third and two thirds of the way through training.

was noted. Table 4.14 summarises the results. As can be seen 1% noise is sufficient to seriously degrade the encoding and decoding performance of every network compared to the performance before the noise was added and 10% noise is enough to prevent any sequences being correctly encoded and decoded. Thus the networks are more sensitive to noise than those trained by back-propagation. This may be due to the fact that the sequences are encoded in a smaller hidden layer than with back-propagation. To eliminate this possibility the analysis was performed on the 39 unit SRAAMs from Chapter 3 trained by Kwasny and Kalman's method to encode and decode 130 sequences. For the 1 unit representation with no noise, the average error per sequence was 0.5 and percentage of sequences correctly encoded and decoded was 50%. With 1% noise, the average error per sequences was 0.54 and the percentage of sequences correctly encoded was 54.62%. With 10% noise the average error was 3.1 and 0% of the sequences were correctly encoded. With 50% noise the error per sequence was 4.55 and again none of

Rep	Seqs	Noise	Err/Seq	% Seqs
1 unit	20	0%	0.1	90%
1 unit	20	1%	0.9	40%
1 unit	20	10%	3.5	0%
1 unit	20	50%	4.9	0%
1 unit	40	0%	0.5	50%
1 unit	40	1%	2.2	7.5%
1 unit	40	10%	3.9	0%
1 unit	40	50%	4.63	0%
1 unit	80	0%	0.5	50%
1 unit	80	1%	1.15	31.25%
1 unit	80	10%	3.15	0%
1 unit	80	50%	4.8	0%
1 unit	130	0%	0.58	50.0%
1 unit	130	1%	1.95	0%
1 unit	130	10%	3.91	0%
1 unit	130	50%	4.78	0%
2 unit	20	0%	0.1	90%
2 unit	20	1%	0.8	45%
2 unit	20	10%	2.45	0%
2 unit	20	50%	4.85	0%
2 unit	40	0%	0.4	60%
2 unit	40	1%	1.3	17.5%
2 unit	40	10%	3.1	0%
2 unit	40	50%	4.62	0%
2 unit	80	0%	0.18	82.5%
2 unit	80	1%	1.81	1.25%
2 unit	80	10%	3.14	0%
2 unit	80	50%	4.41	0%
2 unit	130	0%	0.36	64.61%
2 unit	130	1%	1.52	2.3%
2 unit	130	10%	3.32	0%
2 unit	130	50%	4.71	0%

Table 4.14: Effect of adding varying amounts of noise to the weights of the networks trained by Kwasny and Kalman's method.

the sequences were correctly encoded. For the 2 unit representation with no noise the average error per sequences was 0.15, and 85.38% of the sequences were correctly encoded and decoded. With 1% noise the average error per sequence rises to 0.81 and the percentage of sequences correctly encoded and decoded falls to 51.54%. With 10% noise the error per sequence rises to 2.75 and none of the sequences is encoded and decoded without error. Finally with 50% noise the error per sequence rises to 3.95 and again no sequences are encoded and decoded without error. Thus it seems that the higher sensitivity to error is a consequence of using Kwasny and Kalman's method to train the SRAAM.

Rep	Seqs	Avg Wt	Min Wt	Max Wt	Avg Mag
1 unit	20	-0.03	-3.75	4.15	0.40
1 unit	40	-0.02	-5.08	4.45	0.49
1 unit	80	-0.04	-4.93	4.49	0.49
1 unit	130	-0.02	-6.09	5.41	0.51
2 unit	20	-0.02	-3.60	4.56	0.32
2 unit	40	0.001	-4.95	5.43	0.39
2 unit	80	-0.005	-4.74	5.51	0.41
2 unit	130	-0.002	-4.40	5.44	0.41

Table 4.15: The average, minimum and maximum value of the weights for each network trained via Kwasny and Kalman's method.

Table 4.15 shows the average, minimum and maximum weights for the networks trained with Kwasny and Kalman's method. There seems to be a slight trend towards larger weights with larger numbers of sequences as evidence by the increase in the average magnitude of the weights with increasing numbers of sequences.

Table 4.16 shows the average, minimum, maximum and average magnitude of the weights for the networks trained with Kwasny and Kalman's method on 20 and 130 sequences after training to a tolerance of 0.16 and 0.04. As with back-propagation the magnitude of the weights increases with more training, as does the range of values spanned.

4.6 Discussion

The work presented above has yielded the following findings:

Rep	Seqs	Tolerance	Avg Wt	Min Wt	Max Wt	Avg Mag
1 unit	20	0.16	-0.01	-2.37	1.95	0.24
1 unit	20	0.04	-0.02	-3.21	3.11	0.32
1 unit	130	0.16	-0.02	-3.51	3.32	0.32
1 unit	130	0.04	-0.02	-4.45	4.06	0.37
2 unit	20	0.16	-0.01	-1.85	2.27	0.18
2 unit	20	0.04	-0.02	-2.68	3.40	0.26
2 unit	130	0.16	-0.01	-1.89	2.45	0.19
2 unit	130	0.04	-0.01	-2.92	3.67	0.28

Table 4.16: The average, minimum and maximum value of the weights for the networks trained on 20 and 130 sequences after training to a tolerance of 0.16 and 0.04.

- Solutions definitely exist for the SRAAMs with 20 or more hidden units.
- Using simulated annealing to train a 26-13-26 SRAAM to learn to encode and decode 60 sequences for 1.5 million iterations failed to find a solution and failed to even find an area of hyperspace from which a solution could be found. However, due to the prohibitively expensive training times only one run was performed, so the evidence is only indicative.
- With the hand-derived solutions and back-propagation, adding 10% noise to each of the weights is usually sufficient to prevent training from finding an error-free solution again, and 50% noise leads to much higher levels of error at the end of training than 10%.
- With hand-derived solutions and Kwasny and Kalman's method, an error-free solution can usually be found again after 10% noise has been added and with as much as 50% noise being added in some cases.
- With the partial solutions found by the networks whose performance is summarised in Tables 4.8 and 4.9, adding 10% noise to the weights is sufficient to significantly degrade performance in the case of the back-propagation networks and adding 1% noise is sufficient to considerably degrade performance in the case of the networks trained by Kwasny and Kalman's method. This heightened sensitivity to noise was shown to

exist even when the networks trained by Kwasny and Kalman's method were the same size as those trained by back-propagation.

- There is a trend towards the magnitude of the weights increasing with training with both training methods.

Overall, the message from these results seems to be that where the solution exists it is difficult to find (though perhaps not as difficult for Kwasny and Kalman's method), and that the partial solutions found by both methods are sensitive to noise, with Kwasny and Kalman's method producing solutions that are highly sensitive to noise. It is clear that with both methods, the partial solutions found are significantly disrupted with only small changes in the values of the weights. This suggests that the partial solutions are located in small areas of weight-space where small moves away lead to large increases in the error, and can thus be considered as evidence that the network is stuck in a local optimum. With the hand-derived solutions and back-propagation, again there seems to be a sensitivity to small changes in the values of the weights. 10% noise is usually sufficient to cause subsequent training to fail to find an error-free solution again, and 50% noise leads to much higher error at the end of training. This suggests that the hand-derived solutions can only be found from favourable starting points with back-propagation. With Kwasny and Kalman's method however error-free solutions are findable from a much wider area of weight-space, but not from the random distribution of weights around zero normally used to start training. Thus it seems that there is a rugged error landscape, where the solutions known to exist occupy small troughs in the case of back-propagation and large troughs that cannot be found from the normal starting points with Kwasny and Kalman's method. This is backed up by the poor performance of simulated annealing on a small network with only 60 sequences – none of the networks above managed to learn more than 20 sequences and they are all bigger than the network trained with simulated annealing. Here too, simulated annealing failed to find a solution or even an area of hyperspace from which back-propagation could find the solution after 1.5 million iterations of training, suggesting that the solution is difficult to find at least for networks the same size or smaller than the one used or for larger numbers of sequences. Whether the trend towards larger weights is significant is unclear since the average magnitudes involved are not very large, usually being less than 1, and also being less than 10% of the magnitude of the minimum or maximum.

In conclusion then, the analyses of the weights do give evidence that the solution is difficult to find for the back-propagation networks at least, and for both training methods the partial solutions are located in small troughs in the error landscape which may be provide part of an explanation for the difficulties experienced in training. Whether some modification of training might be able to overcome these difficulties, perhaps by changing the shape of the error landscape is an unanswered question. Also it is not clear why the error landscape is shaped in a way to make the solutions difficult to find. For example these results say nothing about whether the errors in the encoding and decoding performance are in part due to interference between the hidden-layer patterns, or due to the hidden-layer patterns being packed closely together in hyperspace. An examination of how the hidden-layer states are organised in hyperspace and how sensitive they are to noise will indicate whether such problems exist and may also suggest some modifications to the SRAAM or to the training methods, that might yield better performance. Chapter 5 presents several analyses of the hidden-layer states aimed at shedding light on whether such problems exist.

Chapter 5

Analysis of the Sequential RAAM Part 2: Analysis of the representations

5.1 Purpose

Chapter 4 presented a proof that a solution for Chalmers' experiment existed for SRAAMs of 20 or more hidden units, evidence that the solution may be difficult to find, and evidence that the partial solutions found by the SRAAMs are sensitive to noise in the weights and thus occupy small areas of weight-space themselves. Essentially the question of what the relationship between the solutions and the shape of the weight-space was addressed. Whilst this has shed some light on the possible reasons for the failure of the SRAAM to learn to encode and decode Chalmers' sentences perfectly, it only gives a partial picture of what is going on. There is also the question of how the hidden layer states produced during encoding and decoding are organised in hyperspace. This is important as it determines the nature of the representations produced, how the encoding and decoding trajectories are organised and how sensitive the solutions and partial solutions are to noise in the activations of the units of the SRAAM (as opposed to noise in the weights). For example if all the hidden layer states are packed into a small area of hyperspace, it may produce interference between the encodings and lead to a fragile representation that is sensitive to low levels of noise. The interference could then form the basis of an explanation of why the SRAAM

failed to learn its task. This chapter presents a set of analyses of the hidden-layer states aimed at finding out how they are organised, what the nature of the representations produced is, how sensitive they are to noise, how much of the available hyperspace is utilised and how the encoding and decoding trajectories are organised.

5.2 Analysing the representations and the encoding and decoding trajectories

Analyses of the representations used and the paths through hyperspace taken by the hidden-layer during encoding and decoding may yield clues as to what problems there are in learning the task. Such analyses can yield information about the nature of the representations developed, how the available hyperspace is utilised and how the encoding and decoding trajectories are organised in hyperspace. All of these may shed light on why the SRAAM failed to learn the task set in Chapter 3. For example it may be that the SRAAMs are squashing all of the encodings into a small area of hyperspace making it difficult to discriminate between them. Analyses are thus presented here to:

- Determine whether and how the representations cluster. The representations may cluster in a particular manner, if so this clustering might hinder the discriminations required for correct encoding and decoding of the structures. Given the way the SRAAM operates it may be that the representations cluster by the most recently added symbol as it has the most influence at any particular stage. If so this could make it difficult to correctly decode earlier symbols. The clustering method used here is Hierarchical Cluster Analysis (HCA), a technique that seems to have become a standard analysis performed on connectionist representations, e.g. it is used by Pollack (1990), Kwasny and Kalman (1995) and Sharkey and Sharkey (1992). In HCA, one starts with a pool of vectors, and selects the two that are closest together, to form a node which has the two vectors as children, and is then labelled with the average of the two vectors. This then replaces the two vectors in the pool, and the process repeats itself until 1 tree is left. This tree indicates how the vectors cluster. This is then usually plotted as a dendrogram, where the length of the line linking one node to its children is the same as the distance between the children. Section 5.3.1

presents the results of performing HCA on the encodings produced by the SRAAM for Chalmers' sentences.

- Determine the nature of the representations developed by the SRAAM. It may be that the representations are "fragile" in that a small amount of noise in the representations will lead to additional encoding and decoding errors. An analysis of the encodings produced by SRAAM would indicate this. For this purpose a variation of the single unit lesioning performed by (Balogh, 1994) is employed. Balogh took the encodings produced by the RAAM in Chalmers' experiment and then set each unit in turn to zero and decoded the result in order to see what errors resulted. The distribution of errors would thus indicate whether the values of specific units coded for specific information in the sequences This technique is modified here to multiplying the lesioned unit by some constant less than 1, and varying the constant to see how much change is needed to induce errors. With this modification the lesioning indicates both the sensitivity of the representation to noise and how the coding of information in the sequences is distributed across the units. To summarise; the SRAAM encoding for each sequence has each unit in turn lesioned by multiplying it by some number less than 1 and then the encoding is decoded and the errors noted. Section 5.3.2 presents the results of this analysis on SRAAMs trained by both back-propagation and Kwasny and Kalman's method.

- Determine how the encoding and decoding trajectories are organised. The trajectories taken by encoding and decoding are important. Ideally the decoding trajectories should be the reverse of the encoding trajectories, although in practice there will be a small deviation even for correct encoding and decoding. Given that the errors are occurring in the decoding process one would expect significant differences between the encoding and decoding trajectories to occur. Also it may be that the decoding process is continually attracted towards a certain area of hyperspace, thus leading to trajectories of all the sequences coming together and interfering with each other. In the literature several methods of plotting the encoding and decoding trajectories have been used. A straightforward plot of the hidden units in hyperspace was employed by Blank et al. (1992). This is only practical when there are at most three hidden units however. Another technique has been to plot

the trajectories against the top two or three principal components of the hidden layer patterns produced during encoding and decoding, as employed by Sperduti et al. (1995) and Stolcke and Wu (1992). Whilst this improves on the straightforward plotting in hyperspace, in that it allows more information to be taken into account for each dimension plotted, it potentially discards a lot of information since the top two or three principal components may not account for all or even most of the variation in the hidden layer patterns. Thus, both of these techniques suffer from flaws — the first is impractical for more than 3 units, the second discards information. The technique chosen here is to train a self-organising map (SOM) (Kohonen, 1990) on the hidden layer patterns produced during encoding and decoding and then plot the hidden layer states on the resulting map. The SOM forms a topological map of its inputs and can take into account all of the dimensions of the input vectors. When supplemented with information about the areas of hyperspace utilised during encoding and decoding, this forms a more complete method of plotting the encoding and decoding trajectories. Section 5.3.3 presents the results of this analysis.

- Determine how the clustering of the representations and the encoding and decoding trajectories evolve during training. It will also be useful to know the "direction" in which the training is going as this will indicate where the network is being pushed by the training method and whether any problems might arise as a result. To this end the analyses of the clustering of representations and the trajectories are performed at regular intervals during training. These results are presented at the end of the relevant sections — i.e. the HCA results are given at the end of Section 5.3.1.

- Determine the utilisation of hyperspace during encoding and decoding. Each unit in the hidden layer can potentially take on any value on the range (0,1) for sigmoidal units or (-1,1) for units employing the hyperbolic tangent activation function. If the range of values taken on by a unit during encoding and decoding is small (e.g. the range (0.2,0.4)) this would indicate that the network is packing the hidden layer states into a small portion of the available hyperspace and may have difficulty discriminating between sequences as a result. Section 5.3.4 presents the results of this analysis.

5.3 Results

5.3.1 Hierarchical Cluster Analysis

Networks trained by back-propagation

Hierarchical cluster analysis (HCA) was performed on the encodings of each sequence produced by all of the networks from Table 4.8 in order to find out how the networks cluster the encodings, and to see if the clustering changes at all as more and more sequences are learned. Figures 5.1 to 5.8 show the dendrograms produced by the HCA for the networks trained by back-propagation using sigmoidal activation on the 1 unit representation. Each plot has been split into the top and bottom branches of the tree and plotted with the top branch on one page and the bottom branch on the next in order to aid legibility in the plots with large numbers of sequences. The letters in each label are the first letters of each word in the sequences corresponding to that point. For example, "dilbm" would stand in for "diane is love by michael".

All of the plots seem to show clustering by whether the sentence is active or passive. However, as more sequences are learned, a secondary strategy of clustering by the final symbol of the sequence comes into play. With the 20 sequence plots there seems to be no discernible pattern within the active and passive clusters, whilst for 80 and 130 sequences there are clusters within the active and passive clusters where the last letter of the label is the same, although there are exceptions to this rule. One would expect the clustering to have one or both of these features — the active and passive distinction is significant involving different numbers of symbol in each sequence and the final symbol in a sequence is the one most recently added to its encoding and thus will have a stronger influence on the "shape" of the encoding. What this shows is that the active/passive distinction is more prominent and that the clustering by final symbol only comes into play when encoding large numbers of sequences and therefore packing more and more information into the encodings. These patterns of clustering were repeated consistently across all the networks trained by back-propagation as can be seen in the dendrograms for the rest of the networks, which are reproduced in Appendix A.

In addition to the HCA performed on the above networks, HCA was also performed on partially trained networks in the cases of the networks trained on 20 and 130 sequences in order to investigate how the clustering evolved

during training. The HCA was performed on the networks at roughly one third and two thirds through training as well as at the end. Figures 5.9 to 5.16 show the dendrograms for the HCAs performed at one third and two thirds through training for the networks trained with back-propagation using sigmoidal units and the 1 unit representation. As can be seen, the clustering strategies noted for the fully trained networks are evident at both one third and two thirds of the way through training suggesting that the strategies come into play early on in training leaving the rest of the training to improve the encoding and decoding performance under these strategies. This is a consistent finding across all the networks trained with back-propagation on 20 or 130 sequences, as can be seen from Figures A.25 to A.48 in Appendix A.

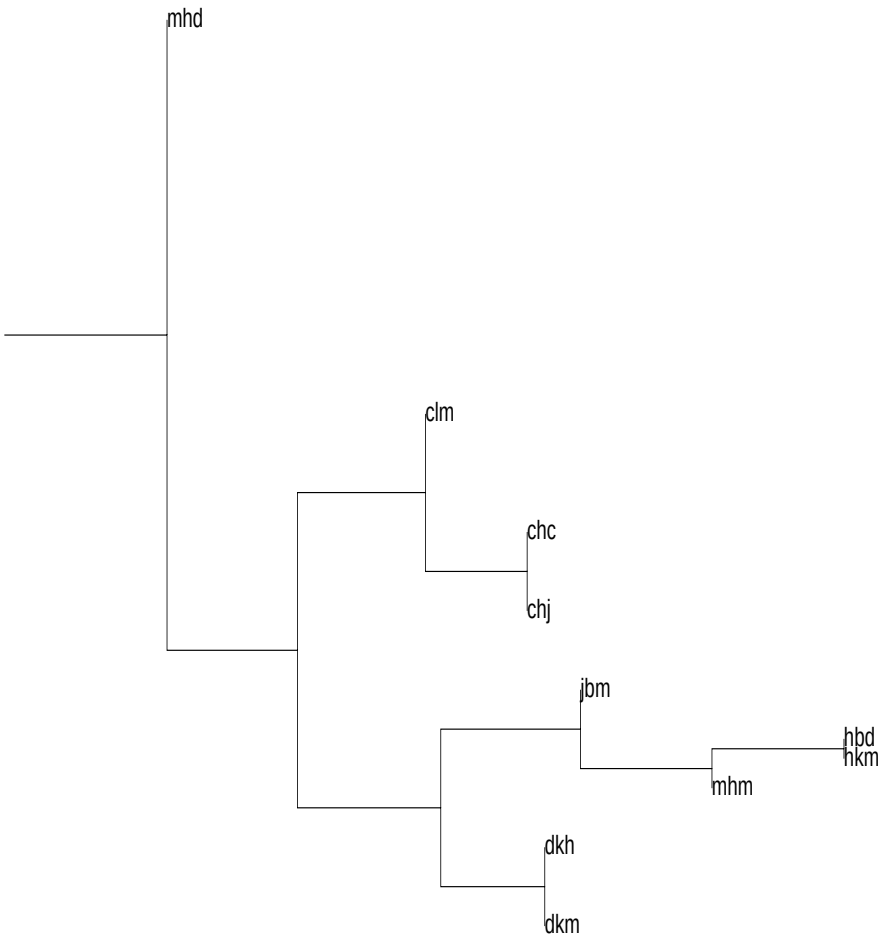

Figure 5.1: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 1 unit representation.

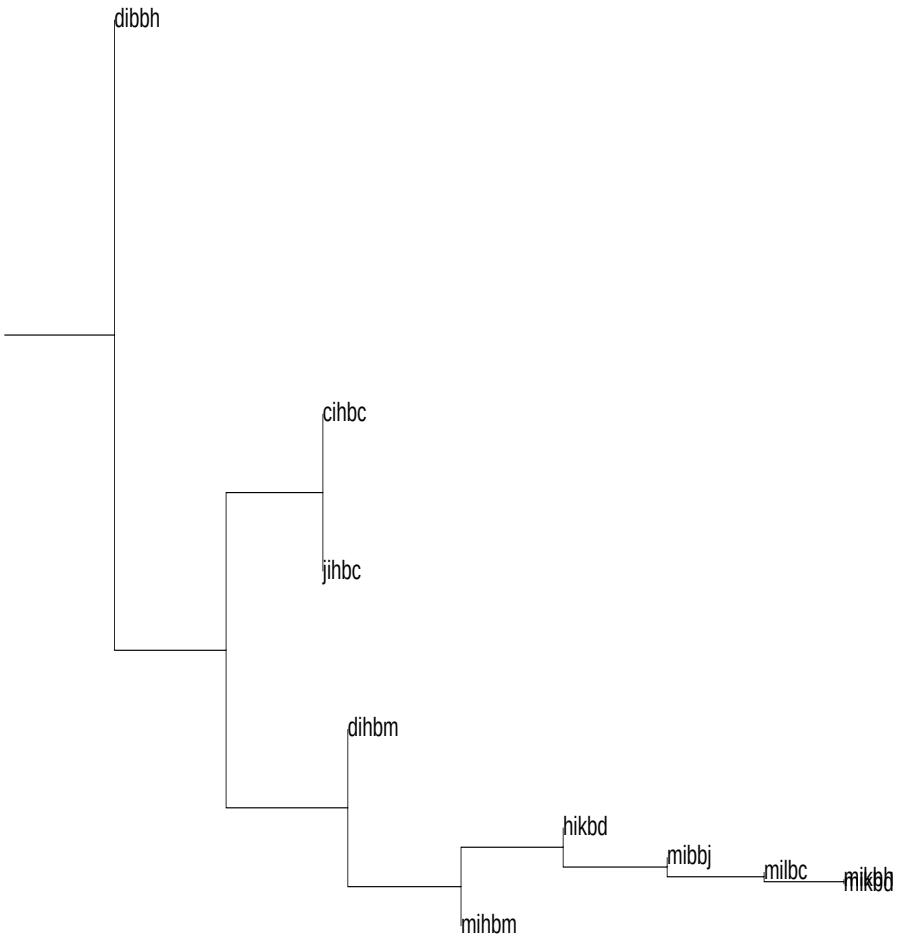

Figure 5.2: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 1 unit representation.

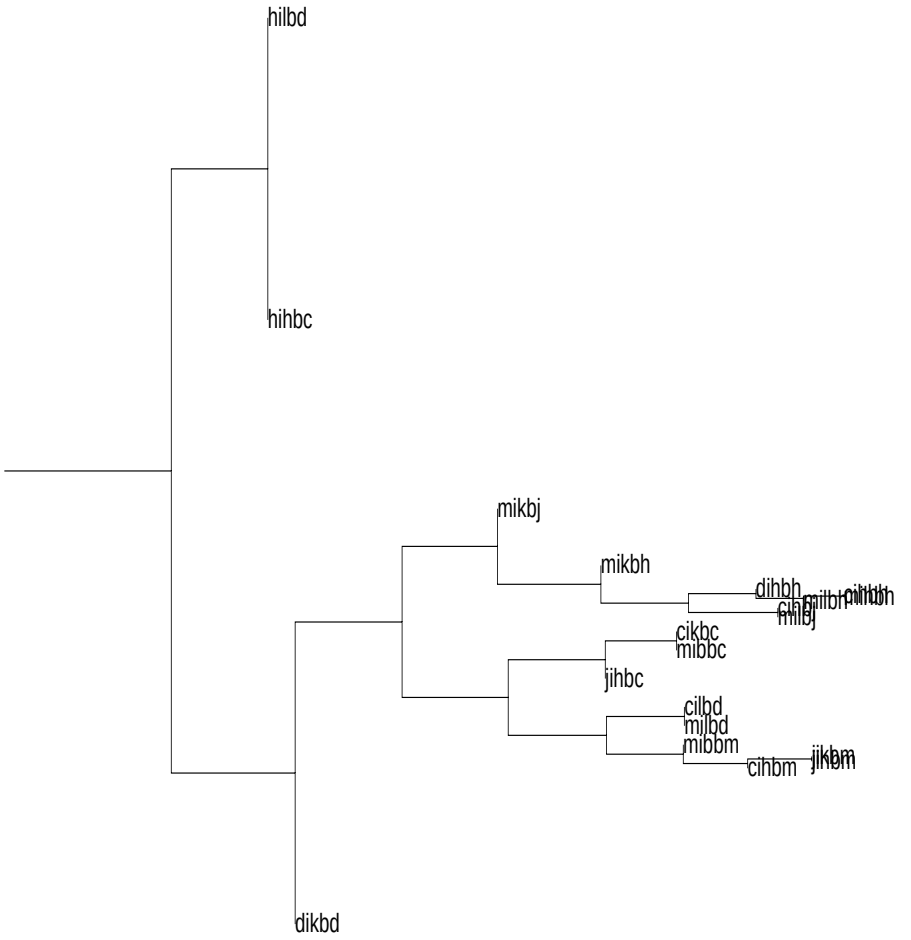

Figure 5.3: Top branch of the HCA plot for the SRAAM trained on 40 sequences with back-propagation, sigmoidal units and the 1 unit representation.

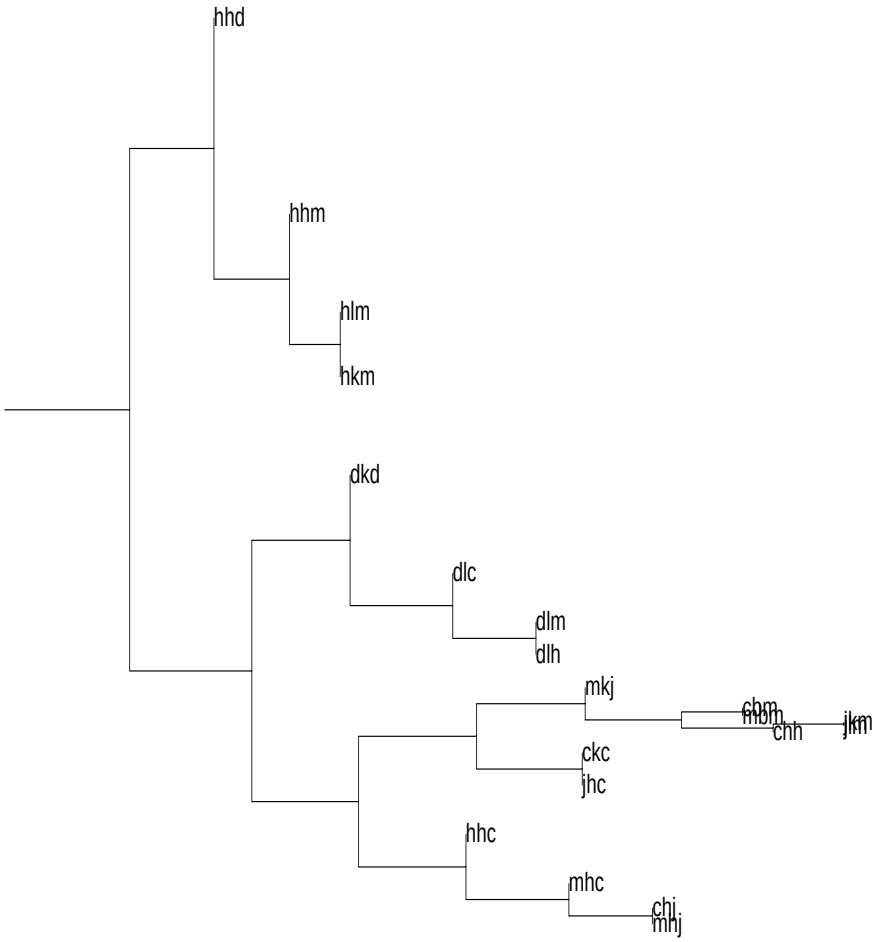

Figure 5.4: Bottom branch of the HCA plot for the SRAAM trained on 40 sequences with back-propagation, sigmoidal units and the 1 unit representations.

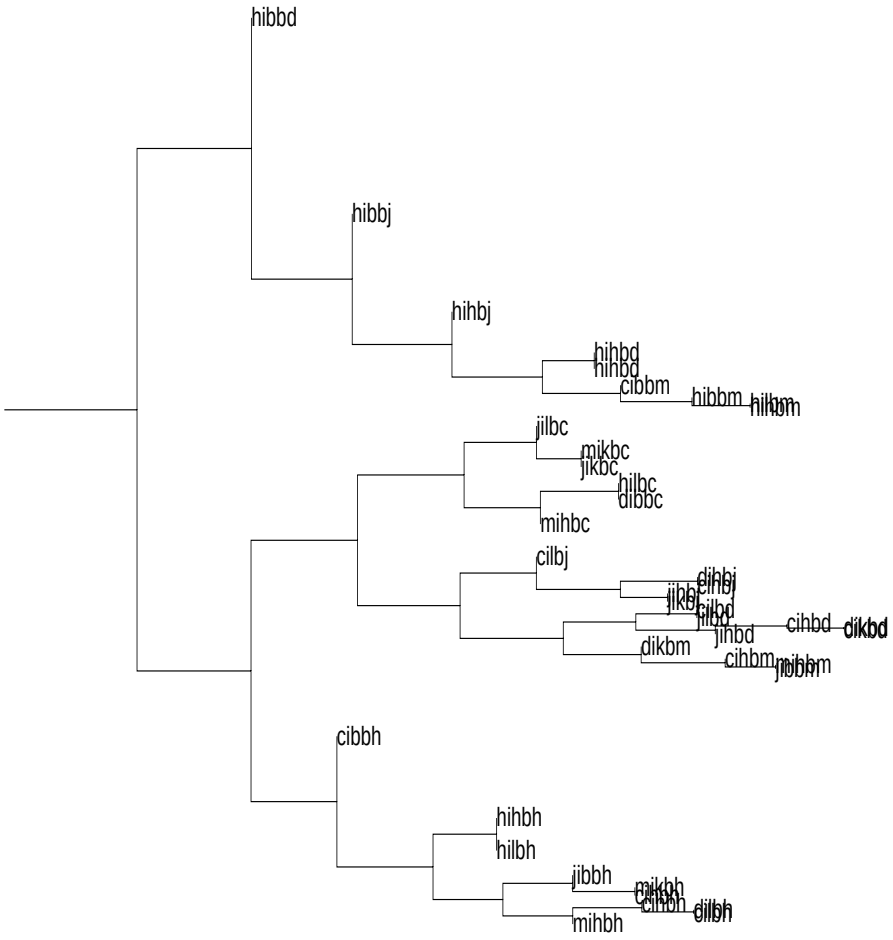

Figure 5.5: Top branch of the HCA plot for the SRAAM trained on 80 sequences with back-propagation, sigmoidal units and the 1 unit representation.

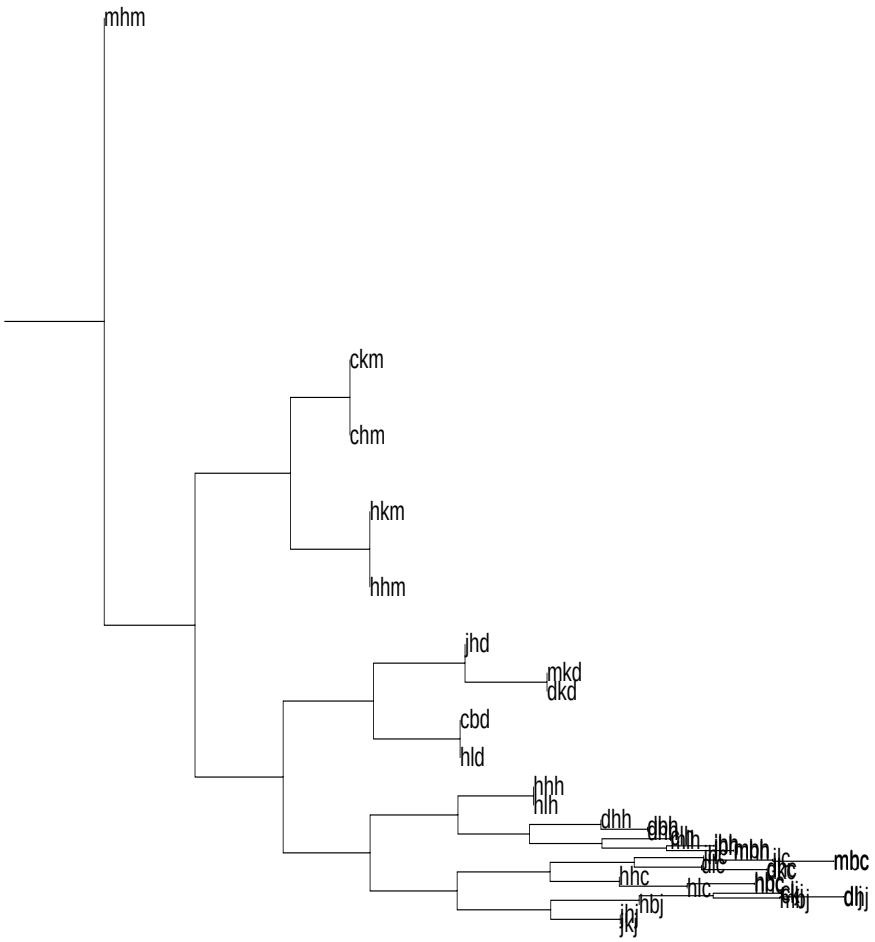

Figure 5.6: Bottom branch of the HCA plot for the SRAAM trained on 80 sequences with back-propagation, sigmoidal units and the 1 unit representation.

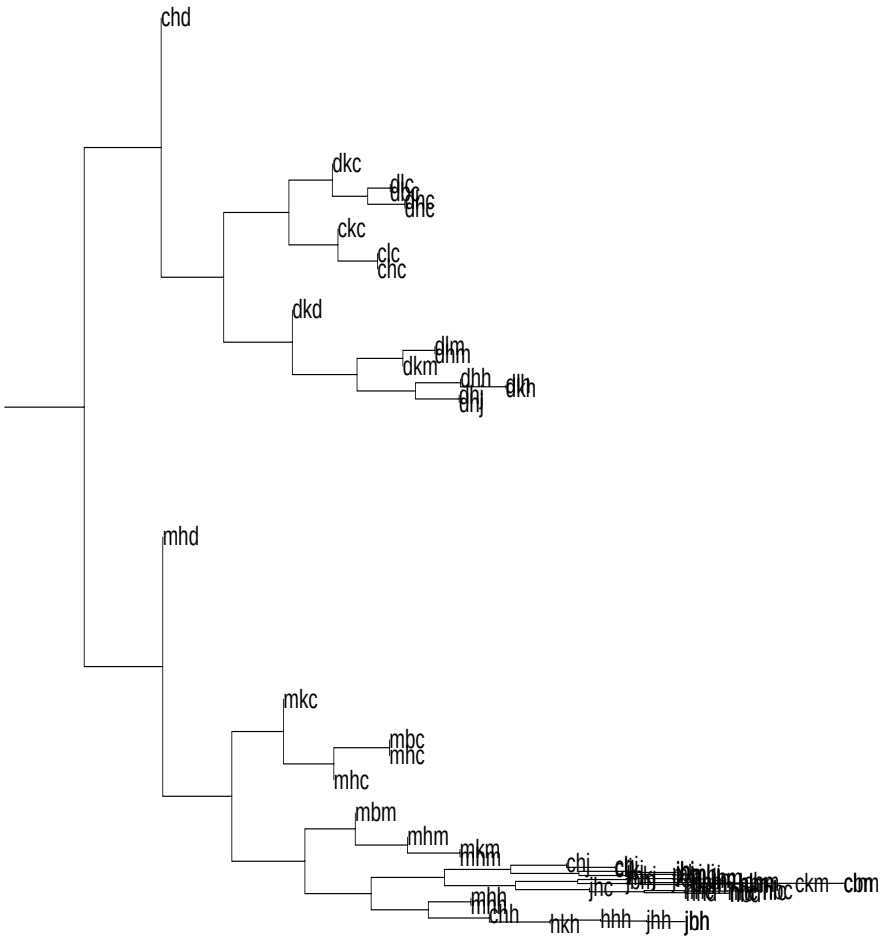

Figure 5.7: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 1 unit representation.

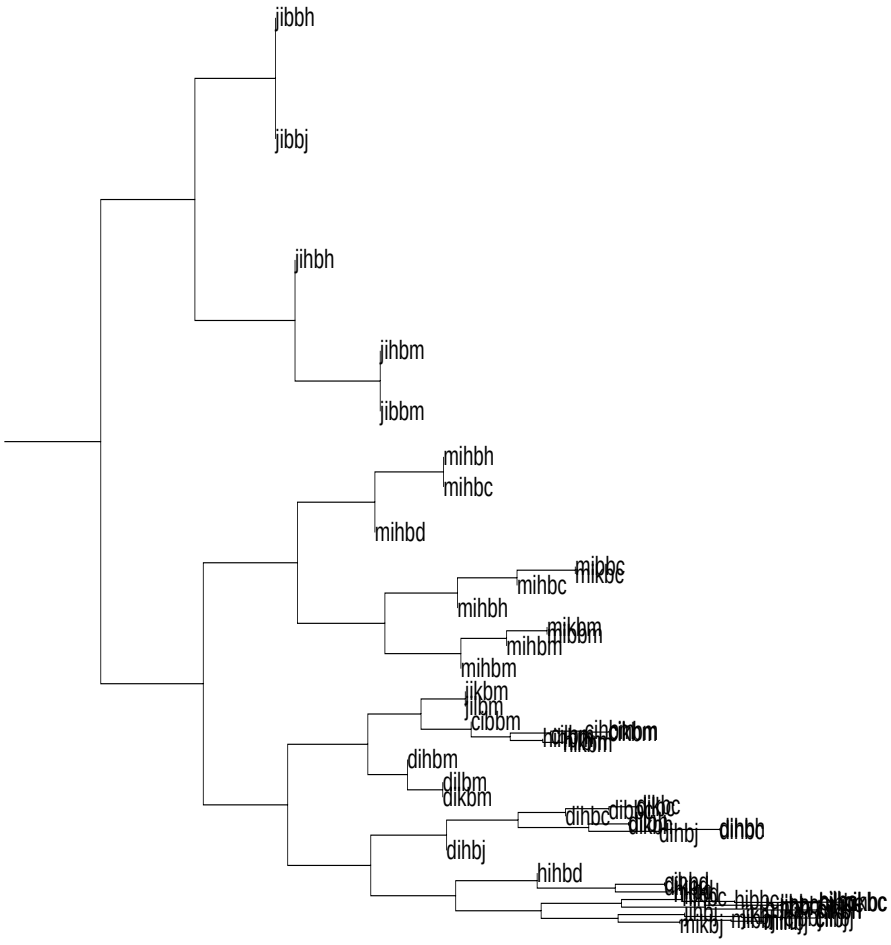

Figure 5.8: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 1 unit representation.

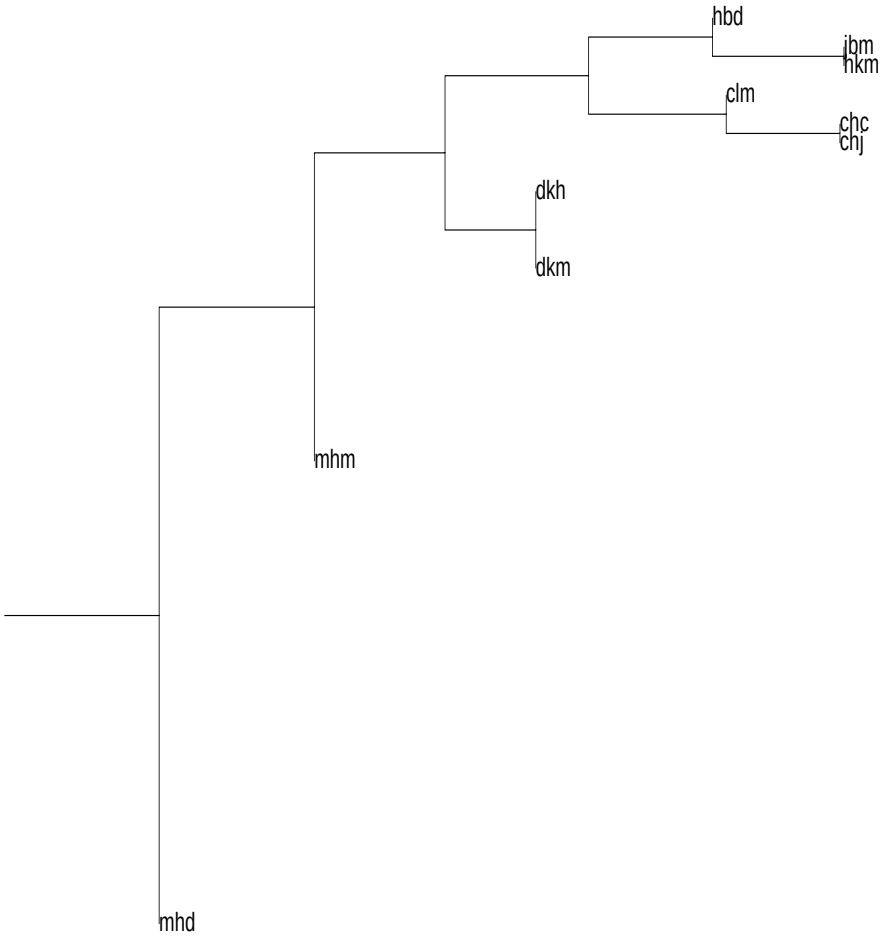

Figure 5.9: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 1 unit representation after 1500 iterations.

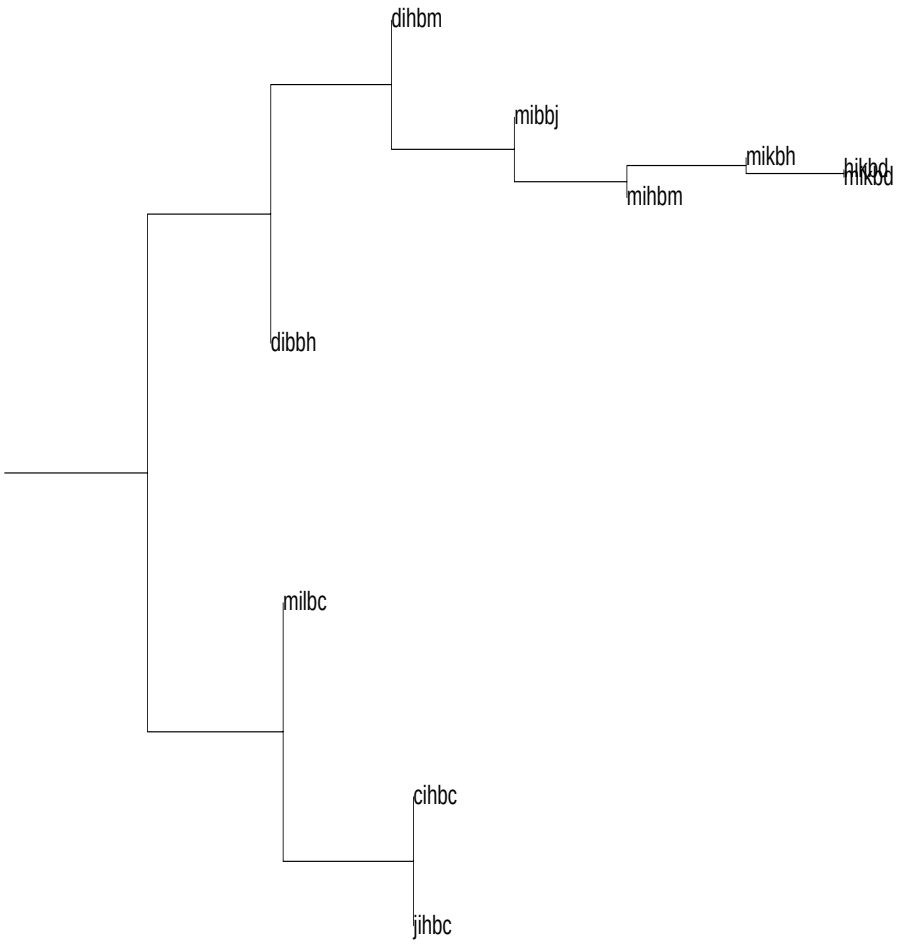

Figure 5.10: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 1 unit representation after 1500 iterations.

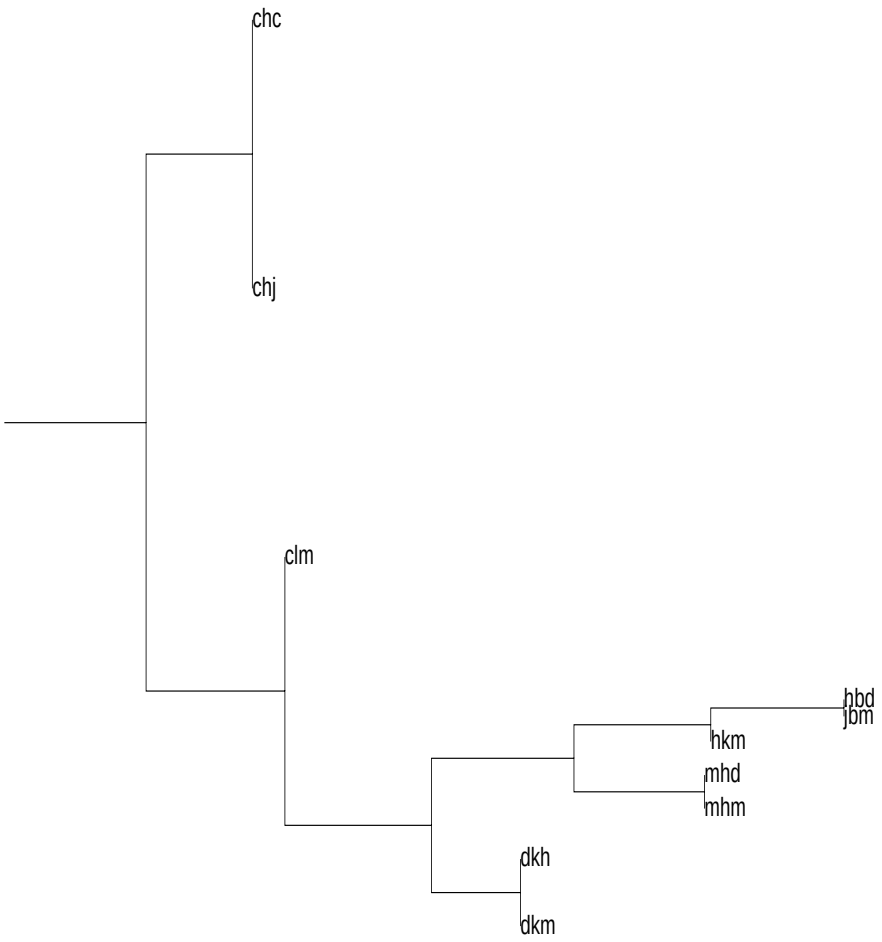

Figure 5.11: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 1 unit representation after 3000 iterations.

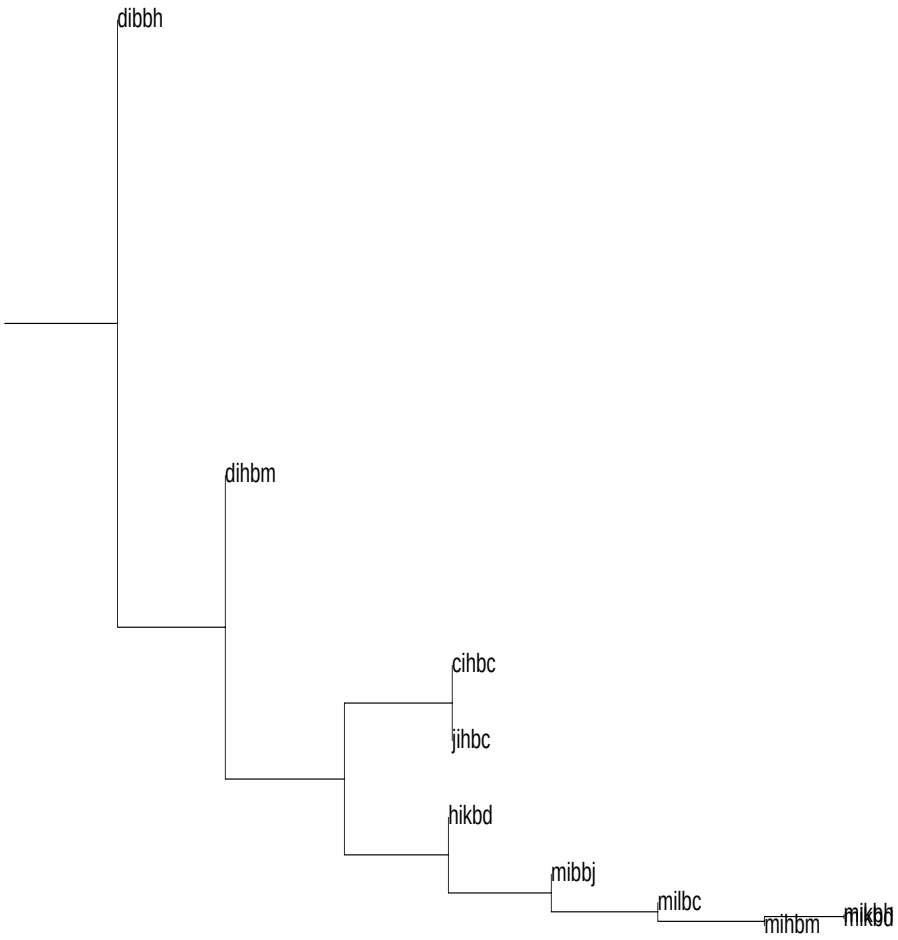

Figure 5.12: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 1 unit representation after 3000 iterations.

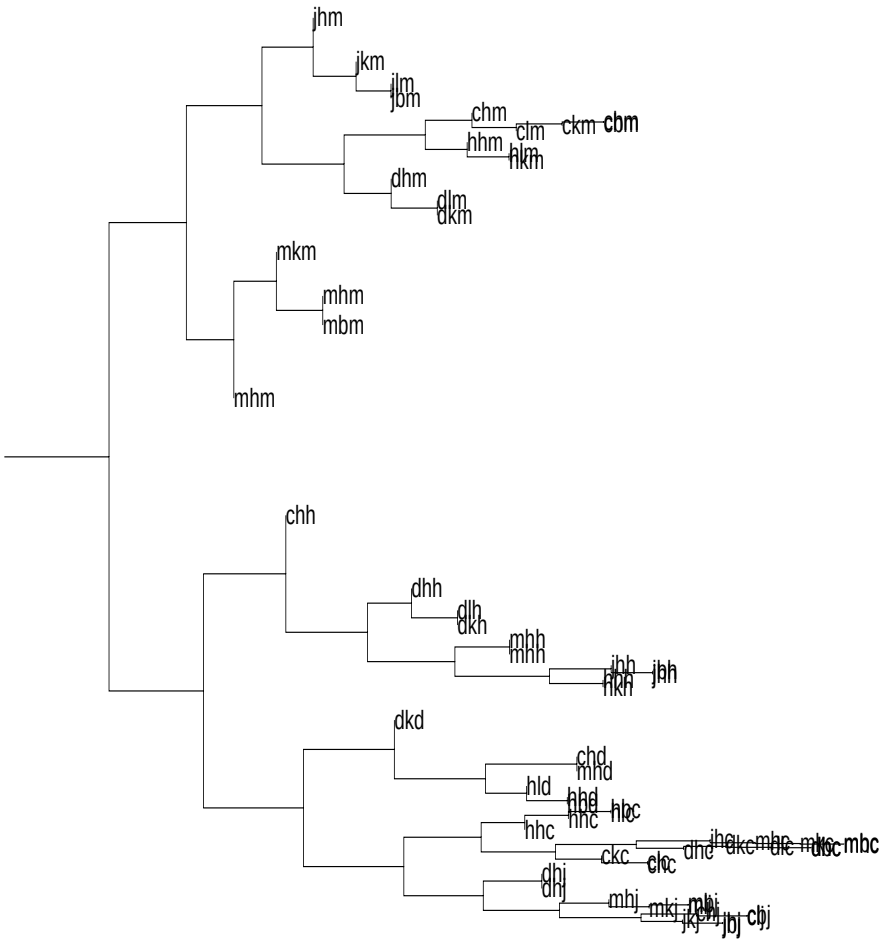

Figure 5.13: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 1 unit representation after 600 iterations.

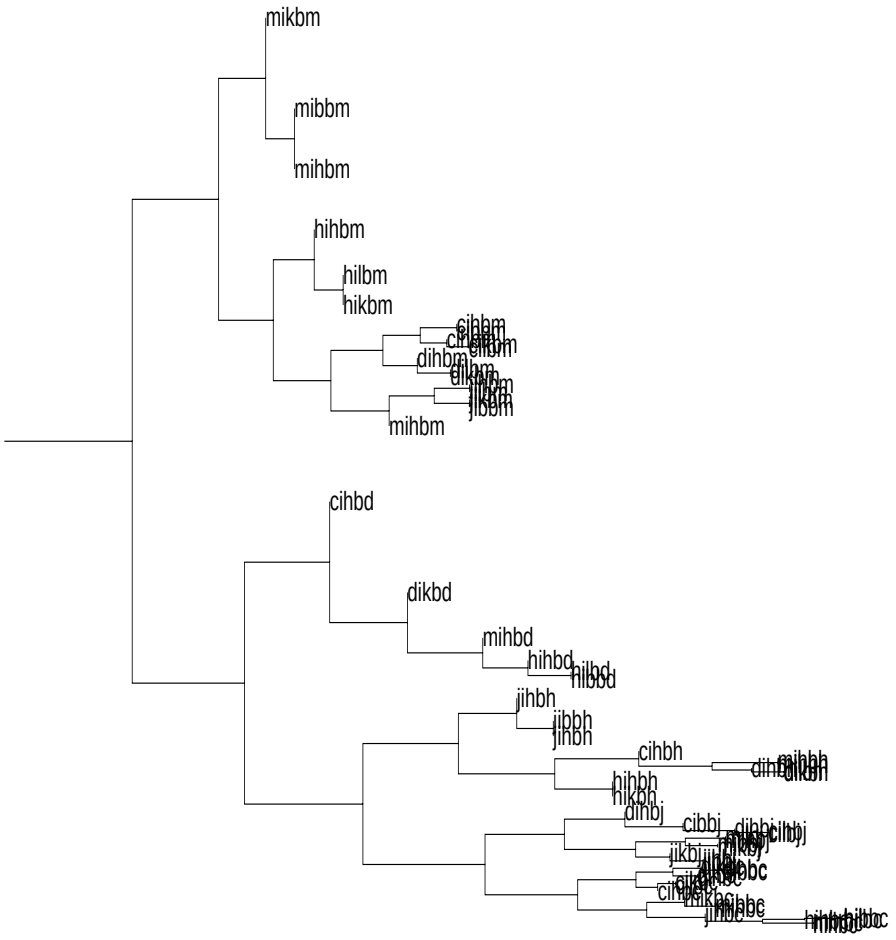

Figure 5.14: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 1 unit representation after 600 iterations.

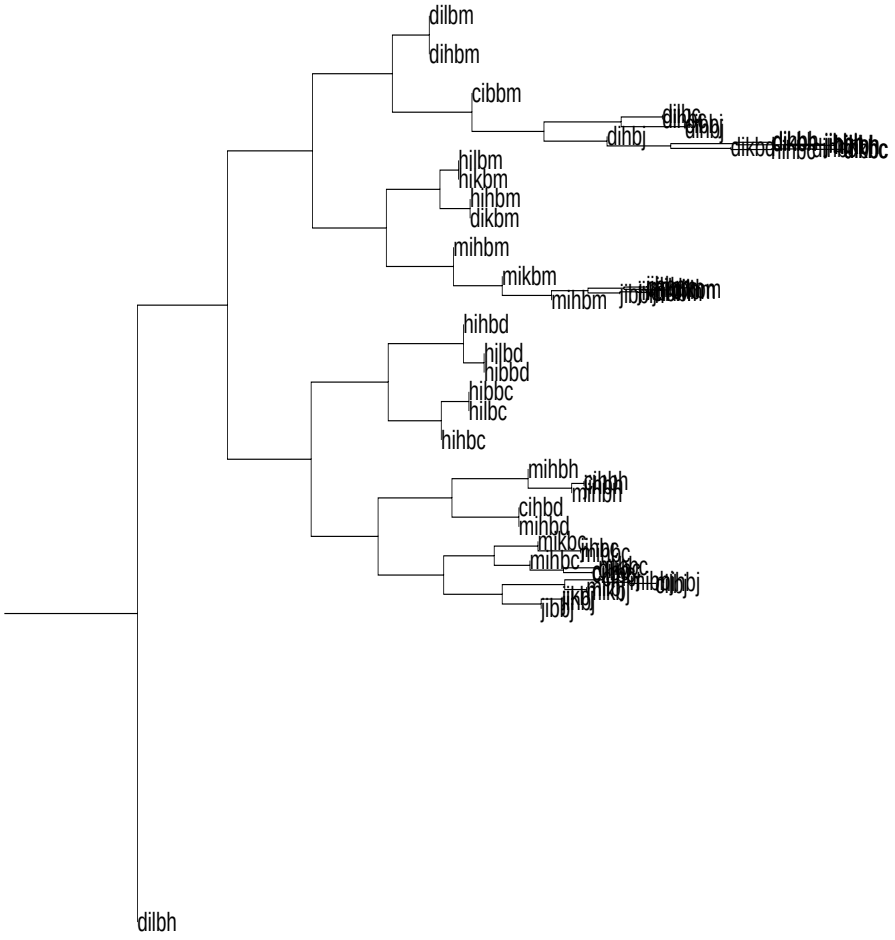

Figure 5.15: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 1 unit representation after 1200 iterations.

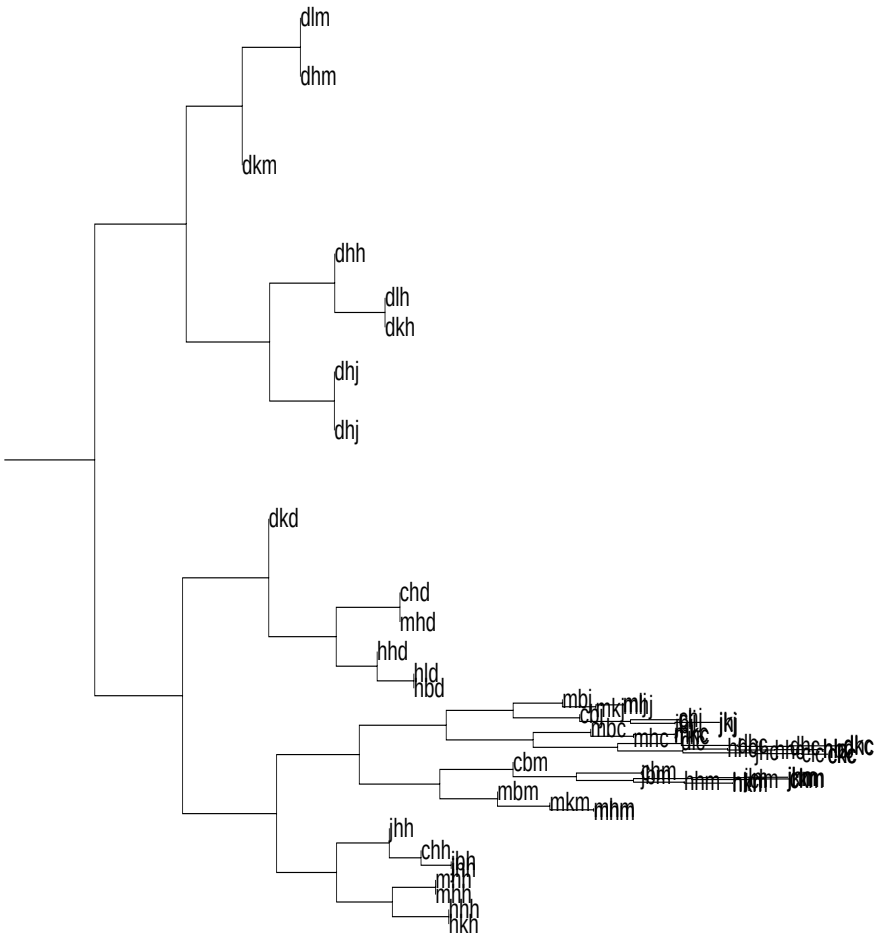

Figure 5.16: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 1 unit representation after 1200 iterations.

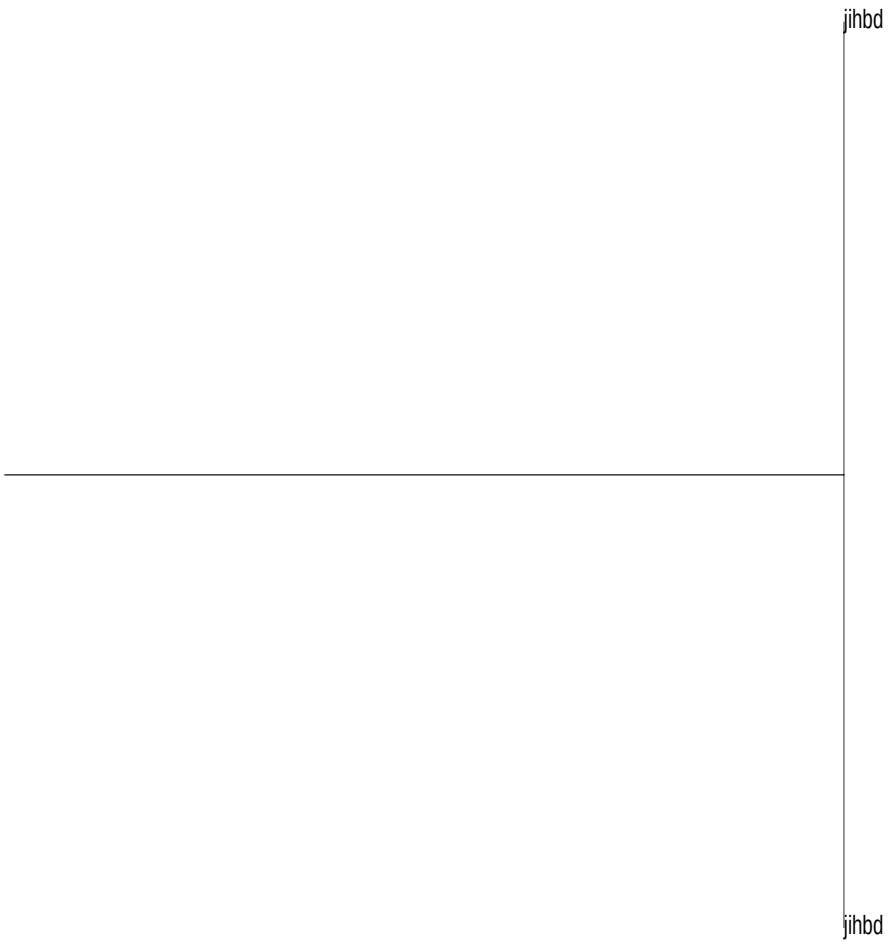

Figure 5.17: Top branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 1 unit representation.

Networks trained via Kwasny and Kalman's method

This section and Appendix A, Section A.2 presents the HCA plots for the networks trained by Kwasny and Kalman's method. The performance of these networks is summarised in Table 4.9. Figures 5.17 to 5.24 show the dendrograms for the networks trained by Kwasny and Kalman's method on the 1 unit representation. Here the pattern is somewhat different from that shown with the back-propagation training. The clustering by final symbol is much stronger and the clustering by whether the sequences is active/passive has become the secondary strategy for all the SRAAMs except the one trained on 20 sequences. This may reflect the fact that the hidden layer is only 20 units compared to the 39 units used with the back-propagation networks, resulting in the information being squeezed into a smaller vector and thus increasing the influence of the most recently added symbol in determining the "shape" of the vector for the final encoding. This pattern of clustering was repeated for the other networks trained by Kwasny and Kalman's method as can be seen in the dendrograms reproduced in Appendix A, Section A.2.

As with the back-propagation networks, HCA was also performed on partially trained networks. Due to differences between the operation of the simulators, here the HCA is performed on networks trained to a tolerance of 0.16 and 0.04 as well as at the end of training[1] (which produced the HCA plots discussed above). Figures 5.25 to 5.32 show the dendrograms for the HCAs performed on the networks trained with 20 and 130 sequences using the 1 bit representation, after training to the above tolerances. The clustering strategies mentioned above are in evidence in all of the dendrograms, again suggesting the strategy comes into effect early on in training. Again this was a consistent finding across the networks trained on 20 and 130 sequences. The dendrograms for the networks trained on the 2 bit representation are in Appendix A, Section A.2.

[1]The Kwasny and Kalman training attempts to train to a tolerance of 0.01. It usually manages 0.02.

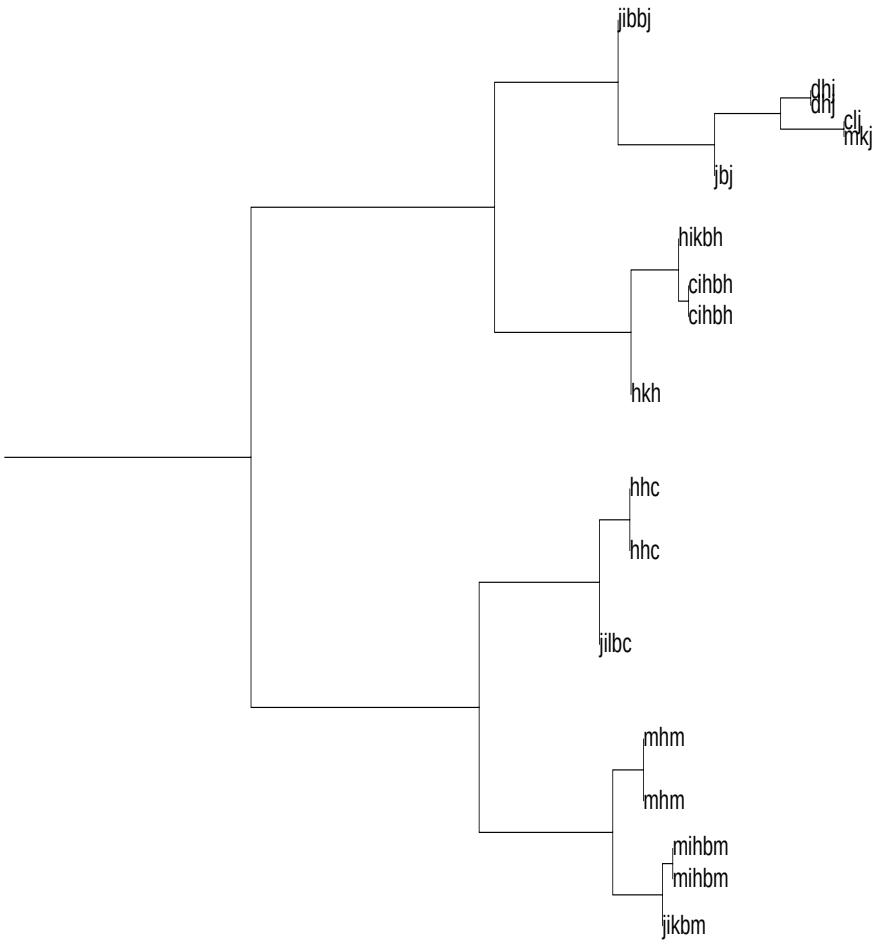

Figure 5.18: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 1 unit representation.

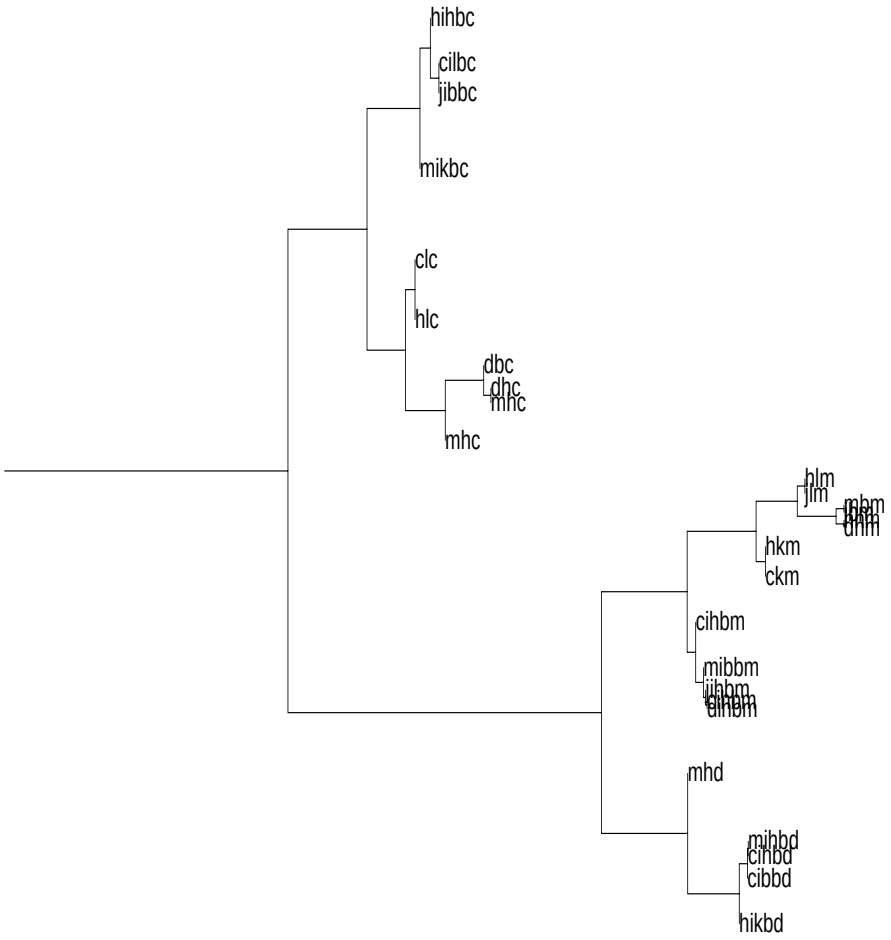

Figure 5.19: Top branch of the HCA plot for the SRAAM trained on 40 sequences with Kwasny and Kalman's method and the 1 unit representation.

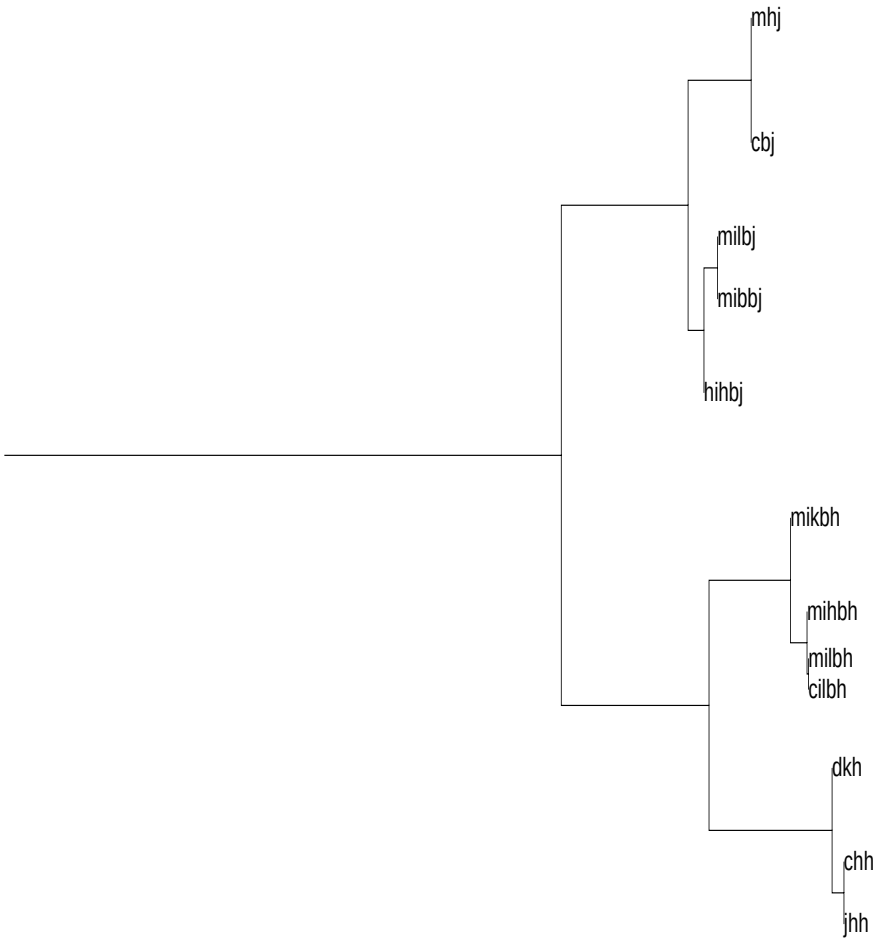

Figure 5.20: Bottom branch of the HCA plot for the SRAAM trained on 40 sequences with Kwasny and Kalman's method and the 1 unit representations.

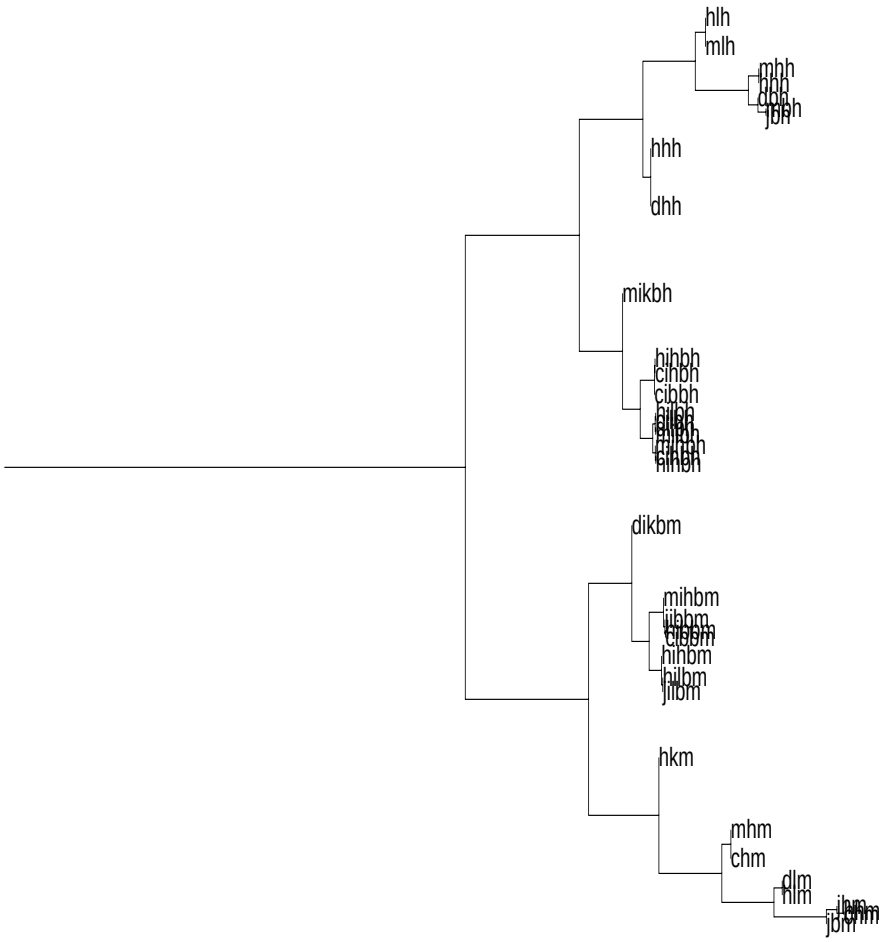

Figure 5.21: Top branch of the HCA plot for the SRAAM trained on 80 sequences with Kwasny and Kalman's method and the 1 unit representation.

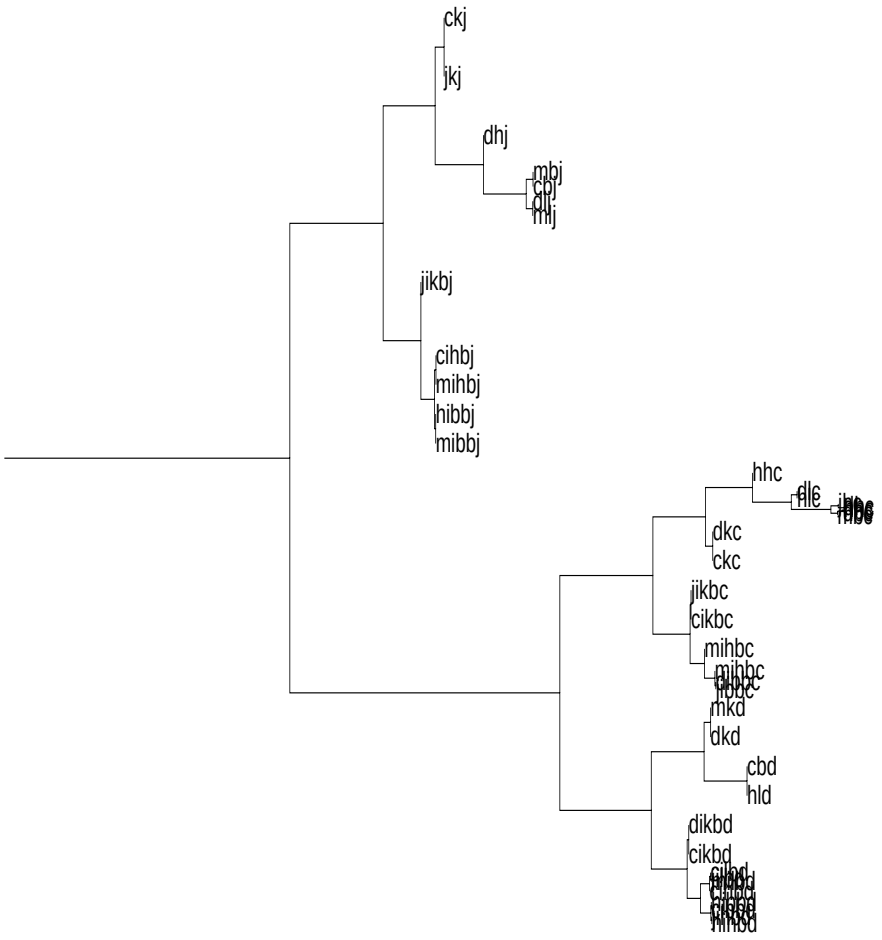

Figure 5.22: Bottom branch of the HCA plot for the SRAAM trained on 80 sequences with Kwasny and Kalman's method and the 1 unit representation.

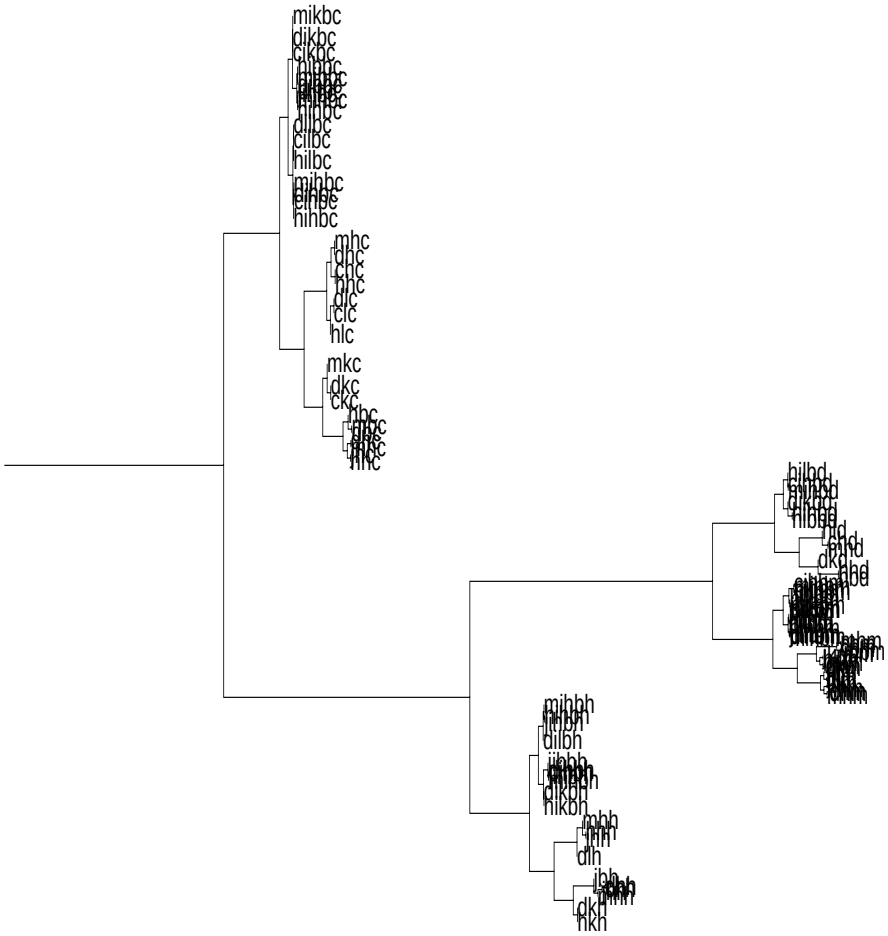

Figure 5.23: Top branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 1 unit representation.

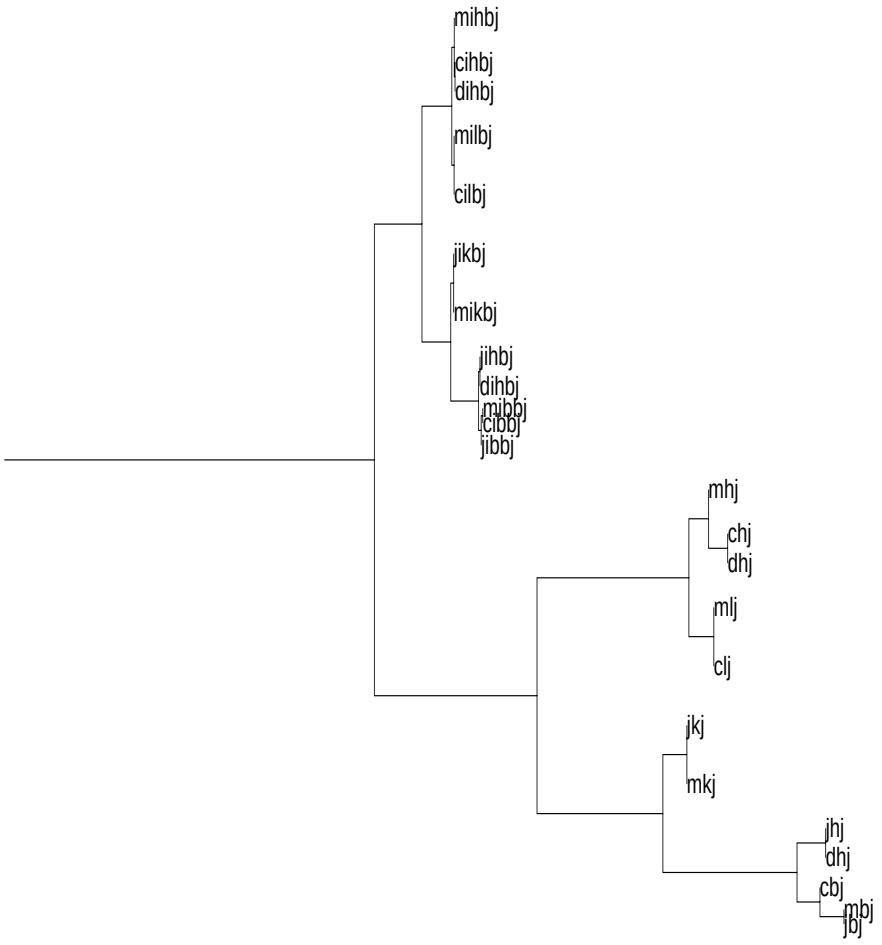

Figure 5.24: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 1 unit representation.

Figure 5.25: Top branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 1 unit representation after training to a tolerance of 0.16.

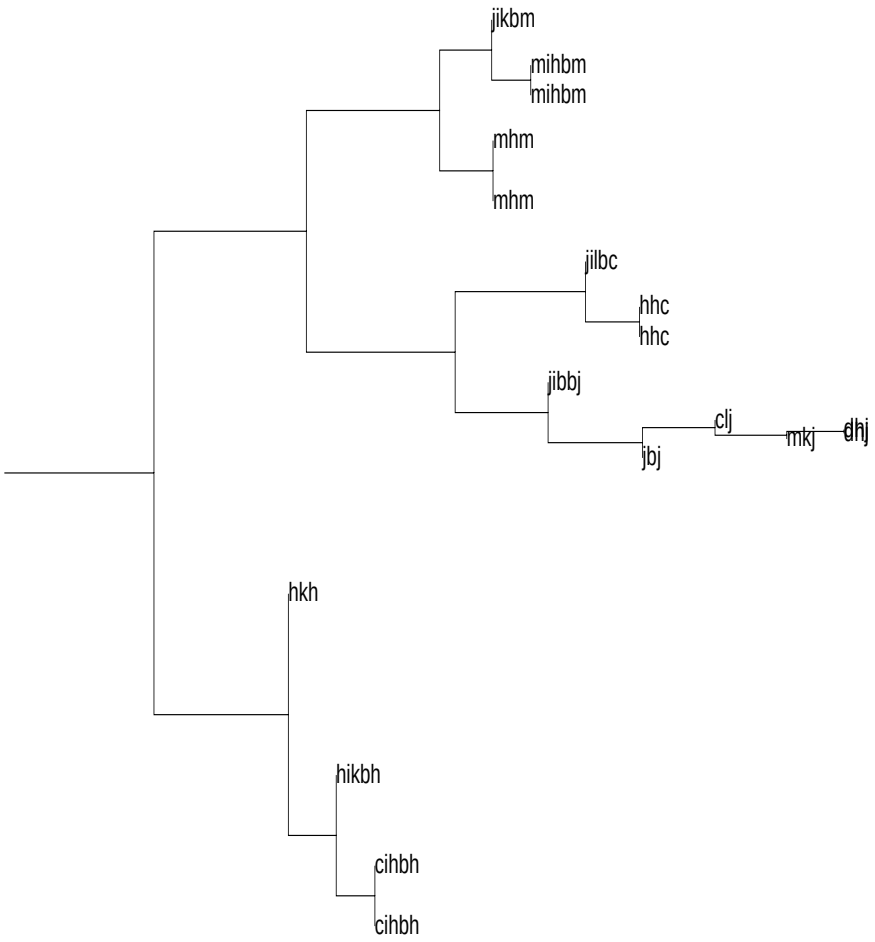

Figure 5.26: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 1 unit representation after training to a tolerance of 0.16.

Figure 5.27: Top branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 1 unit representation after training to a tolerance of 0.04.

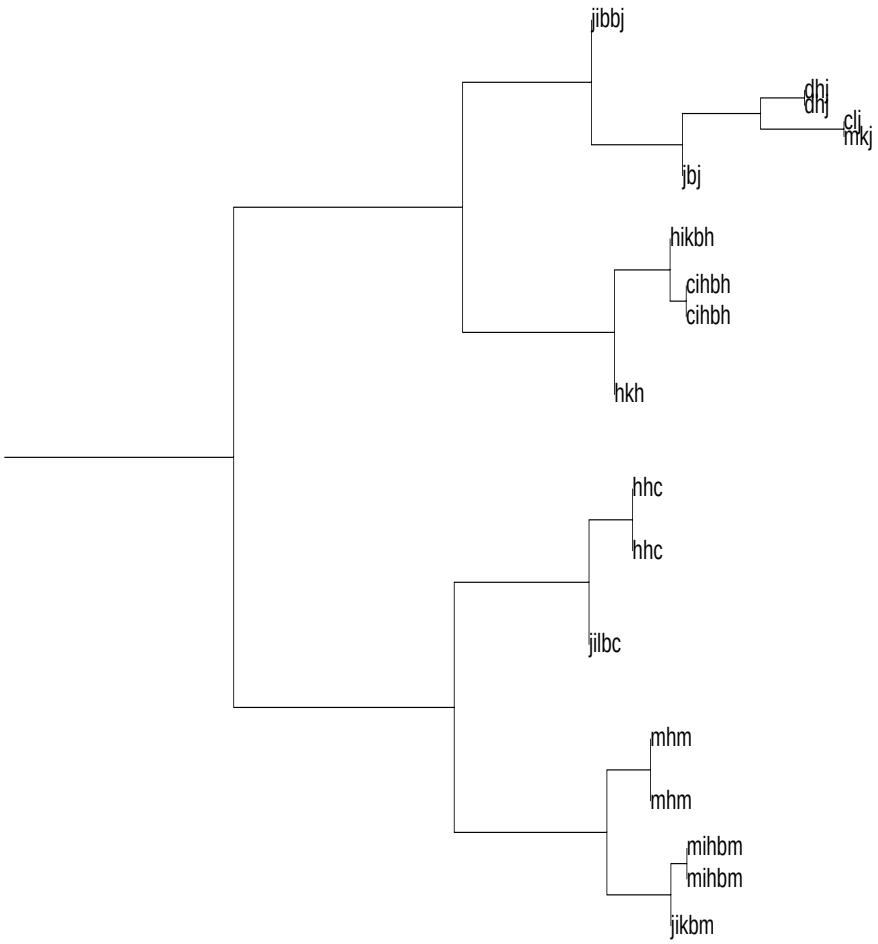

Figure 5.28: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 1 unit representation after training to a tolerance of 0.04.

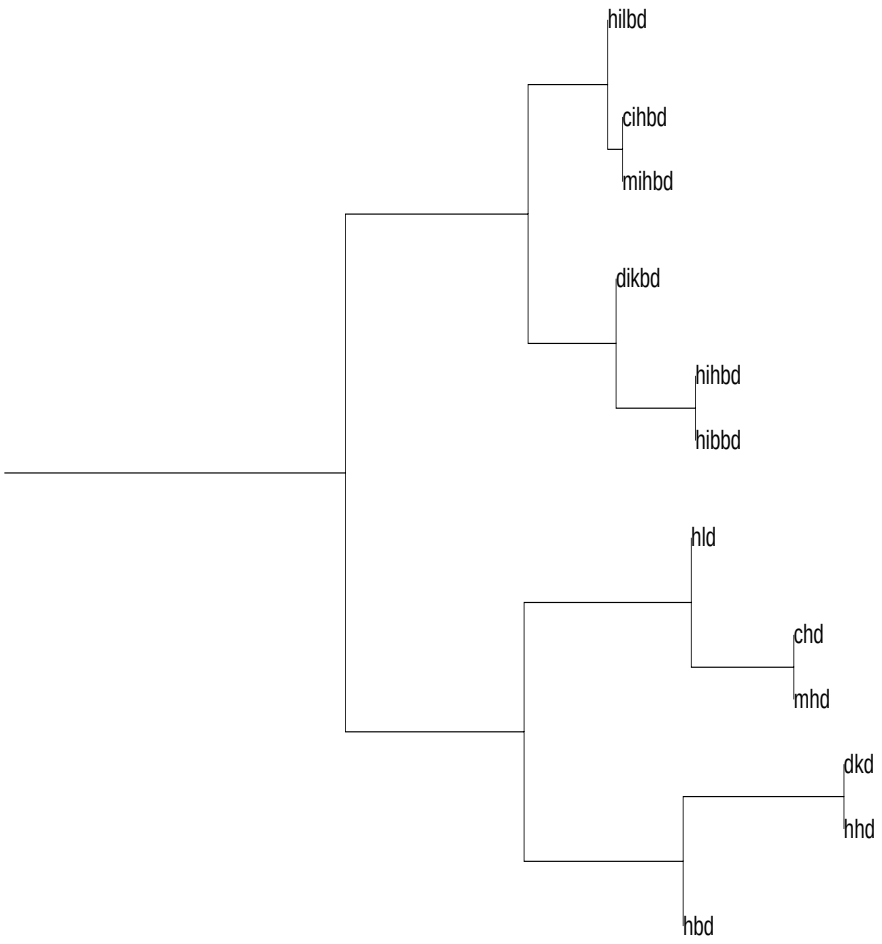

Figure 5.29: Top branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 1 unit representation after training to a tolerance of 0.16.

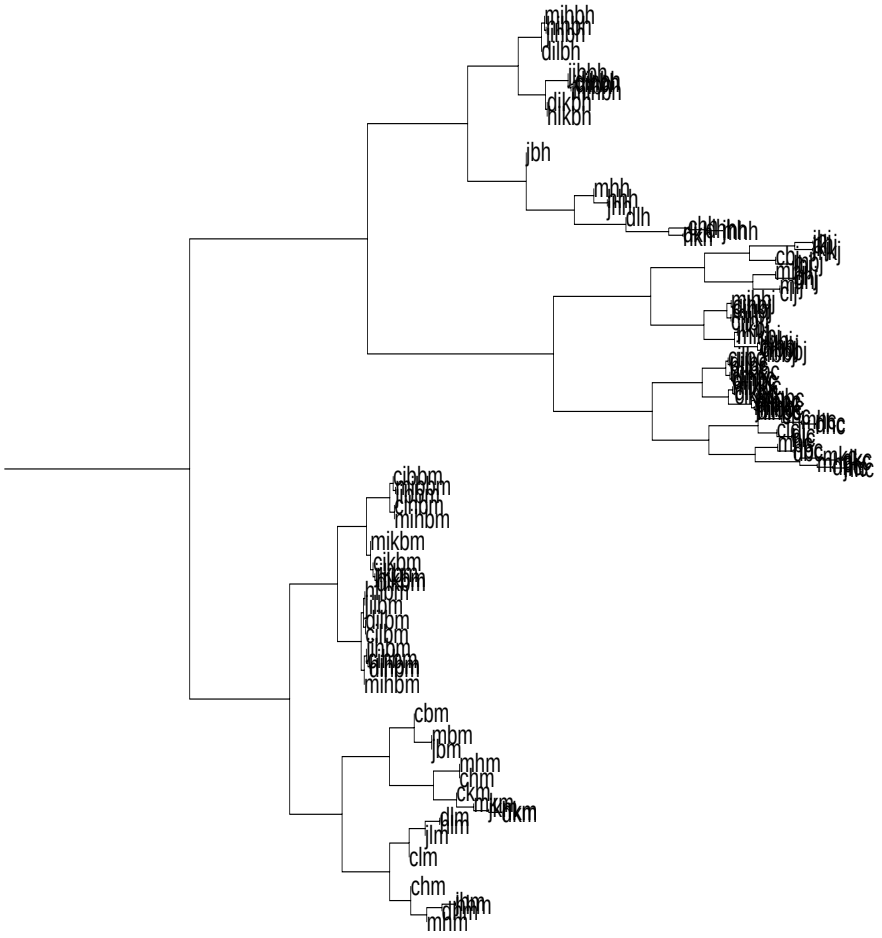

Figure 5.30: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 1 unit representation after training to a tolerance of 0.16.

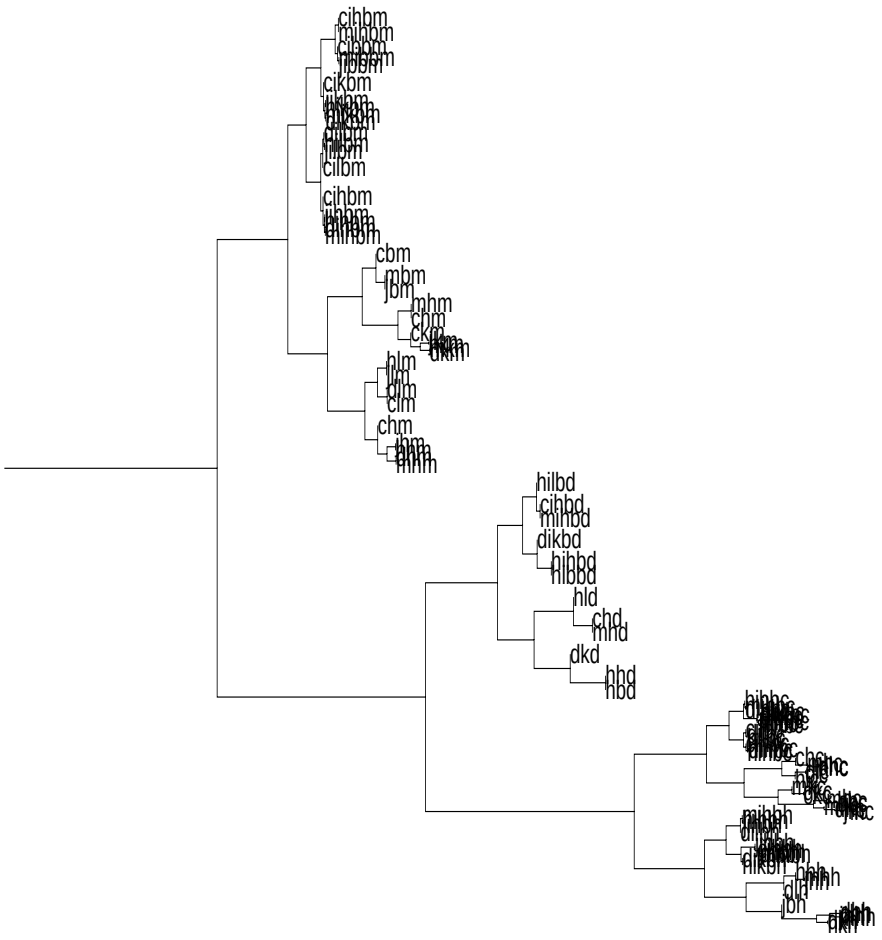

Figure 5.31: Top branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 1 unit representation after training to a tolerance of 0.04.

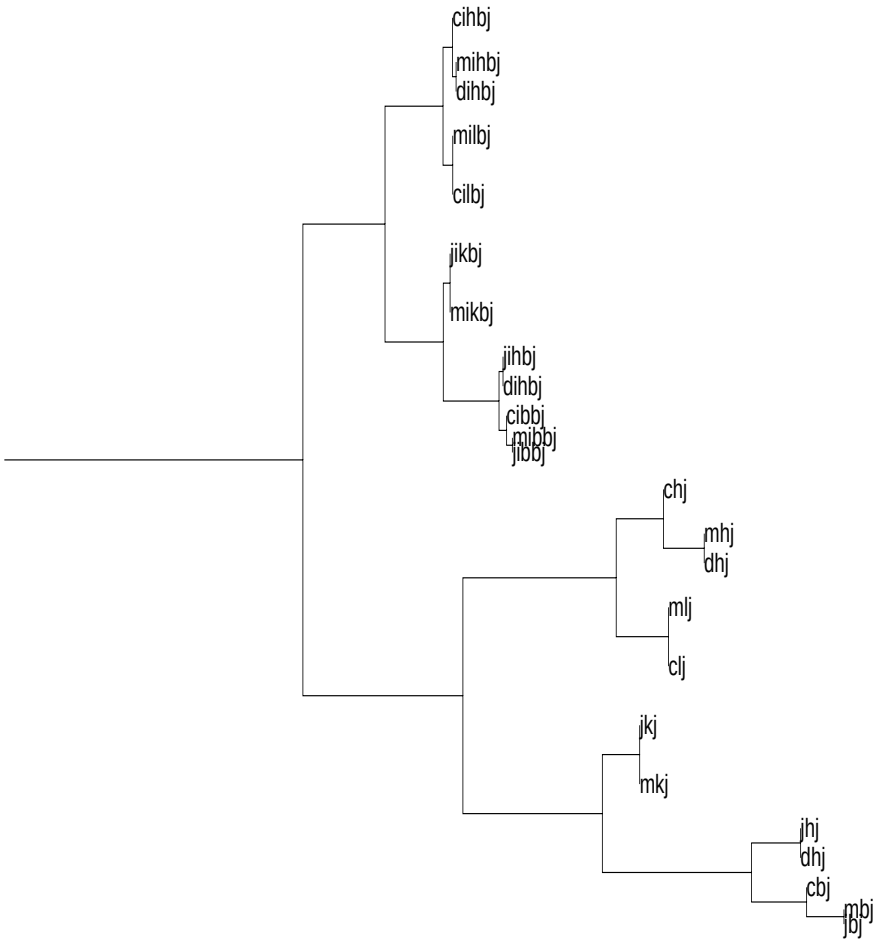

Figure 5.32: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 1 unit representation after training to a tolerance of 0.04.

5.3.2 Representational lesioning

In addition to HCA, representational lesioning was applied to the networks trained above. A single unit in the encoding for each sequence was perturbed by a percentage of its value, and the extra encoding and decoding errors induced by this perturbation noted. This was performed for each unit in turn across every encoding and then for each position in the sequence, the percentage of sequences with an error at that position was noted. Also the percentage of sequences decoded to the wrong length was noted. To provide direct comparison to Balogh's analysis, the positions of each sequence are split up into slots. For active sentences, slot 0, 1 and 2 are simply the first, second and third words of the sequences respectively. For passive sentences, slot 0 is the first word, slot 1 is the second and third words, and slot 2 is the final two words. In Balogh's thesis the slots are numbered 1 to 3 rather than 0 to 2 and corresponded to the three input and output banks of the ternary RAAM used. Here errors in the length of the sequence are also plotted under the label of "slot 3". If the sequence is of incorrect length this counts as one error. 1%, 10%, 50% and 90% perturbations were performed for each of the networks. The graphs below plot the percentage error for each slot against the unit lesioned.

Back-propagation networks

Figures 5.33 to 5.48 illustrate the results of lesioning the representations developed by the SRAAMs trained via back-propagation on the 1 unit representation using sigmoidal units. The following observations can be made of these graphs:

- There is a greater sensitivity to noise with the larger numbers of sequences learned as evidenced by the increased error levels with increasing numbers of sequences.
- The error levels for a particular slot are higher the more deeply embedded it is in the encoding, i.e. slot 0 tends to have higher error than slot 1, and slot 1 tends to have higher error than slot 2. Slot 3, which stands for the length of the sequence, tends to have similar error levels to slot 0 or 1.
- Where the SRAAM has learned 80 or more sequences, a 1% perturbation of a single unit can be enough to induce extra errors in encoding

and decoding and 10% perturbation of a single unit is often enough to cause errors in more than one slot.

- Perturbing the value of a single unit by 50% or more is usually enough to cause errors across all the slots, and this holds for most units.

Taken together, these observations suggest that the SRAAM is developing a distributed representation (though not perfectly distributed as evidenced by the uneven levels of error across the units for each slot) that is highly sensitive to error — 1% error on 1 unit being enough to cause errors in encoding and decoding to occur when 80 sequences or more have been learned. Similar results were found with sigmoidal units and the 2 unit representation as can be seen in Appendix B, Section B.1.1, with the exception that slot 3 has much lower levels of error with the 2 unit representation than with the 1 unit representation suggesting that this information is encoded more robustly than with the 1 unit representation.

Figures 5.49 to 5.56 show the results of the lesioning on encodings developed by SRAAMs trained on the 1 unit representation using hyperbolic tangent units for 20 and 130 sequences. The plots for 40 and 80 sequences can be found in Appendix B, Section B.1.2. With the hyperbolic tangent units there are significant differences compared to the sigmoidal units. Whilst it is still the case that there is greater sensitivity to error with the greater number of sequences learned, and for slots 0, 1 and 2 the sensitivity to error is higher for the more deeply embedded slot, the error levels for slot 3, the length of the sequence, are *much* higher than they are with the sigmoidal units and much higher than the levels for the other slots, suggesting this information is encoded in a very fragile manner. A 1% error on a single unit is enough to cause errors in the length, even when everything else is unaffected, with the exception of the network trained on 40 sequences. Even here, the length shows higher levels of error than the other slots. Similar results were obtained for the SRAAMs trained on the 2 unit representation and hyperbolic tangent units as can be seen in Appendix B, Section B.1.2.

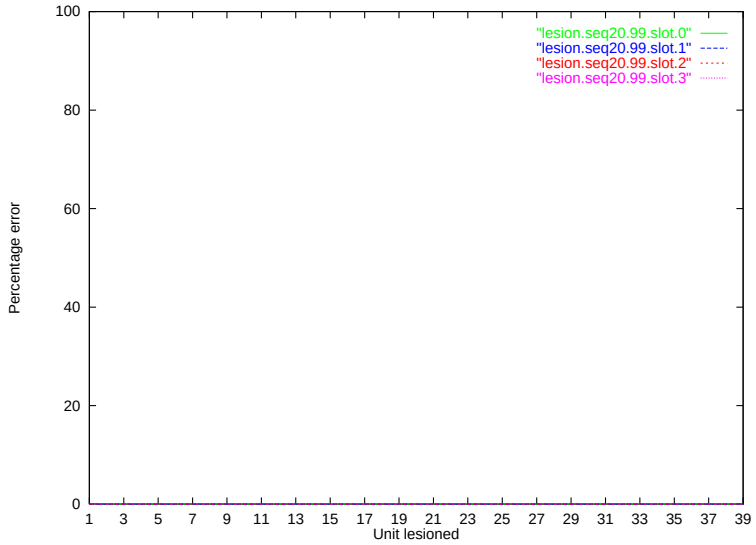

Figure 5.33: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 20 sequences and the 1 unit representation, using 1% noise. No errors were produced.

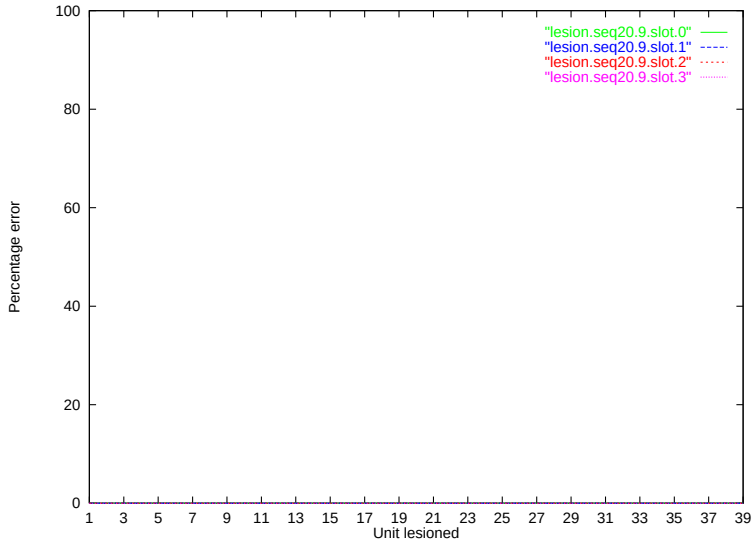

Figure 5.34: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 20 sequences and the 1 unit representation, using 10% noise. No errors were produced.

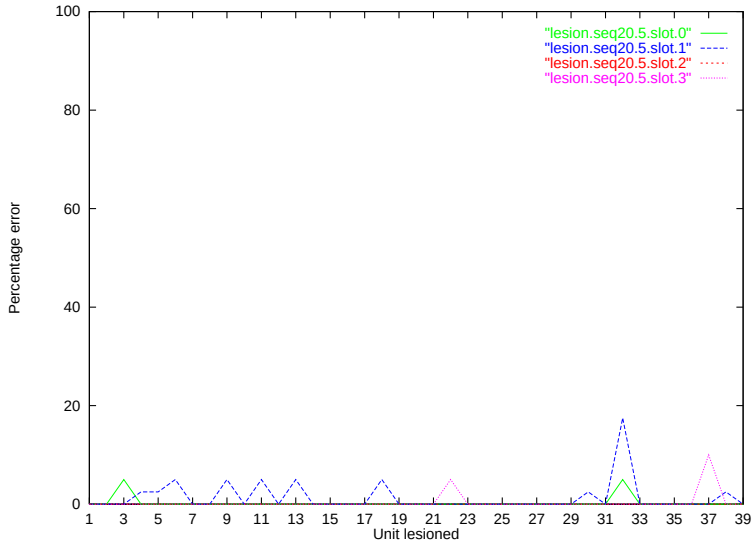

Figure 5.35: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 20 sequences and the 1 unit representation, using 50% noise.

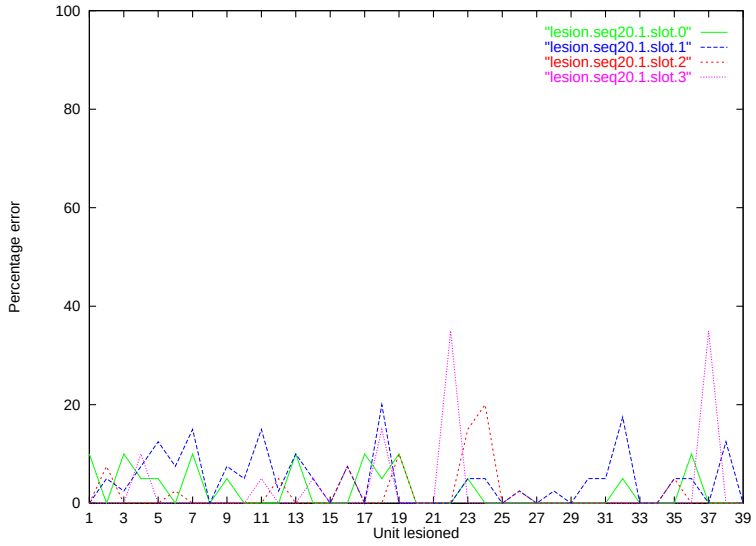

Figure 5.36: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 20 sequences and the 1 unit representation, using 90% noise.

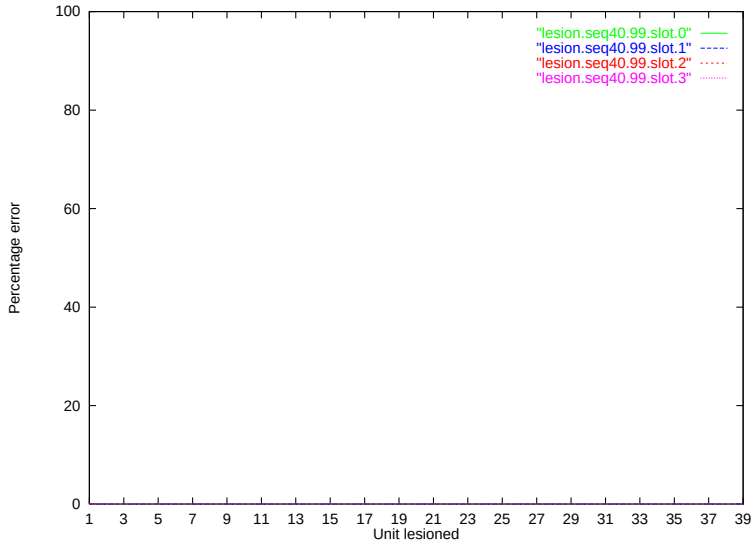

Figure 5.37: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 40 sequences and the 1 unit representation, using 1% noise. No errors were produced.

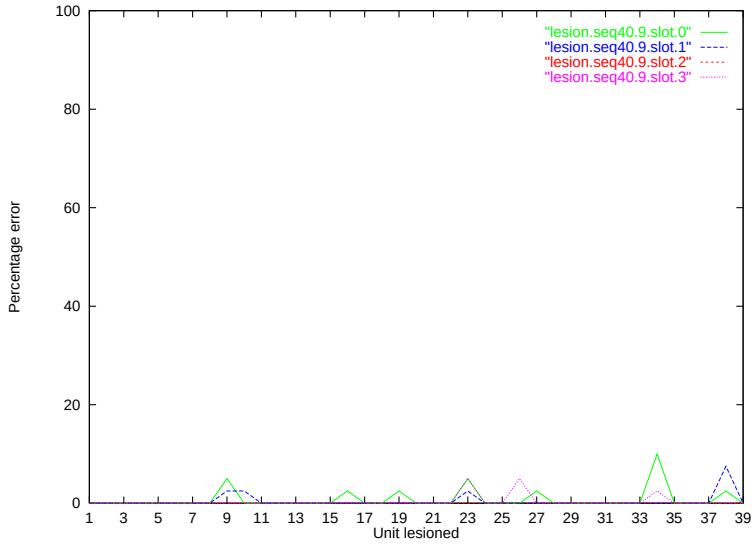

Figure 5.38: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 40 sequences and the 1 unit representation, using 10% noise.

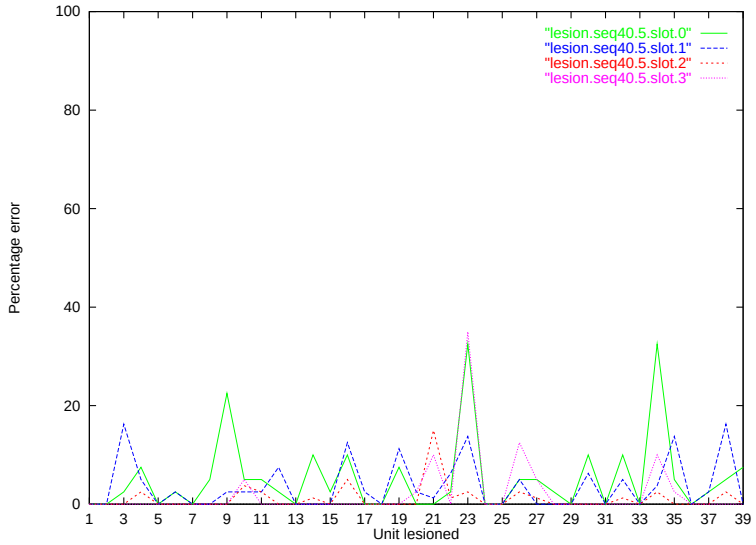

Figure 5.39: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 40 sequences and the 1 unit representation, using 50% noise.

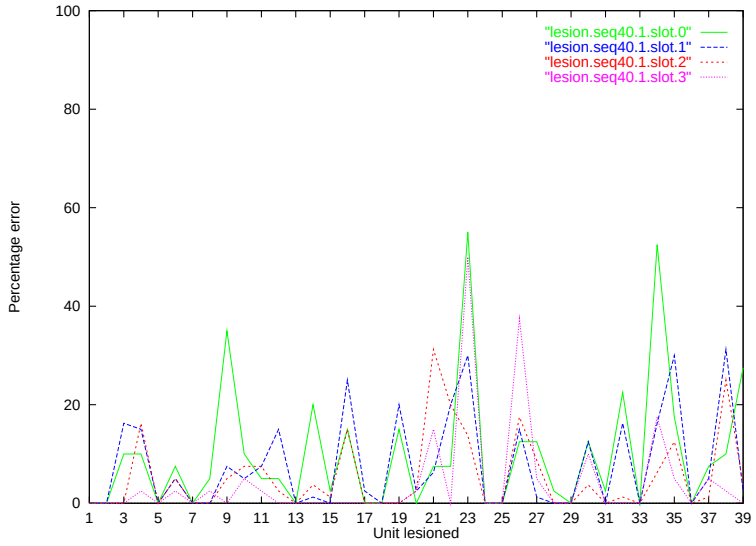

Figure 5.40: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 40 sequences and the 1 unit representation, using 90% noise.

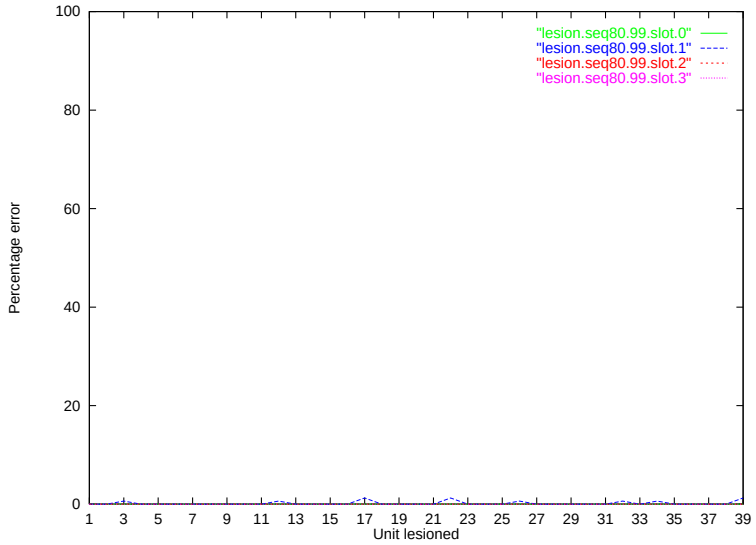

Figure 5.41: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 80 sequences and the 1 unit representation, using 1% noise.

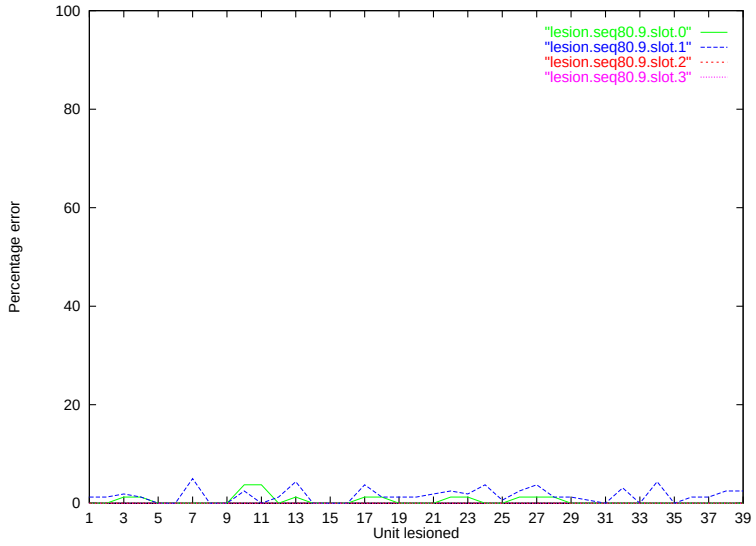

Figure 5.42: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 80 sequences and the 1 unit representation, using 10% noise.

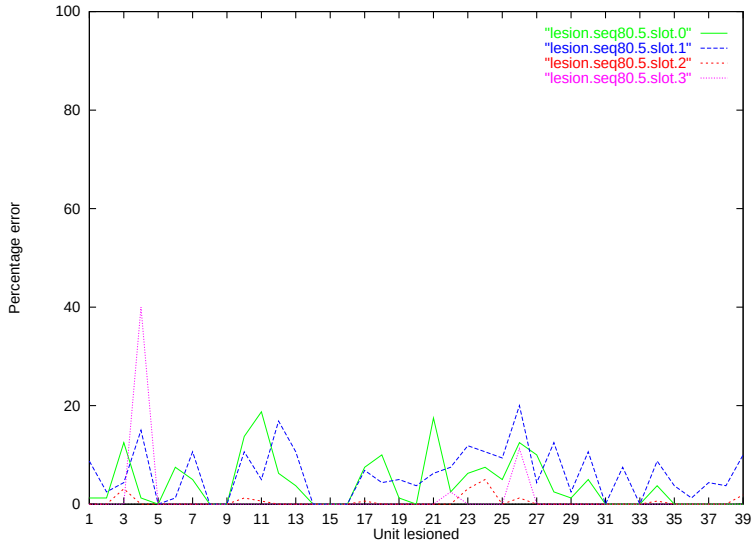

Figure 5.43: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 80 sequences and the 1 unit representation, using 50% noise.

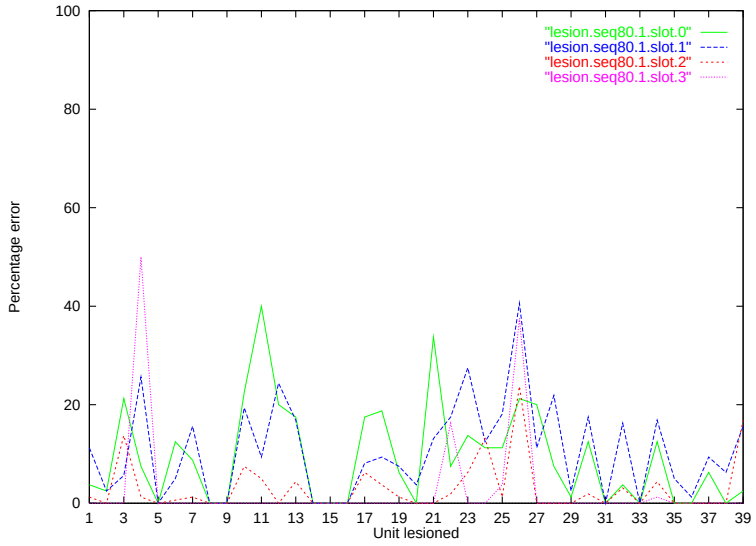

Figure 5.44: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 80 sequences and the 1 unit representation, using 90% noise.

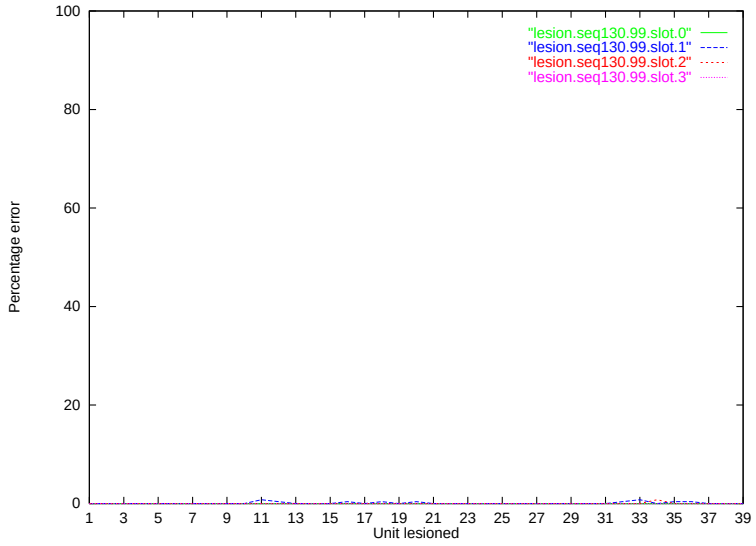

Figure 5.45: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 130 sequences and the 1 unit representation, using 1% noise. A small amount of error was produced at units 11, 16, 18, 20 and 33-36.

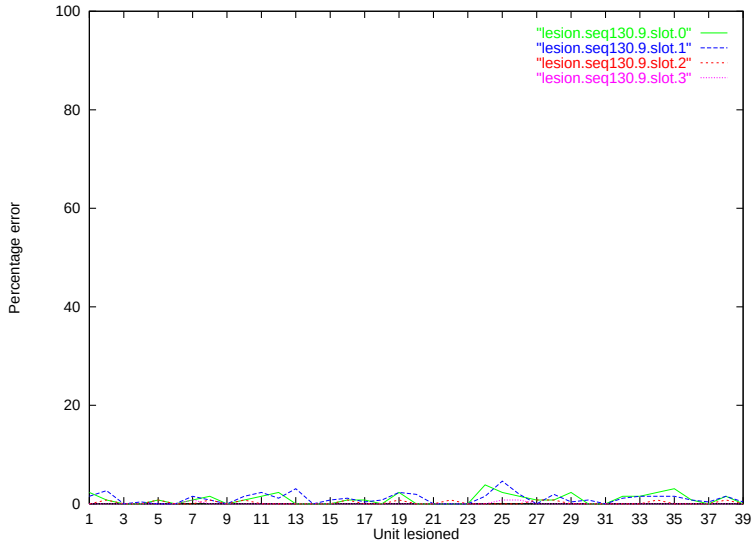

Figure 5.46: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 130 sequences and the 1 unit representation, using 10% noise.

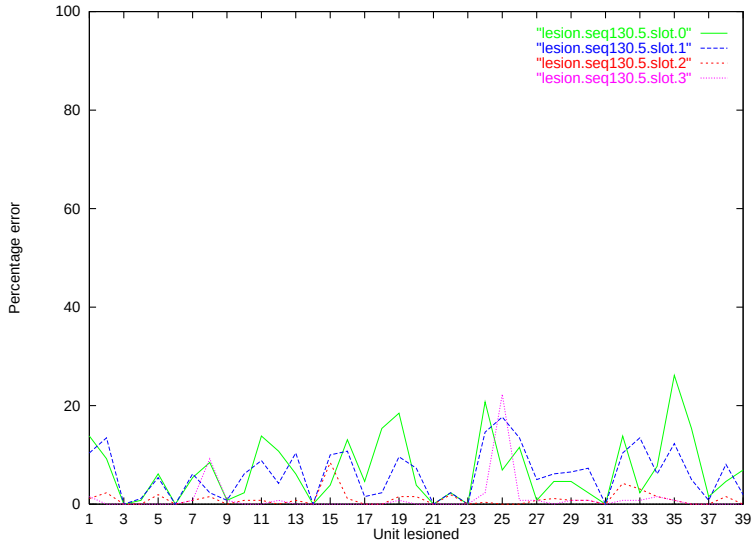

Figure 5.47: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 130 sequences and the 1 unit representation, using 50% noise.

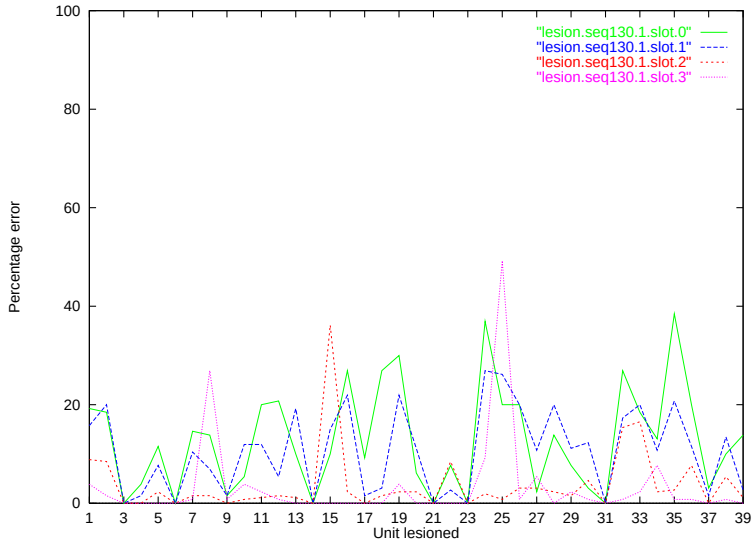

Figure 5.48: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 130 sequences and the 1 unit representation, using 90% noise.

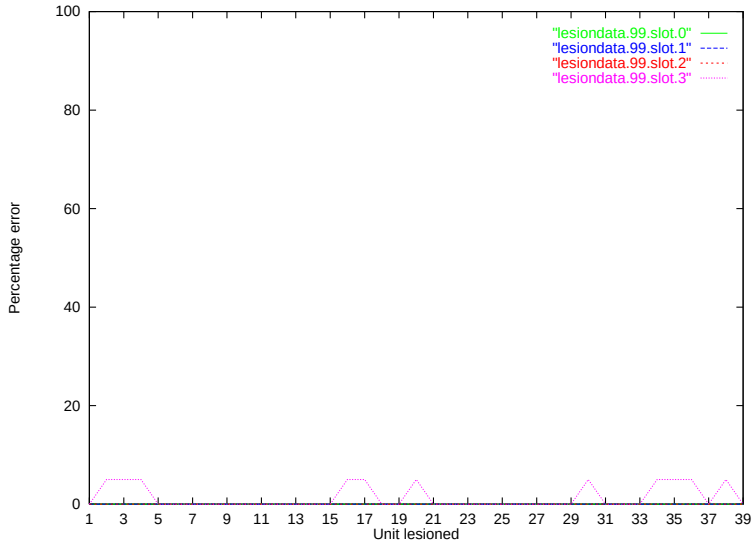

Figure 5.49: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 20 sequences and the 1 unit representation, using 1% noise.

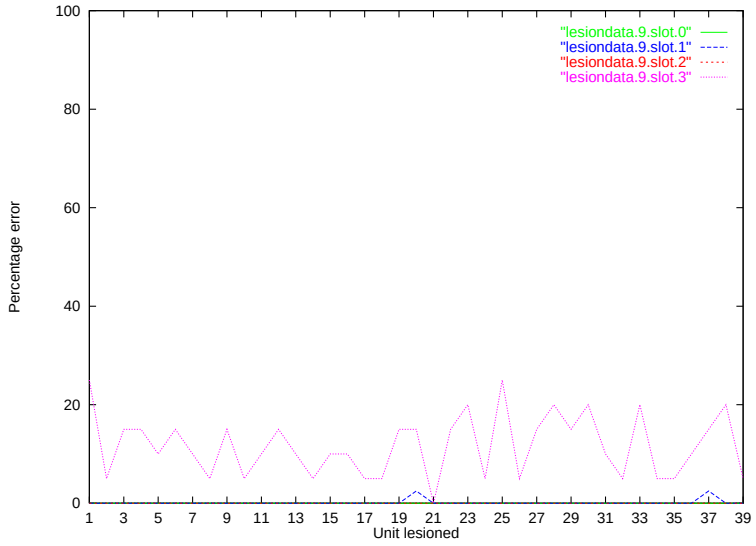

Figure 5.50: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 20 sequences and the 1 unit representation, using 10% noise.

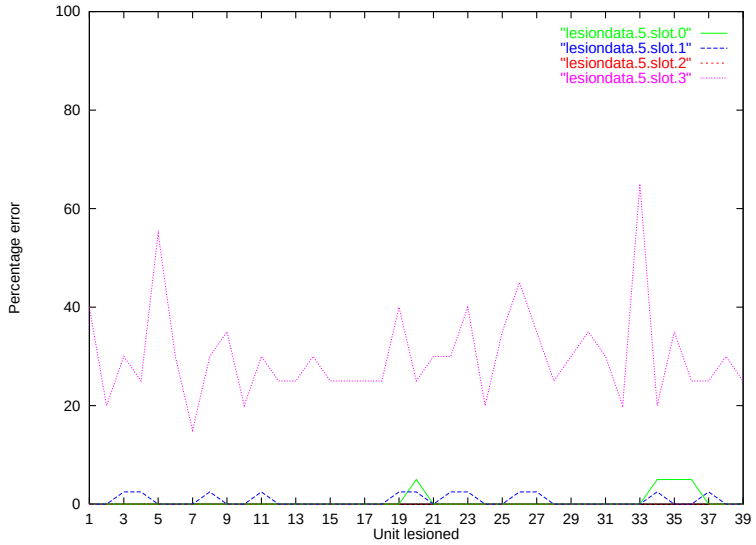

Figure 5.51: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 20 sequences and the 1 unit representation, using 50% noise.

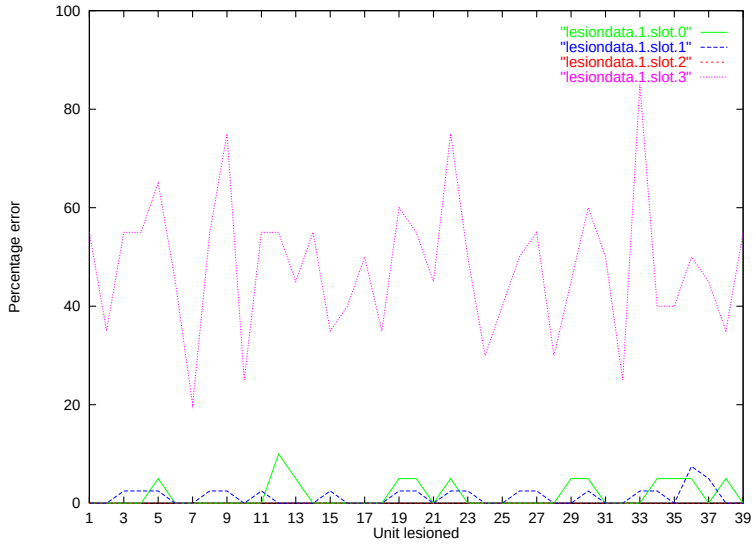

Figure 5.52: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 20 sequences and the 1 unit representation, using 90% noise.

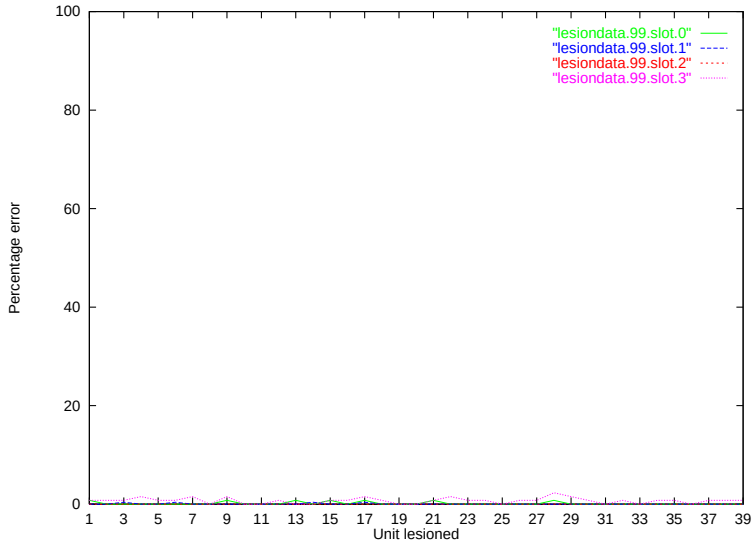

Figure 5.53: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 130 sequences and the 1 unit representation, using 1% noise.

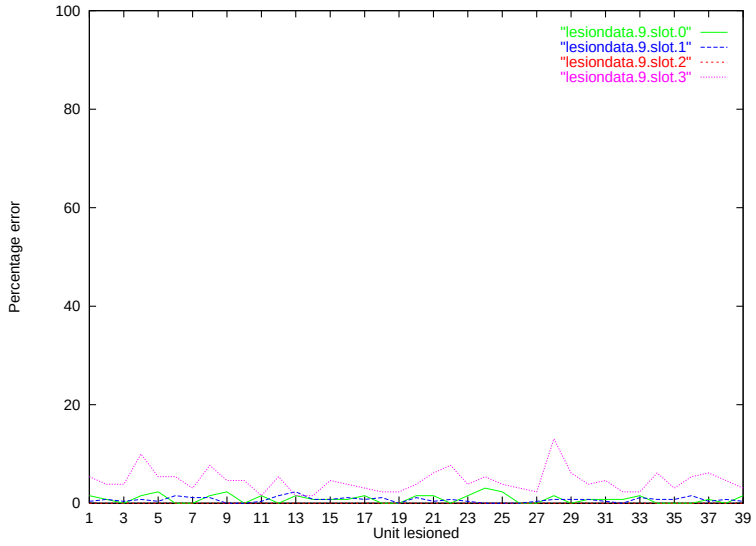

Figure 5.54: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 130 sequences and the 1 unit representation, using 10% noise.

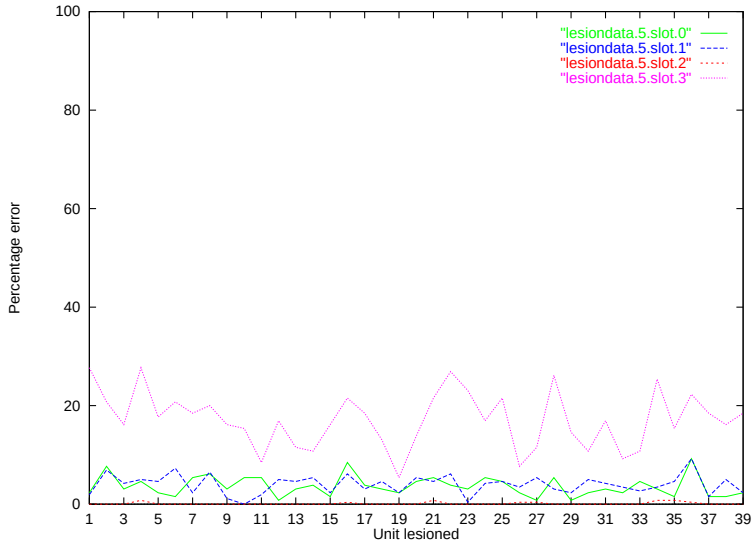

Figure 5.55: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 130 sequences and the 1 unit representation, using 50% noise.

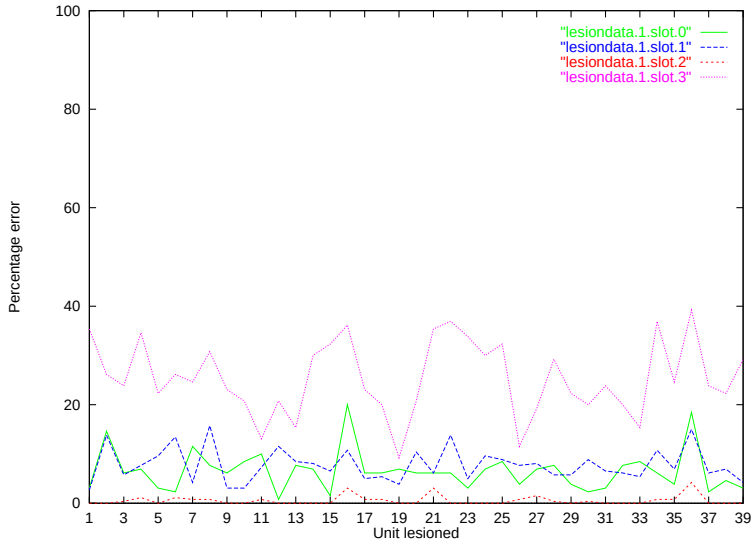

Figure 5.56: Results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 130 sequences and the 1 unit representation, using 90% noise.

Networks trained via Kwasny and Kalman's method

Figures 5.57 to 5.64 illustrate the results of lesioning the representations developed by the SRAAMs trained via Kwasny and Kalman's method on 20 and 130 sequences. The results for 40 and 80 sequences can be found in Appendix B, Section B.2. The representational lesioning plots for the networks trained by Kwasny and Kalman's method are rather different to those for networks trained by back-propagation. Firstly, the error levels are *much* higher than with back-propagation. 1% error is enough to induce considerable levels of error in one or two slots regardless of the number of sequences learned, with 50% or more leading to errors of 50-100% across most slots. Secondly, the increased error associated with more deeply embedded information is more pronounced here when compared to the networks trained by back-propagation. The error levels for each slot are much more widely separated, and slot 2 tends to have fairly low levels of error whilst slot 0 has the higher levels of error after slot 3. Similar results apply for the networks trained on the 2 unit representation with Kwasny and Kalman's method as can be seen in Appendix B, Section B.2. Thus it seems that the representation developed here is still distributed, but much more sensitive to noise than the representation developed with back-propagation.

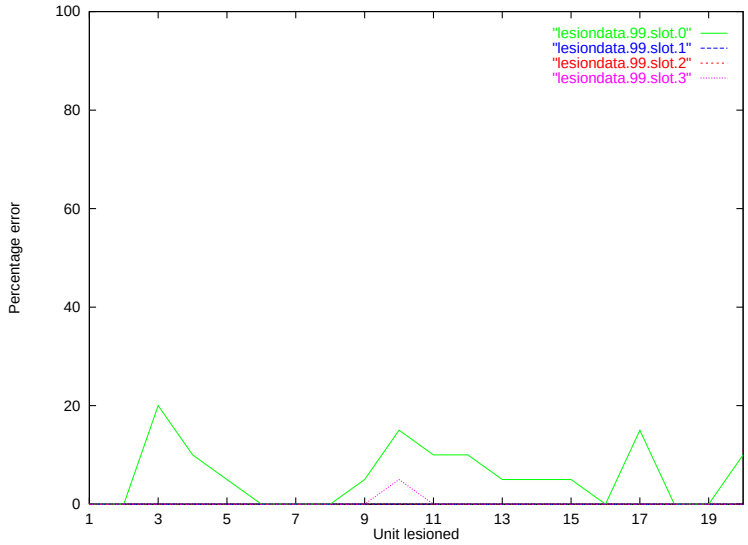

Figure 5.57: Results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method and Kwasny-Kalman units on 20 sequences and the 1 unit representation, using 1% noise.

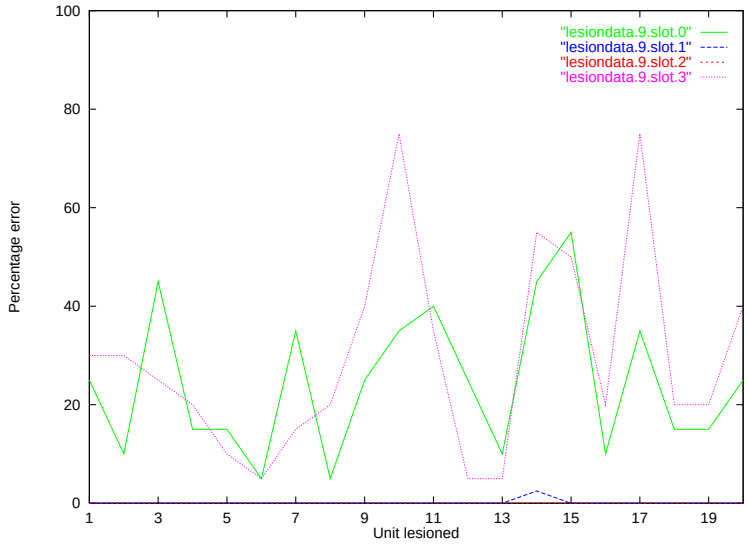

Figure 5.58: Results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 20 sequences and the 1 unit representation, using 10% noise.

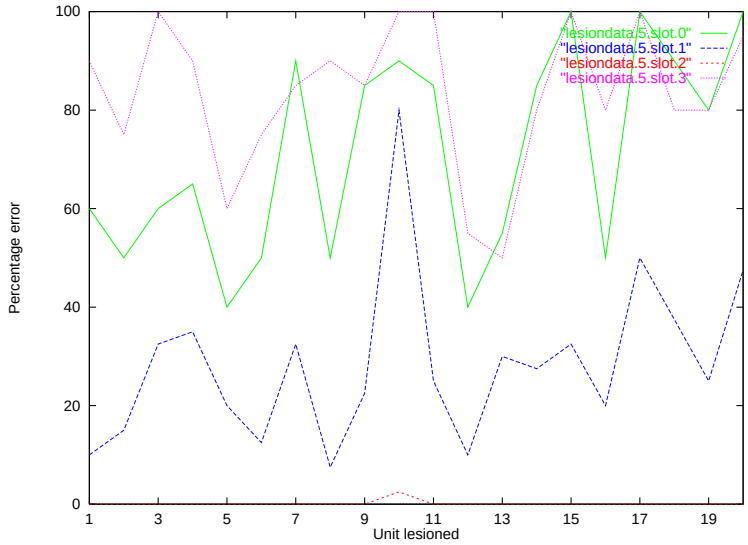

Figure 5.59: Results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 20 sequences and the 1 unit representation, using 50% noise.

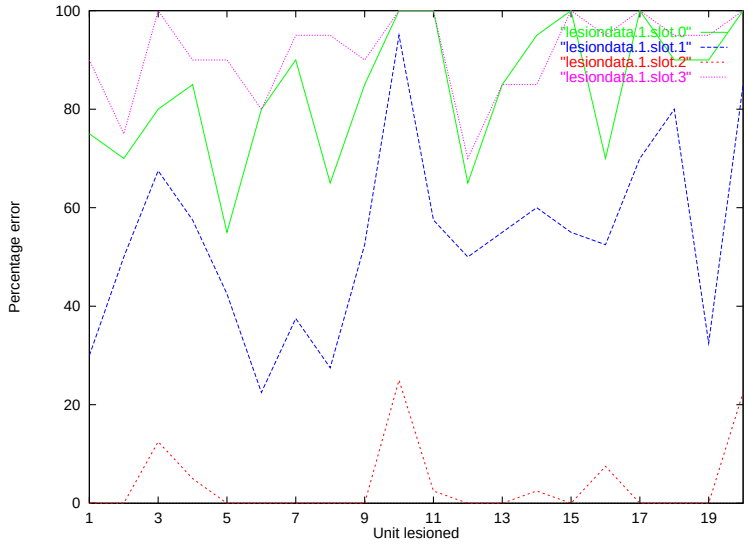

Figure 5.60: Results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 20 sequences and the 1 unit representation, using 90% noise.

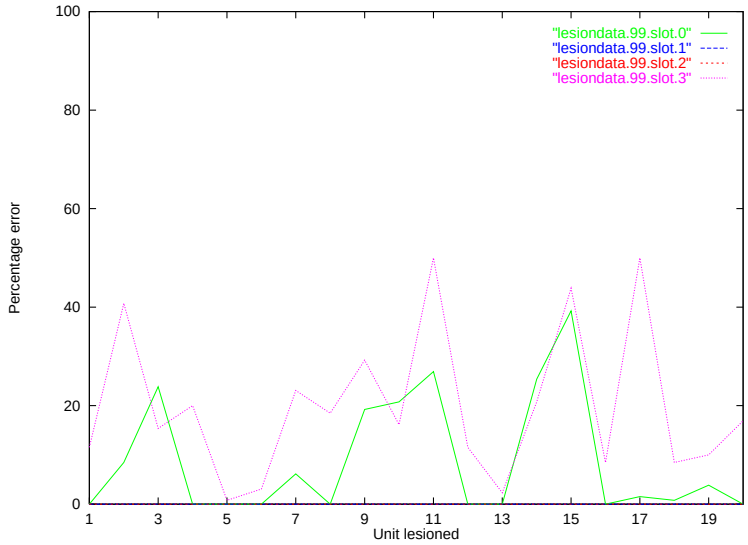

Figure 5.61: Results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 130 sequences and the 1 unit representation, using 1% noise.

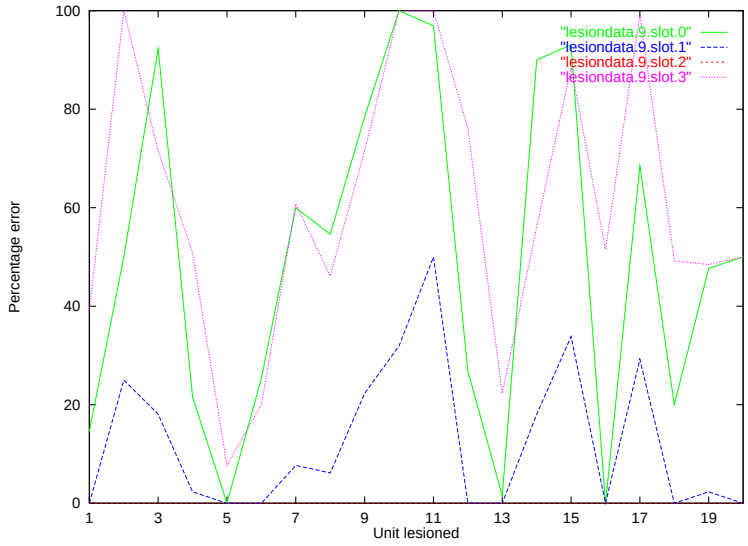

Figure 5.62: Results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 130 sequences and the 1 unit representation, using 10% noise.

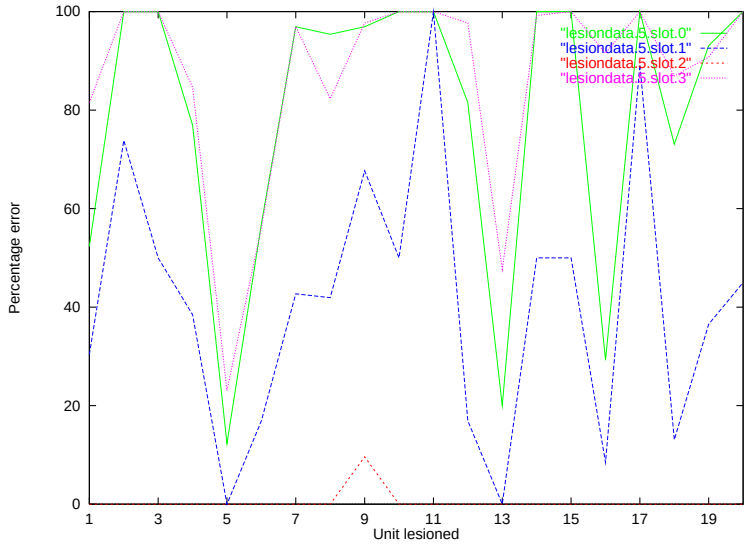

Figure 5.63: Results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 130 sequences and the 1 unit representation, using 50% noise.

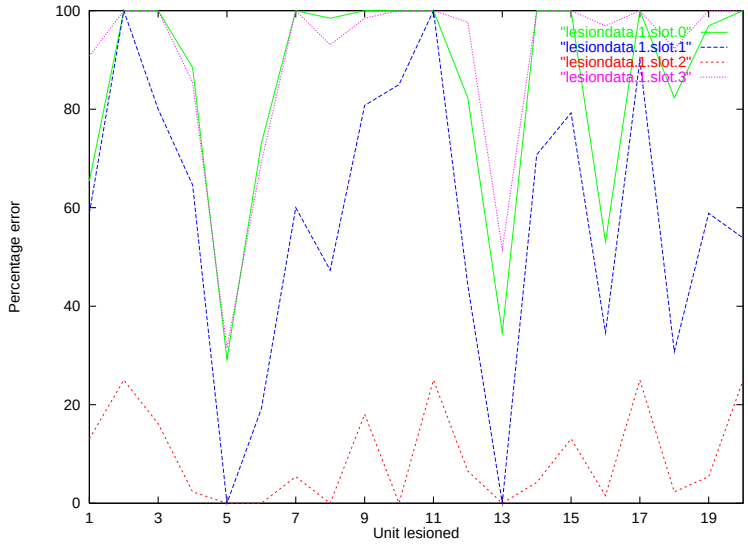

Figure 5.64: Results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 130 sequences and the 1 unit representation, using 90% noise.

5.3.3 Analyses of the encoding and decoding trajectories

For the networks trained on 20 and 130 sequences, the encoding and decoding trajectories were plotted on 25×25 self-organising maps (SOMs) of the hidden-layer states produced by encoding and decoding the training set of sequences used to train the SRAAMs. This analysis was performed twice during training as well as on the completely trained network in order to see if any trends occured during training. Section 5.3.3 presents the results of this analysis for the networks trained with back-propagation and Section 5.3.3 presents the results for the networks trained with Kwasny and Kalman's method. The figures depict the trajectories taken across the SOMs. The arrows show the direction of processing. The encoding and decoding trajectories are plotted separately as are the trajectories for active and passive sentences. The labels show which position in the sequence each point belongs to, i.e. "0"s depict the first word, "1"s the next and so on.

Networks trained via back-propagation

Figures 5.65 to 5.70 show the encoding and decoding trajectories for the SRAAMs with sigmoidal units trained on the 1 unit representation.

There are several things to note about these plots. Firstly there seems to be no particular trend in the organisation of the trajectories as training proceeds. All of the trajectories seem to be organised in a rather jumbled manner. Secondly all of the labels seem to cluster together into one or two groups per label, suggesting that the hidden layer states for a particular position in different sequences are located in the same areas of hyperspace. Thirdly, the encoding and decoding trajectories are similar to each other but not identical for the plots of 20 sequences. For 130 sequences the similarity of the encoding and decoding trajectories for active sentences is more pronounced than for passive sentences, especially at the completion of training. Ideally the encoding and decoding trajectories would be the reverse of each other, however in practice this would not be achieved, but they would be close. The greater similarity of the encoding and decoding trajectories for active and passive sentences reflects the greater success the SRAAM has in encoding and decoding these sequences compared to the passive sequences.

With the 2 unit representation (Figures C.1 to C.6), the networks trained on 20 sequences, the plots are similar in form to the 1 unit representation.

However with 130 sequences, the plot after 1000 iterations shows all the points collapsing to two points on the SOM, which taken at face value suggests that all the hidden layer states are very similar to each other and fall into one of 2 groups. By the end of training the points are well spread out again, showing the same features as in the other plots, such as the labels clustering together and greater similarity in the encoding and decoding trajectories for active sentences than for the passive sentences.

Figures 5.71 to 5.76 show the encoding and decoding trajectories for the SRAAMs with hyperbolic tangent units trained on the 1 unit representation. The plots for the SRAAM trained on 20 sequences here all show the hidden layer states being mapped onto two or three points. The plots after 2600 iterations and at the end of training all show the states being mapped to two points whilst the plot after 5000 iterations shows that passive sentences are mapped onto three points, with the third word (label "2") sometimes being mapped onto one point and the rest onto the other two. The plots for 130 sequences start off with the same phenomenon and then revert to the behaviour of the earlier plots.

Figures 5.77 to 5.82 show the encoding and decoding trajectories for the SRAAMs with hyperbolic tangent units trained on the 2 unit representation. The 20 sequence plots here revert to the form of the plots for the sigmoidal units and 1 unit representation, however the 130 sequence plots show the same collapsing of the hidden-states to two points for the active sequences, but with a cluster of points for the passive sequences. Again the encoding and decoding trajectories for the active sequences are more similar than for the passive in this case, and the plots suggest that the hidden layer states for the passive sequences are more spread out than for the active sequences.

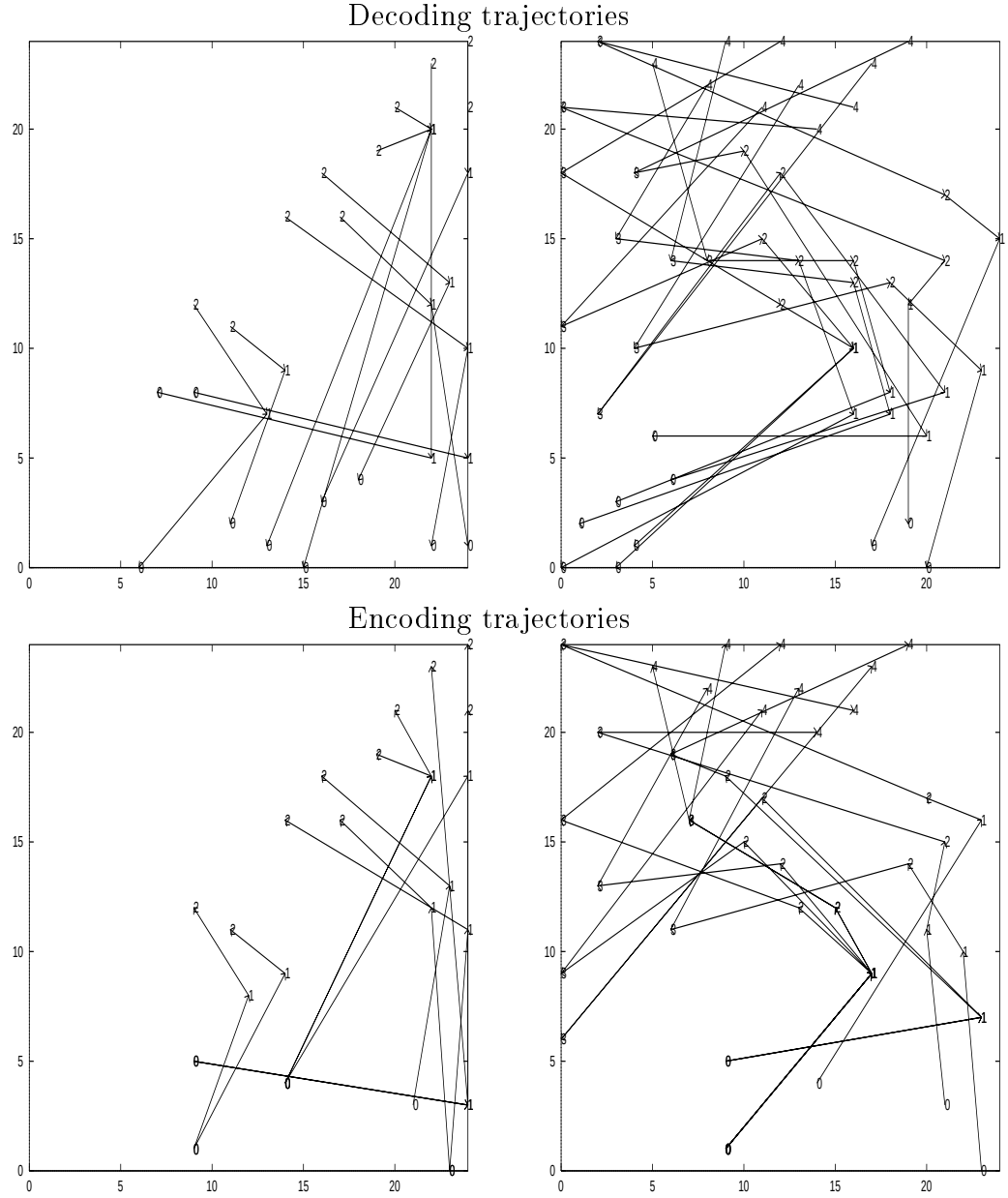

Figure 5.65: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation and sigmoidal units after 1500 iterations. Active sentences on the left, passive on the right.

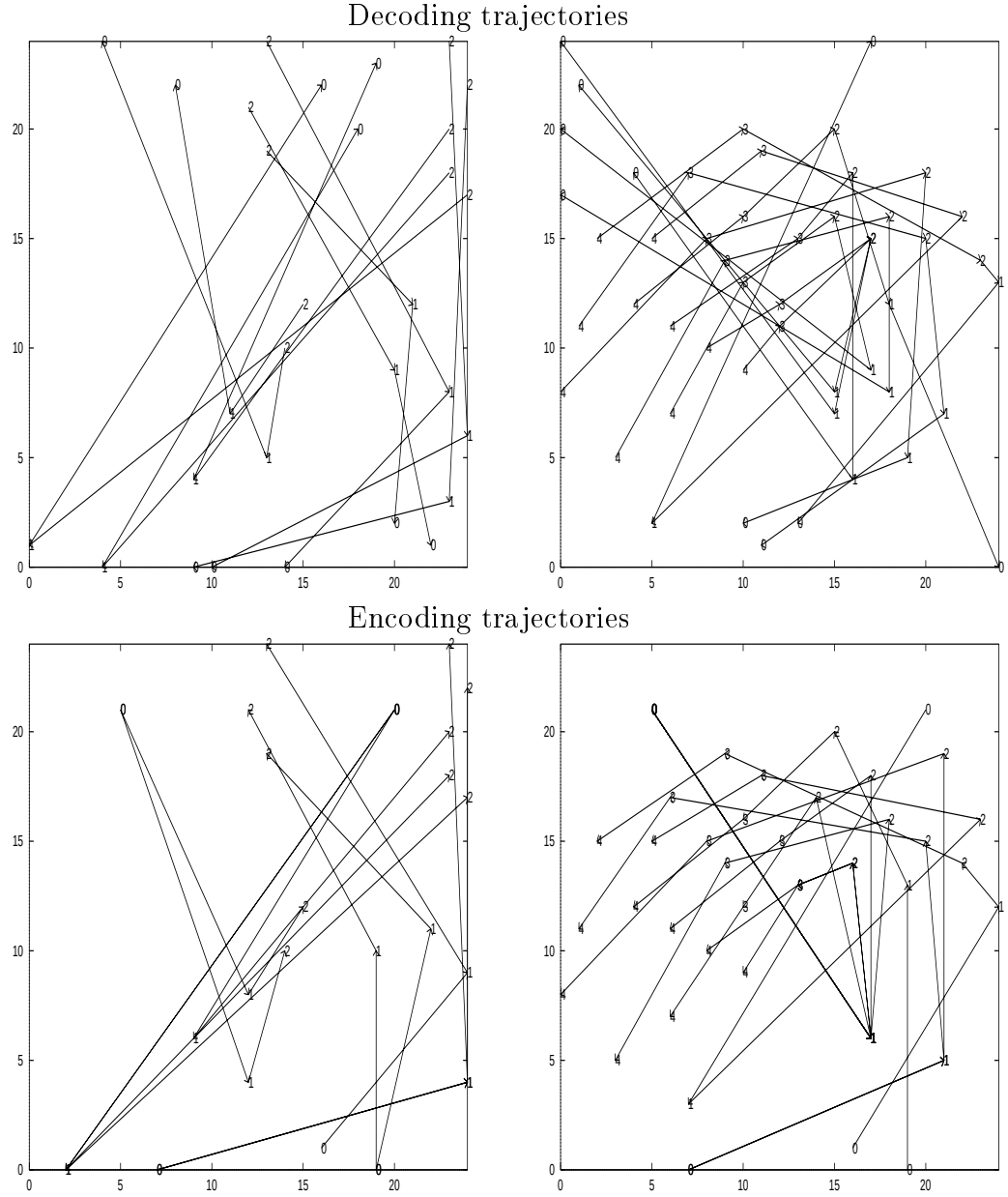

Figure 5.66: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation, back-propagation and sigmoidal units after 3000 iterations. Active sentences on the left, passive on the right.

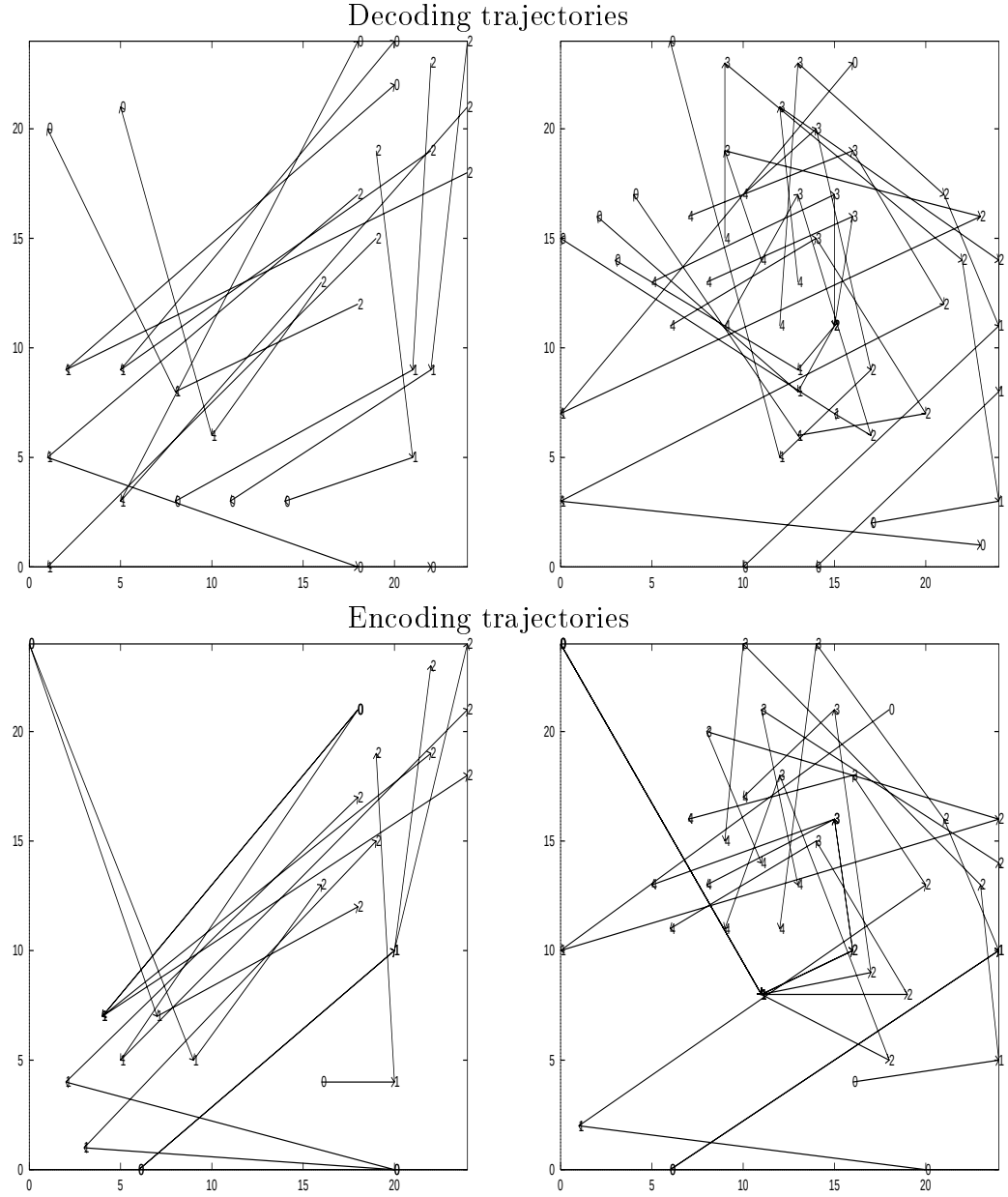

Figure 5.67: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation, back-propagation and sigmoidal units after 4318 iterations (training complete). Active sentences on the left, passive on the right.

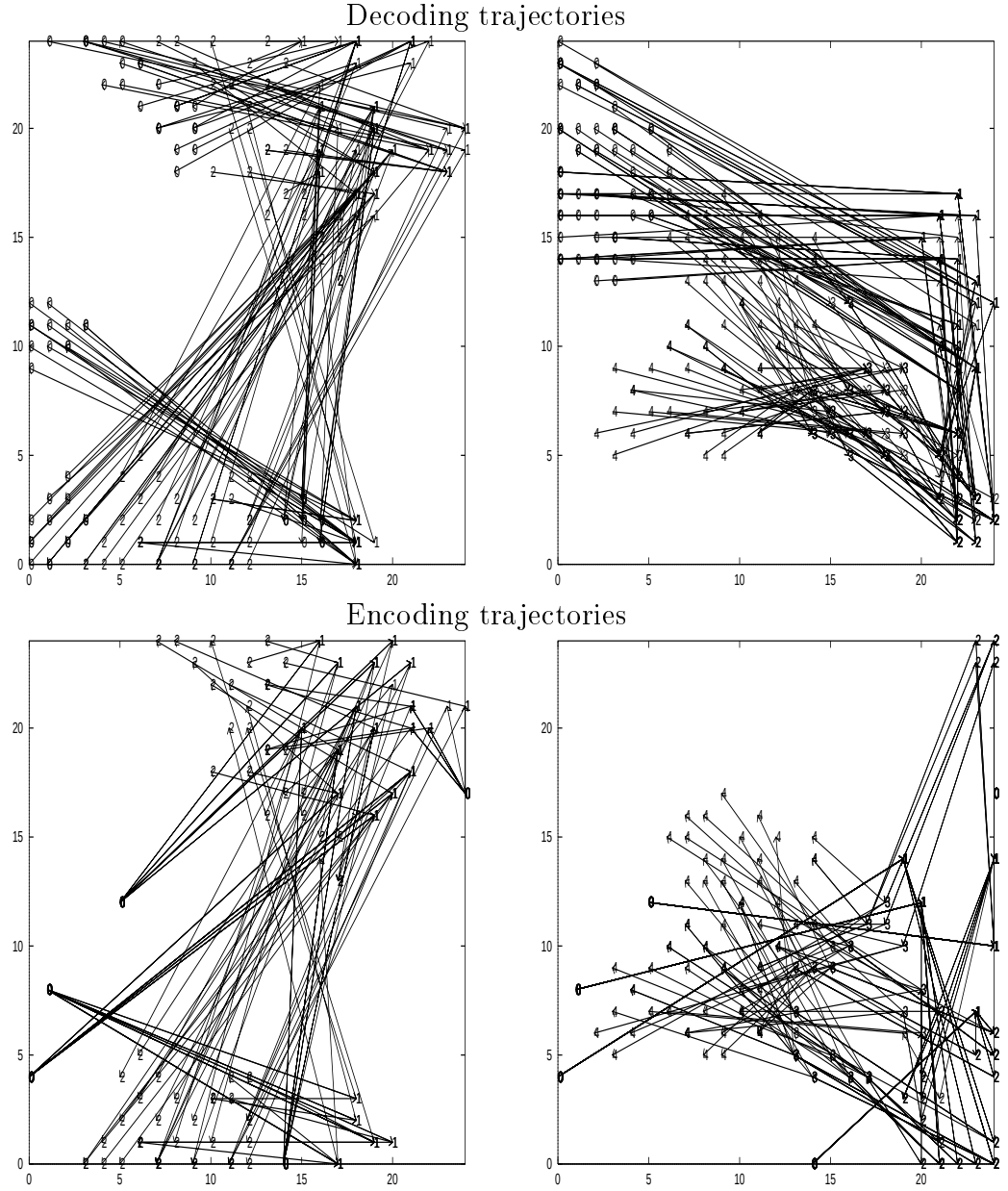

Figure 5.68: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation, back-propagation and sigmoidal units after 600 iterations. Active sentences on the left, passive on the right.

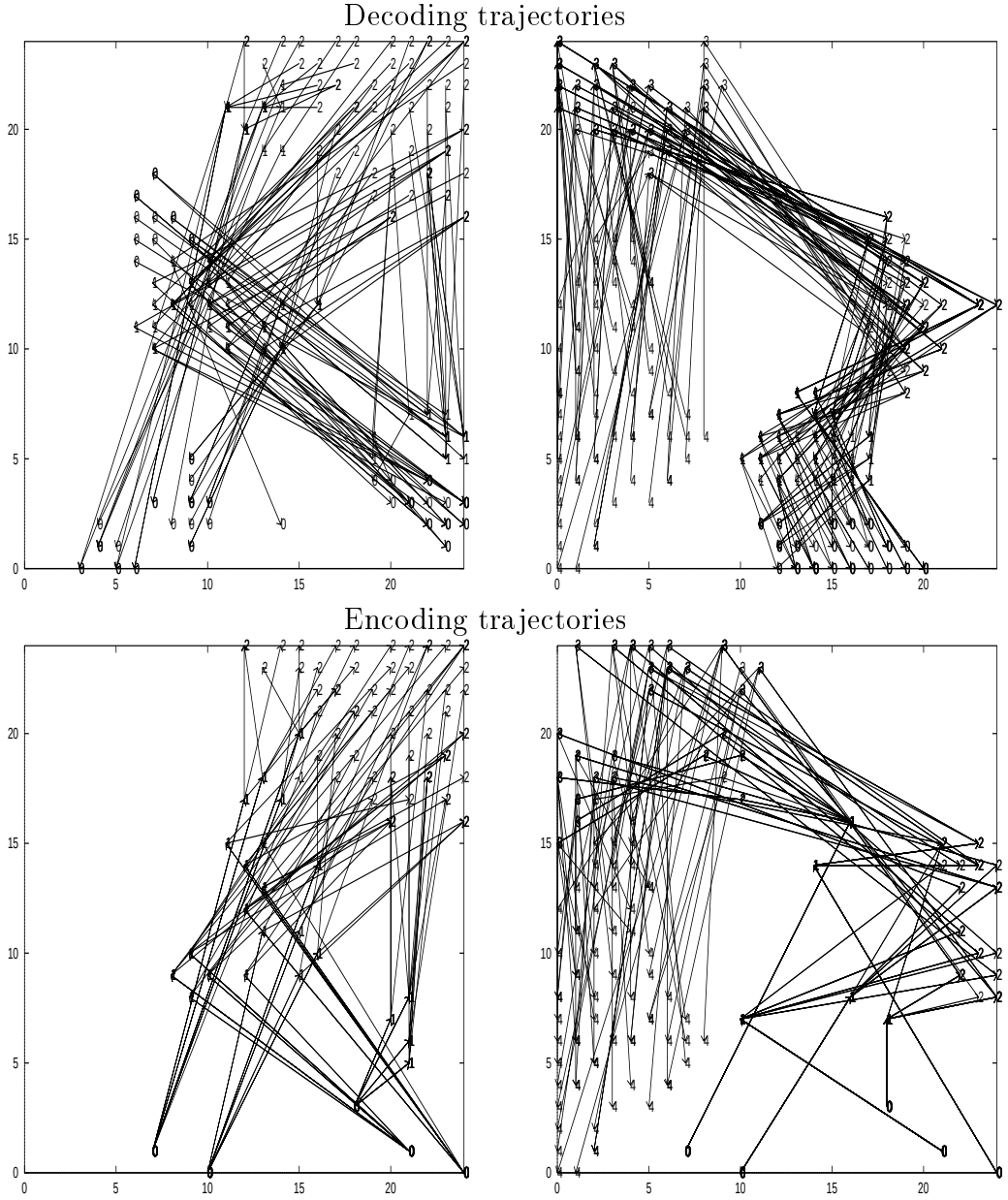

Figure 5.69: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation, back-propagation and sigmoidal units after 1200 iterations. Active sentences on the left, passive on the right.

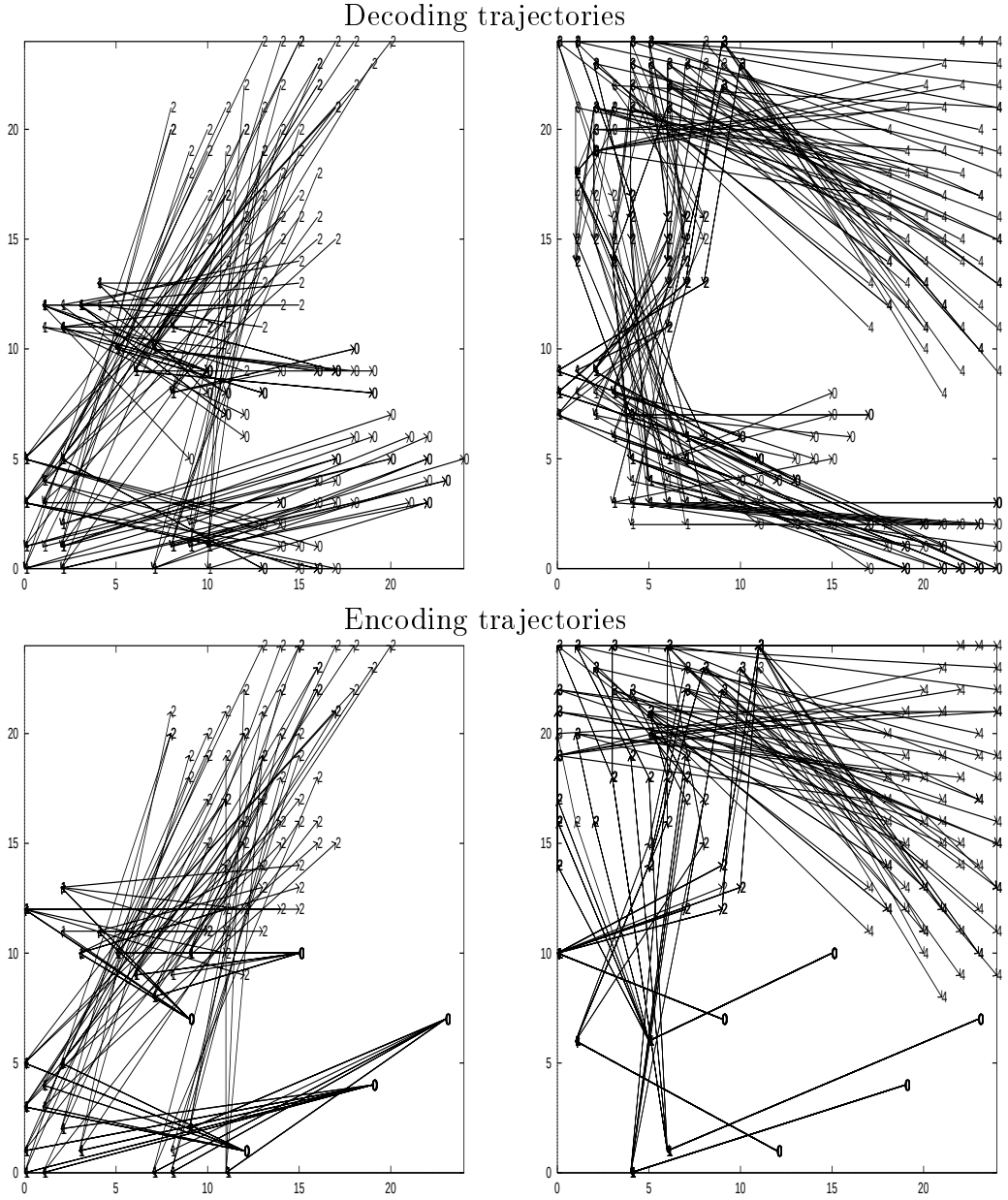

Figure 5.70: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation, back-propagation and sigmoidal units after 1820 iterations. Active sentences on the left, passive on the right.

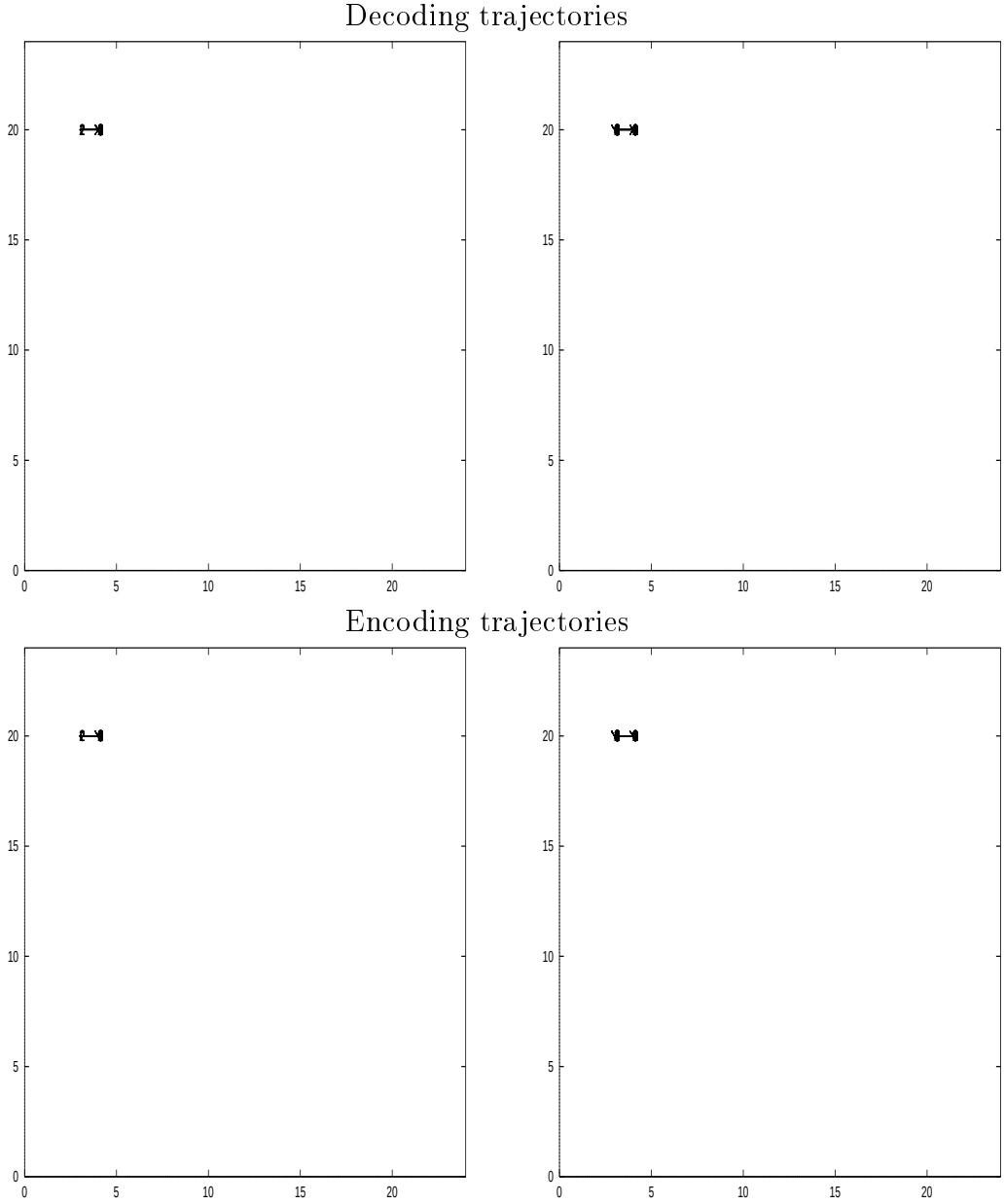

Figure 5.71: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation and hyperbolic tangent units after 2600 iterations. Active sentences on the left, passive on the right.

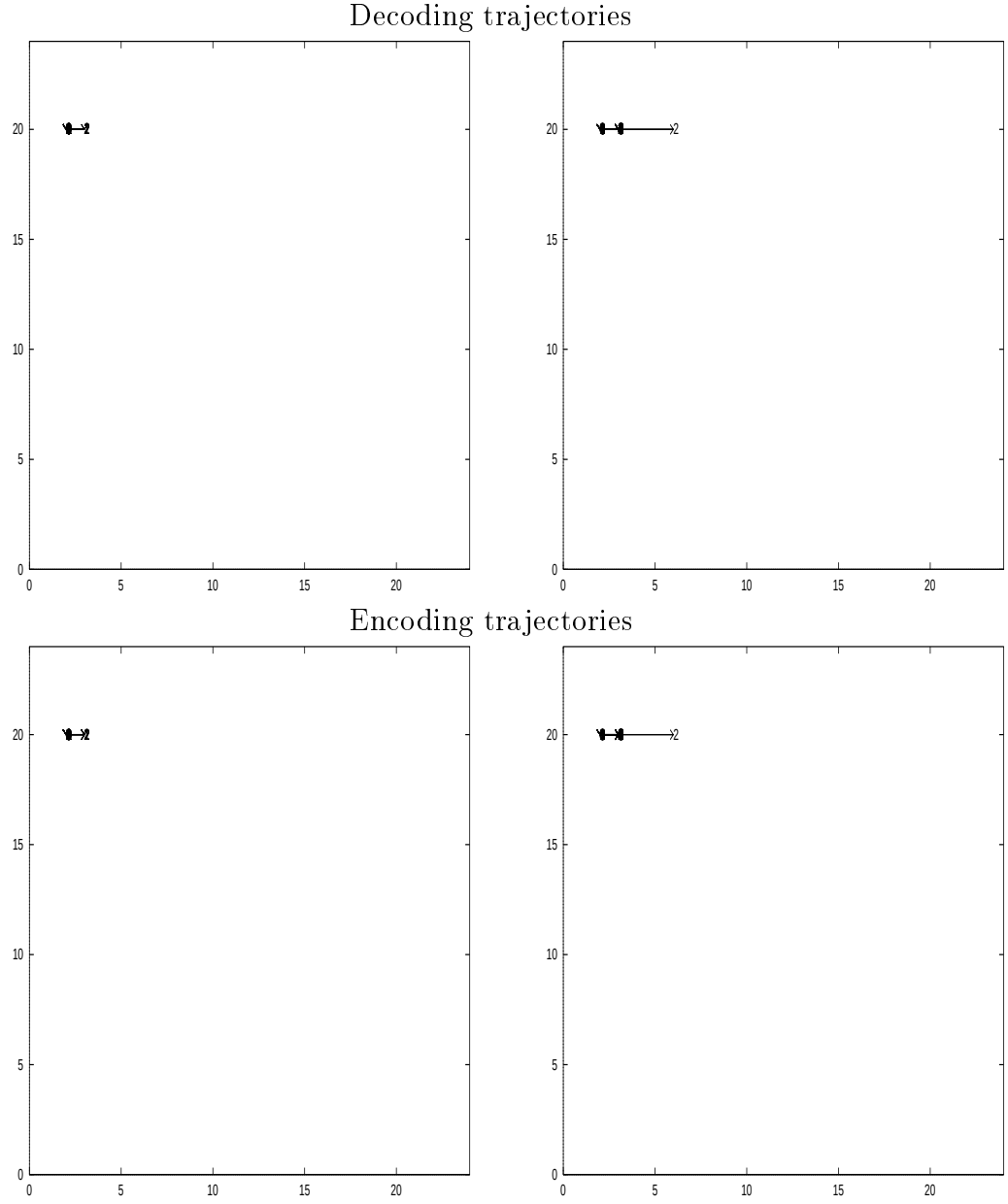

Figure 5.72: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation, back-propagation and hyperbolic tangent units after 5000 iterations. Active sentences on the left, passive on the right.

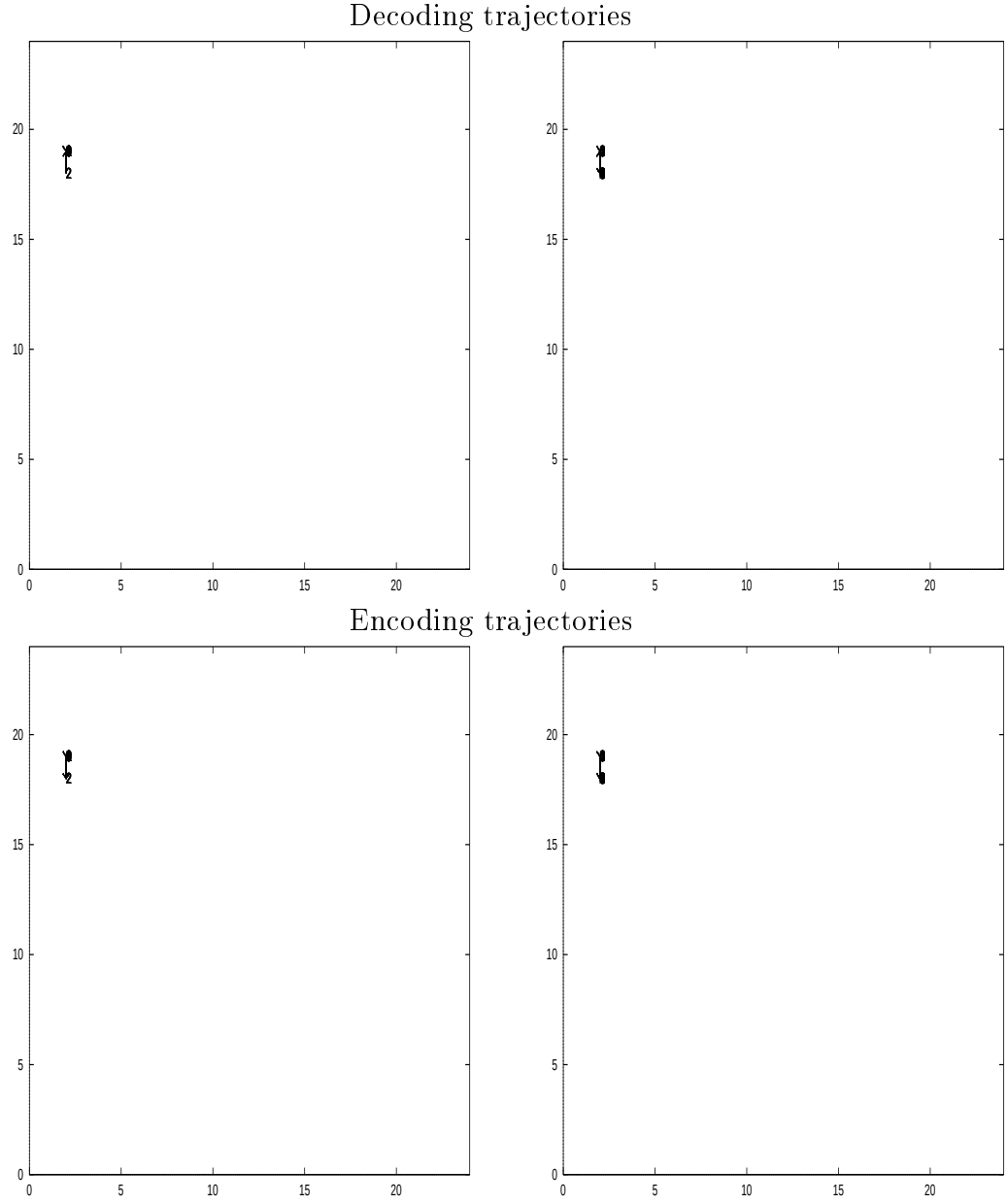

Figure 5.73: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation, back-propagation and hyperbolic tangent units after 7571 iterations (training complete). Active sentences on the left, passive on the right.

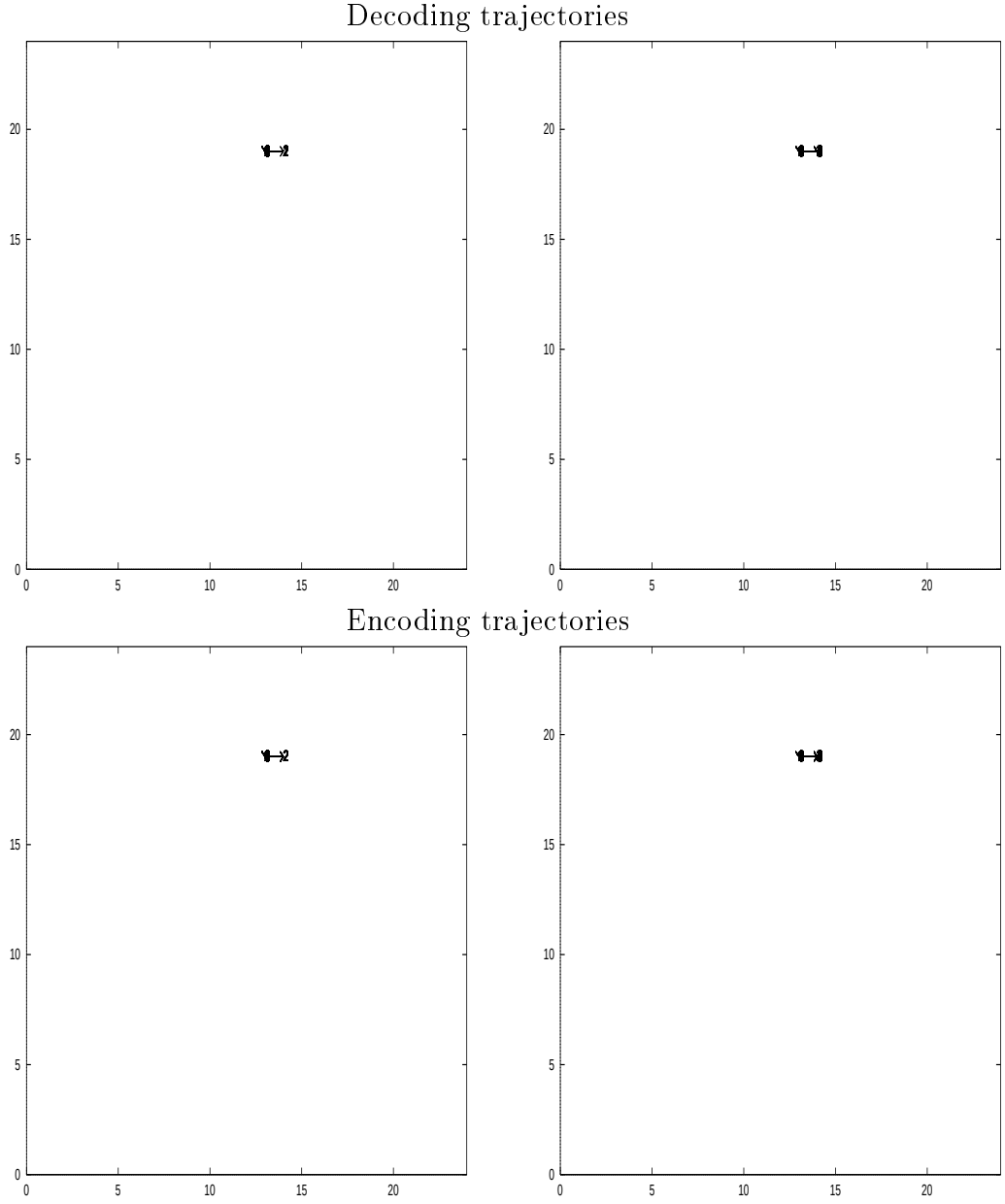

Figure 5.74: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation, back-propagation and hyperbolic tangent units after 100 iterations. Active sentences on the left, passive on the right.

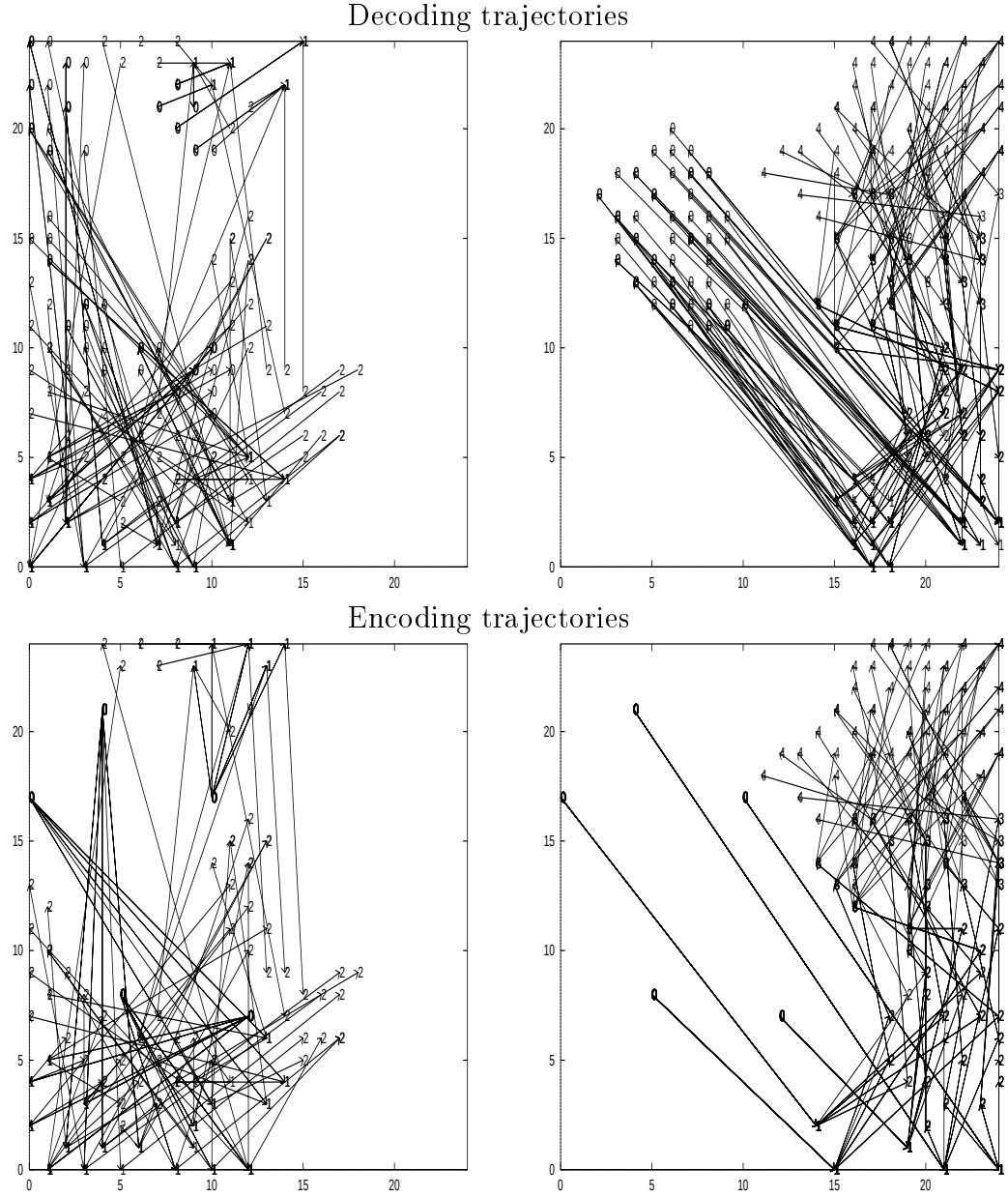

Figure 5.75: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation, back-propagation and hyperbolic tangent units after 200 iterations. Active sentences on the left, passive on the right.

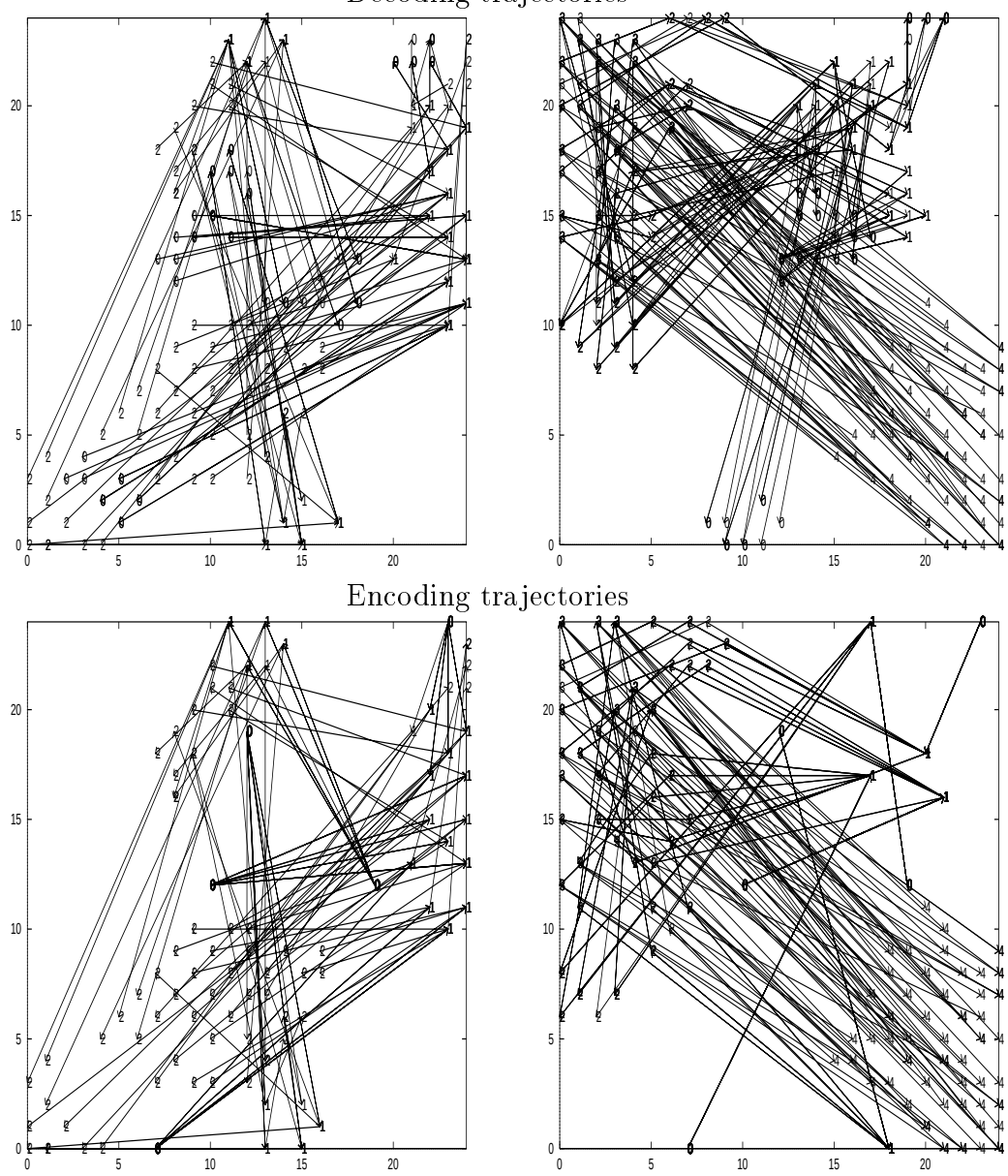

Figure 5.76: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation, back-propagation and hyperbolic tangent units after 270 iterations. Active sentences on the left, passive on the right.

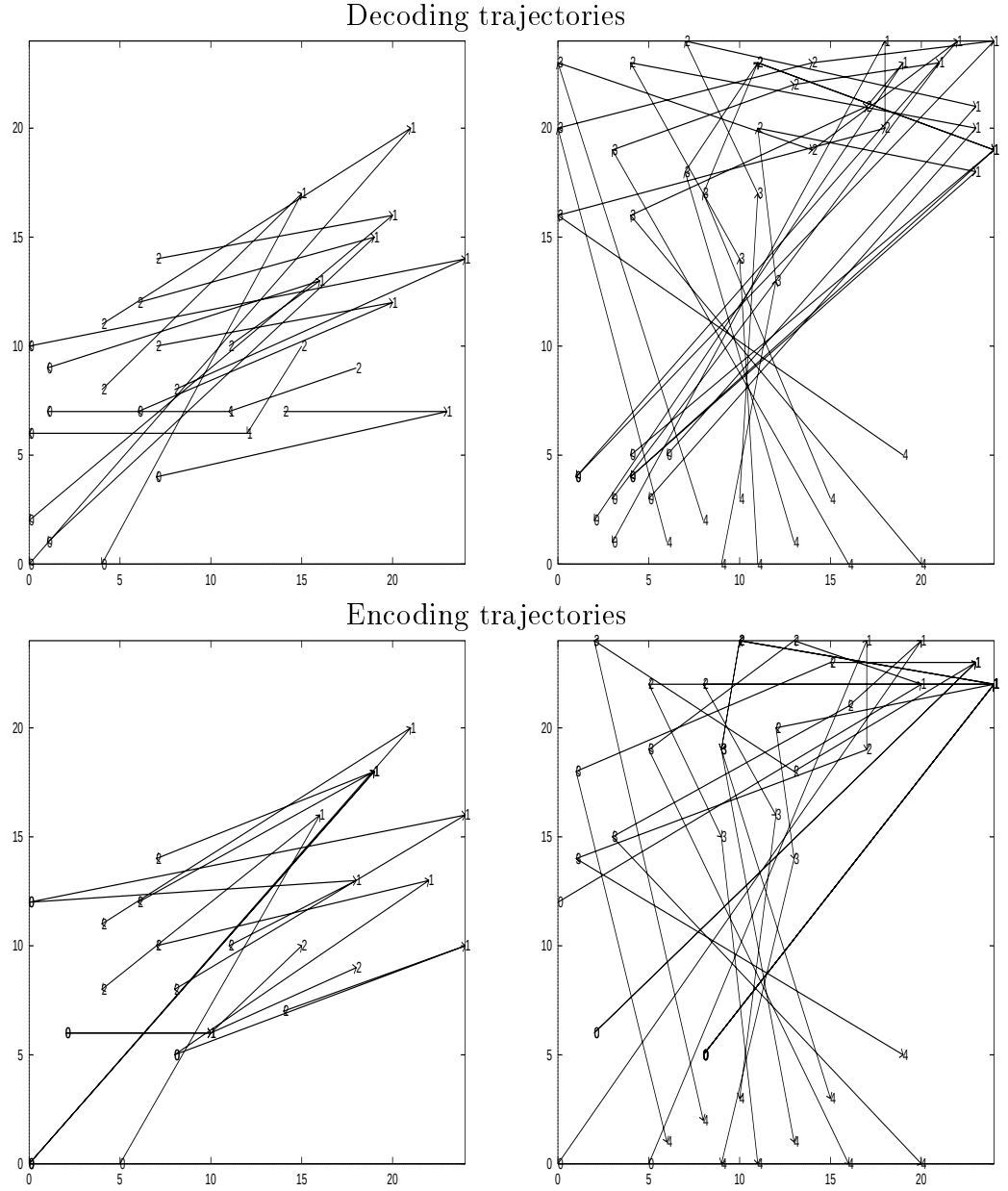

Figure 5.77: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation, back-propagation and hyperbolic tangent units after 1500 iterations. Active sentences on the left, passive on the right.

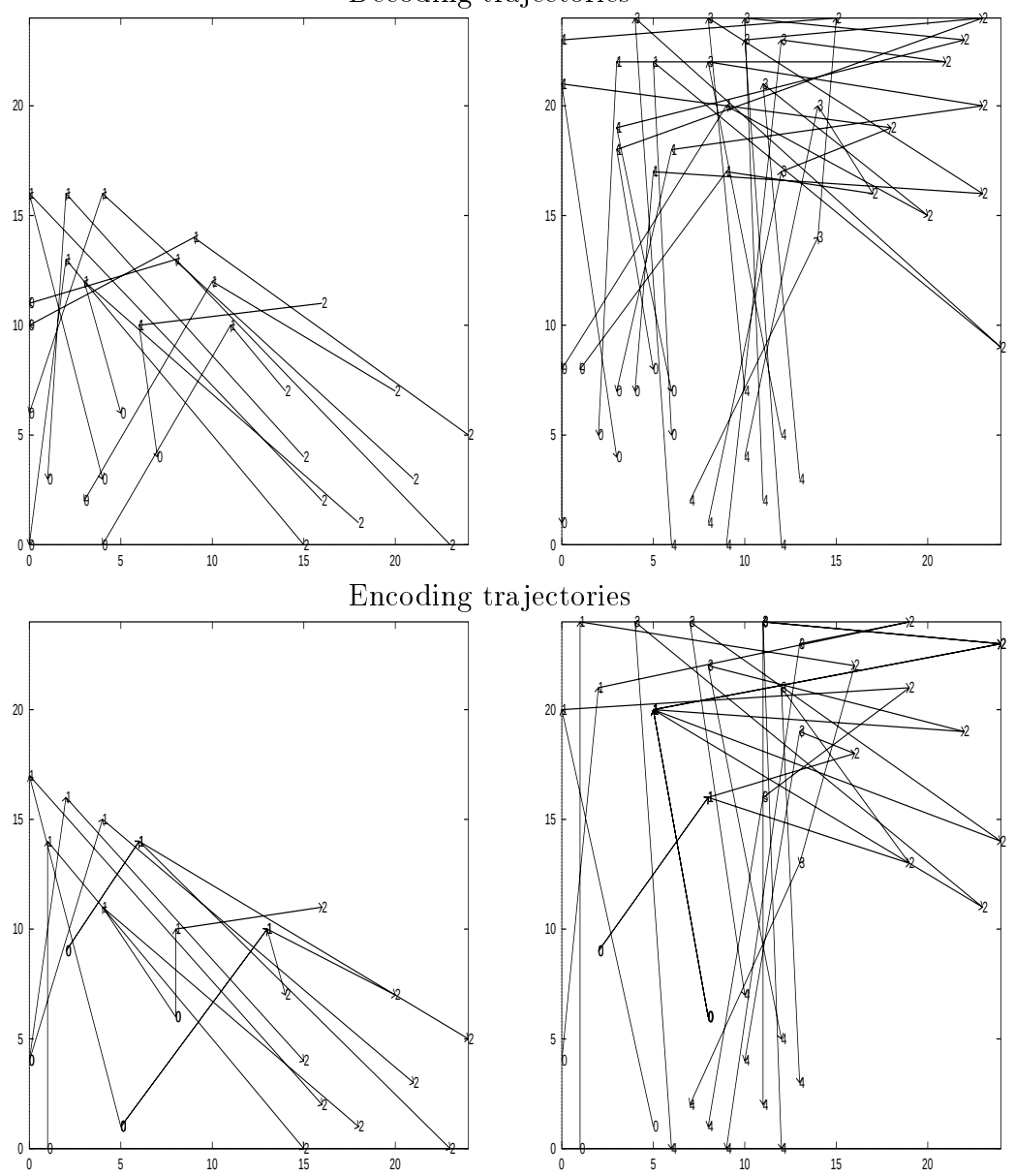

Figure 5.78: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation, back-propagation and hyperbolic tangent units after 3000 iterations. Active sentences on the left, passive on the right.

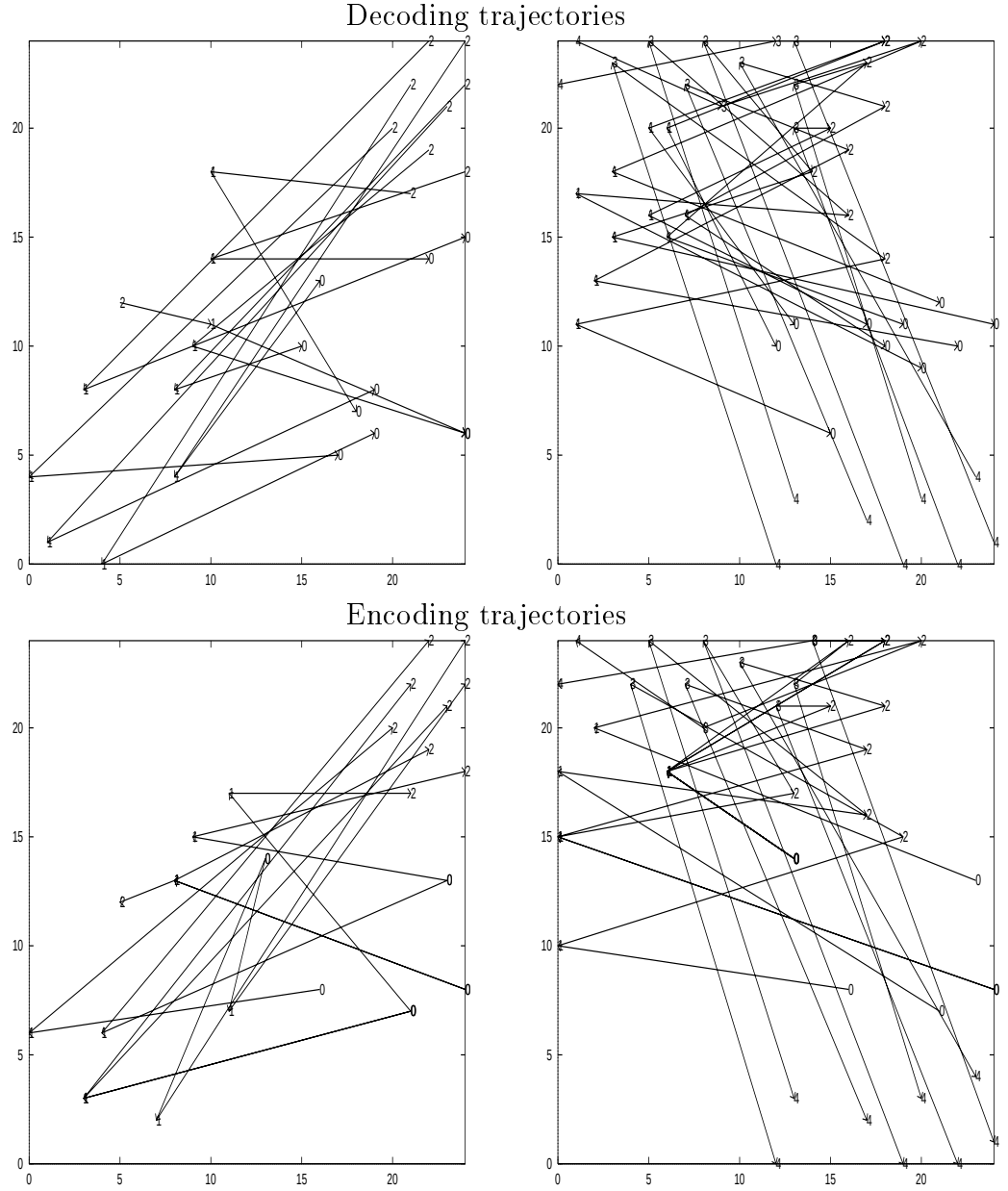

Figure 5.79: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation, back-propagation and hyperbolic tangent units after 4286 iterations (training complete). Active sentences on the left, passive on the right.

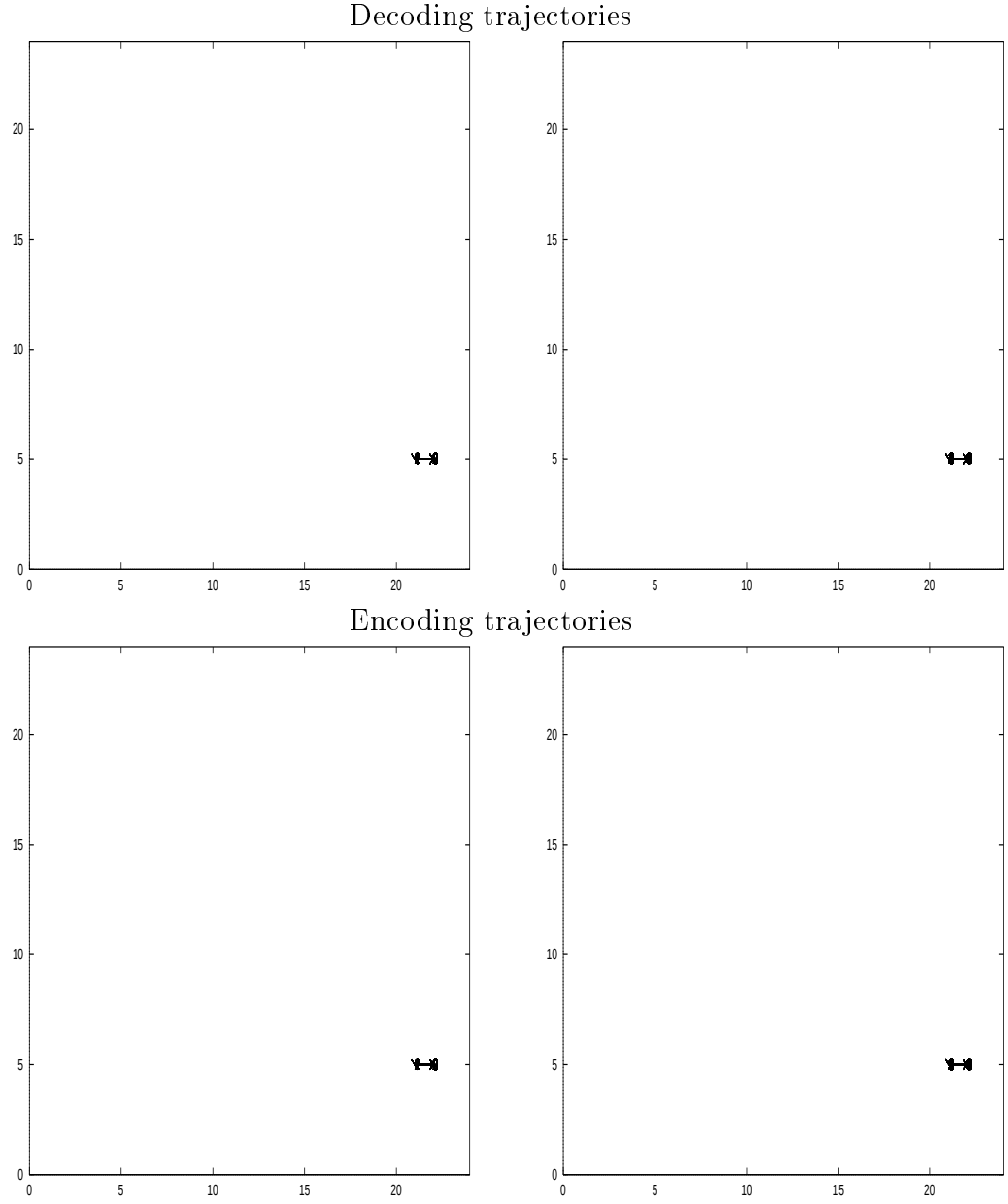

Figure 5.80: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation, back-propagation and hyperbolic tangent units after 100 iterations. Active sentences on the left, passive on the right.

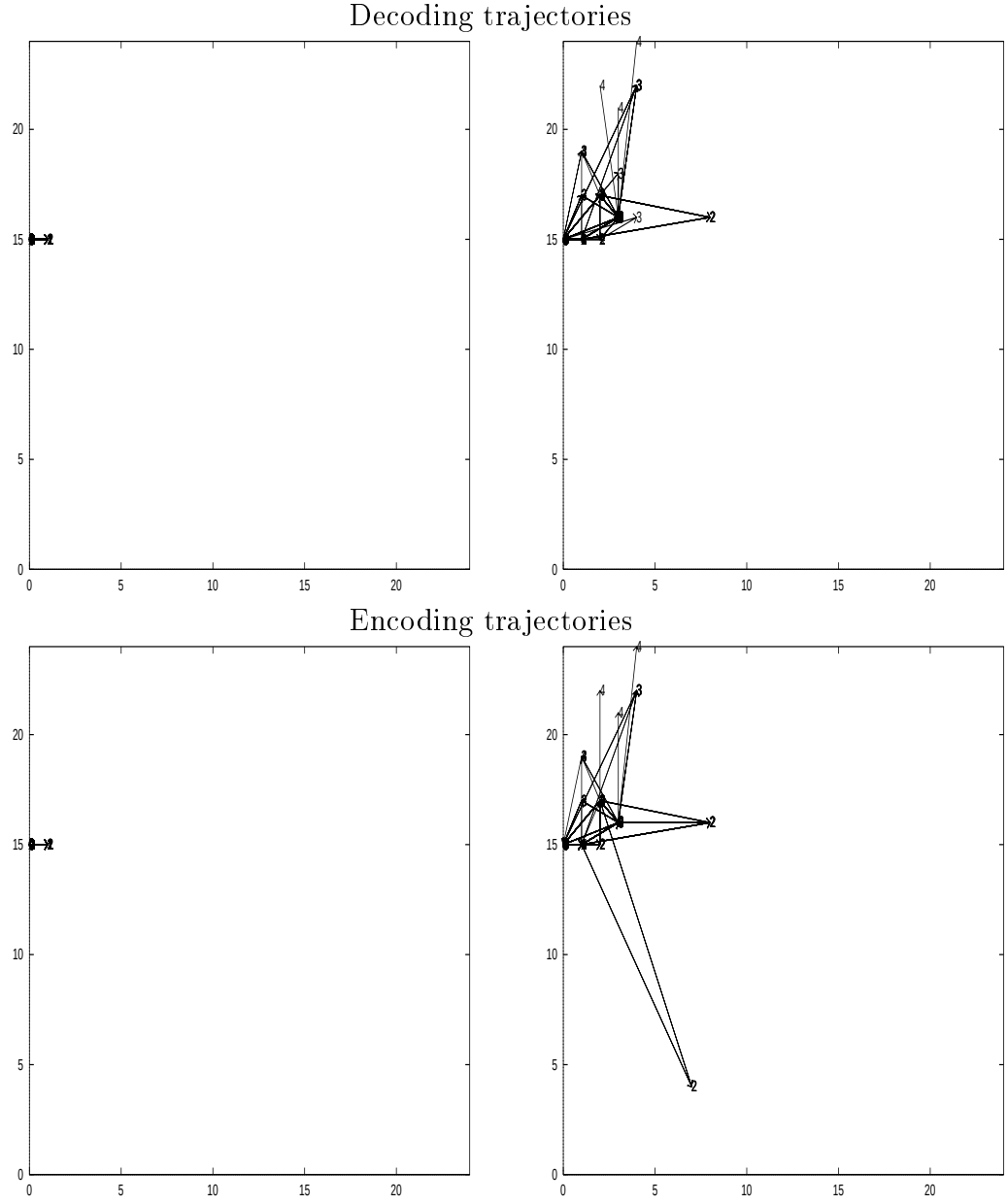

Figure 5.81: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation, back-propagation and hyperbolic tangent units after 200 iterations. Active sentences on the left, passive on the right.

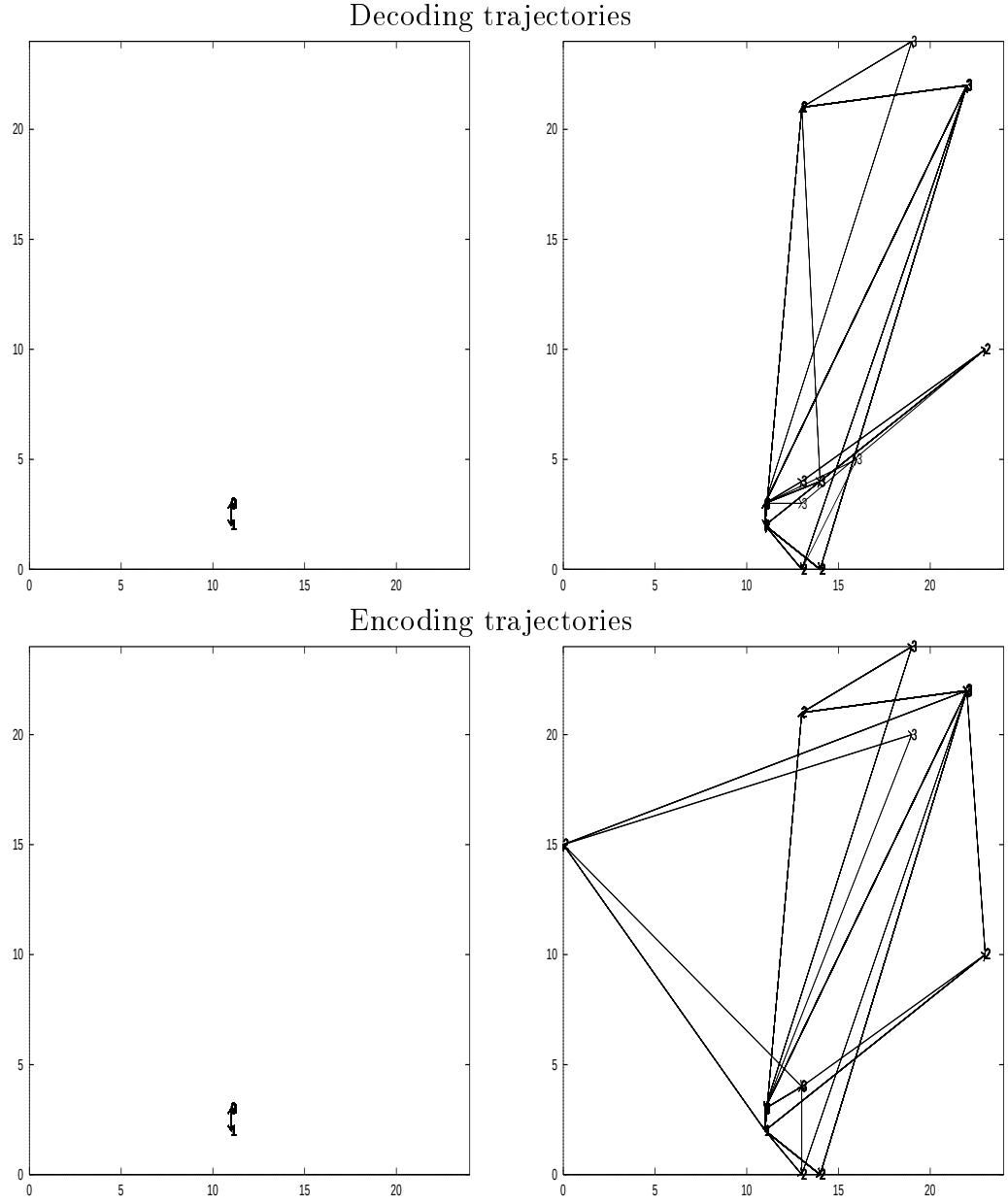

Figure 5.82: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation, back-propagation and hyperbolic tangent units after 243 iterations. Active sentences on the left, passive on the right.

Networks trained via Kwasny and Kalman's method

Figures 5.83 to 5.88 show the encoding and decoding trajectories for the SRAAMs trained with Kwasny and Kalman's method on the 1 unit representation. All of the plots here show the hidden layer states collapsing to 2 points as did the plots for the 2 unit representation (see Appendix C), with the exception of the plots of 130 sequences trained to a tolerance of 0.16 where the hidden layers collapsed to just one point!

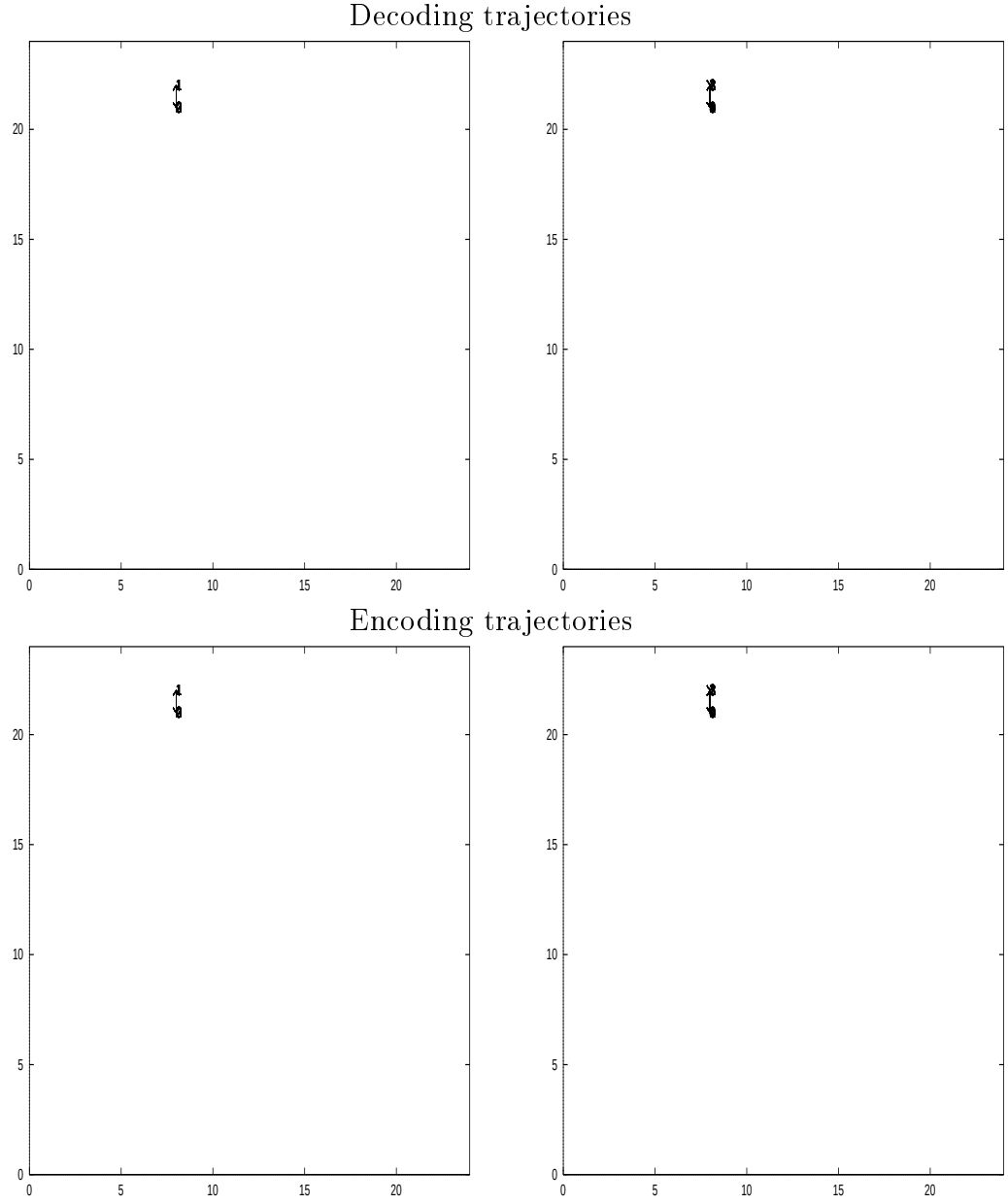

Figure 5.83: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation and Kwasny and Kalman's training after training to a tolerance of 0.16. Active sentences on the left, passive on the right.

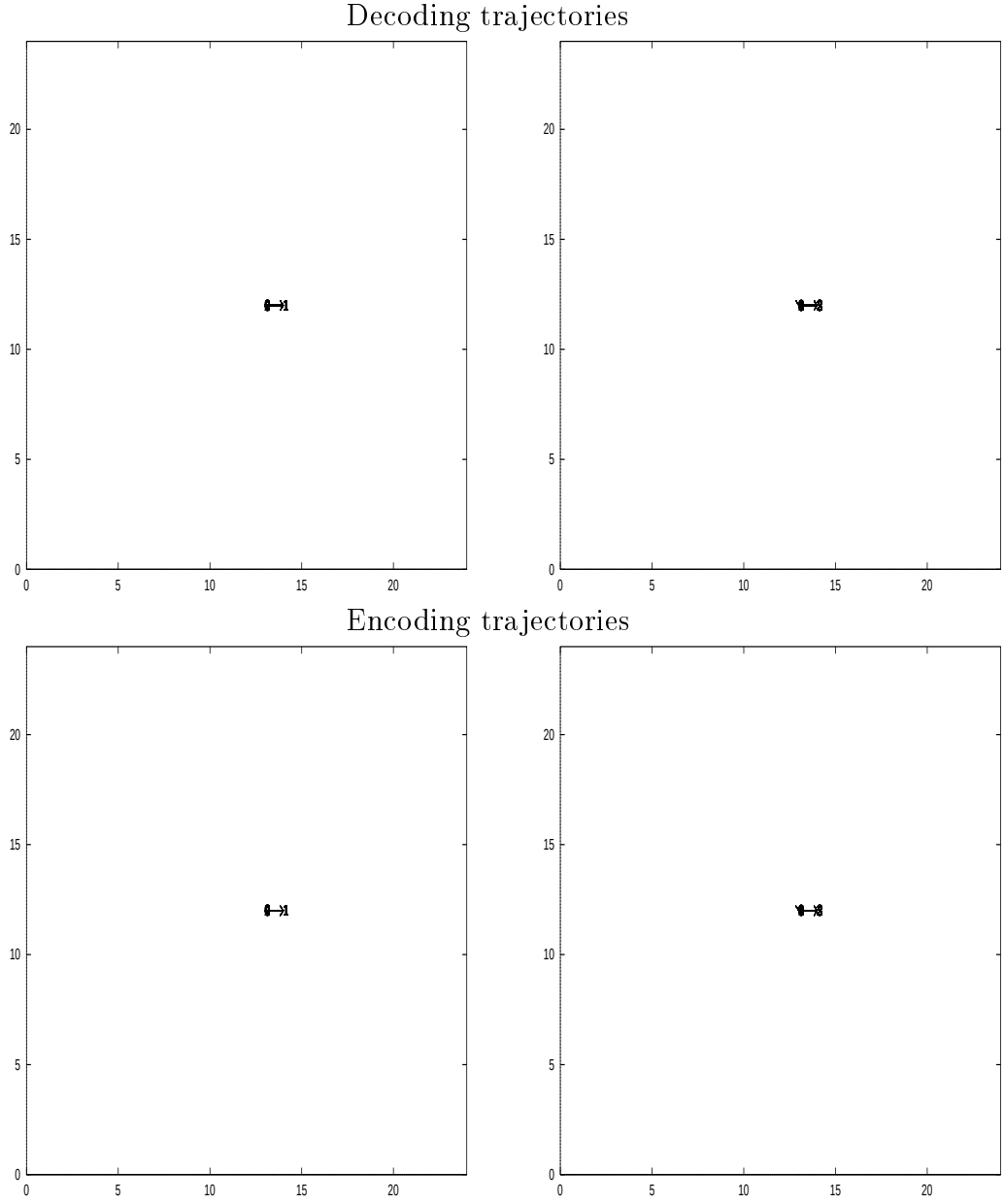

Figure 5.84: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation and Kwasny and Kalman's training after training to a tolerance of 0.04. Active sentences on the left, passive on the right.

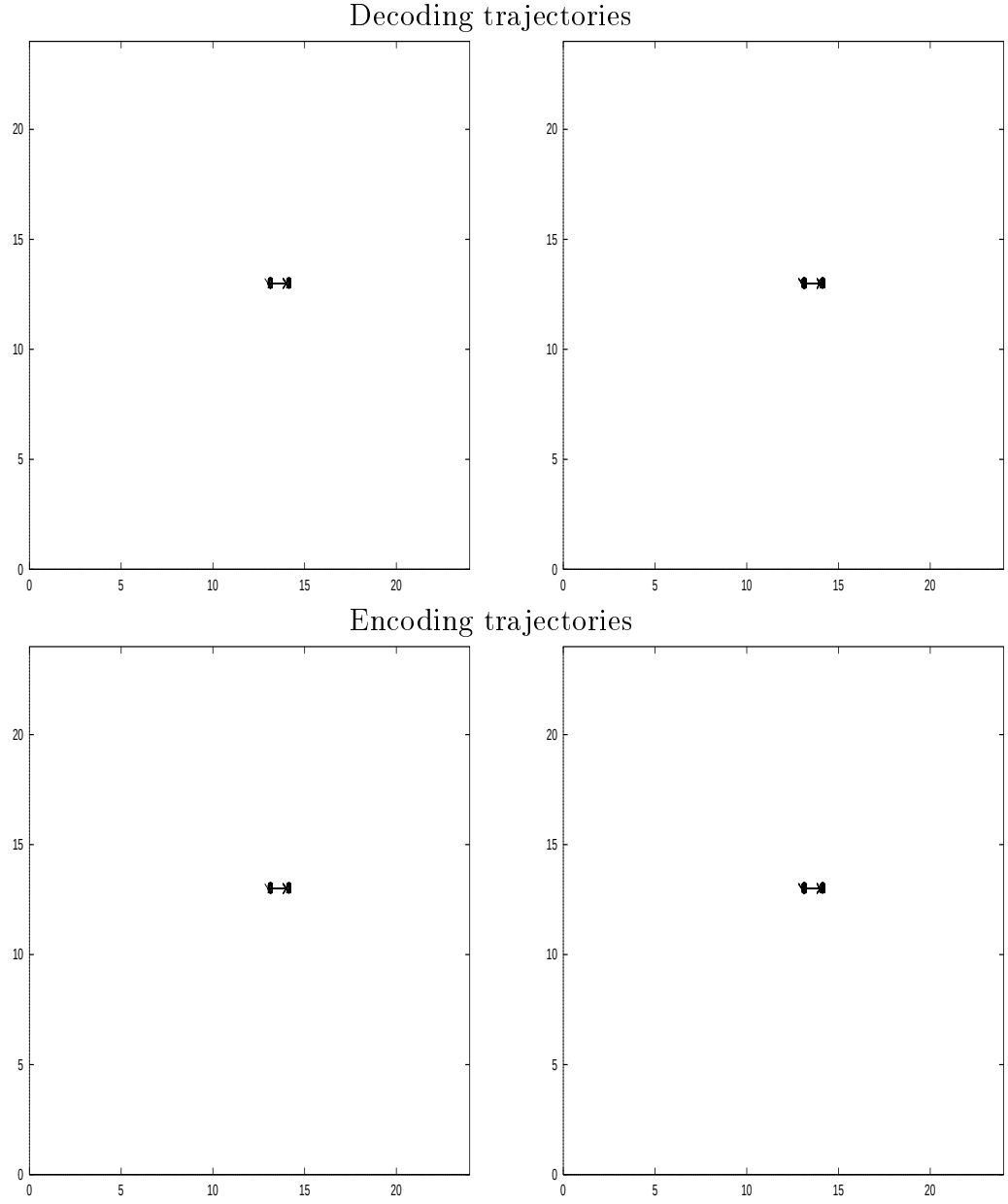

Figure 5.85: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 1 unit representation and Kwasny and Kalman's training after training has completed. Active sentences on the left, passive on the right.

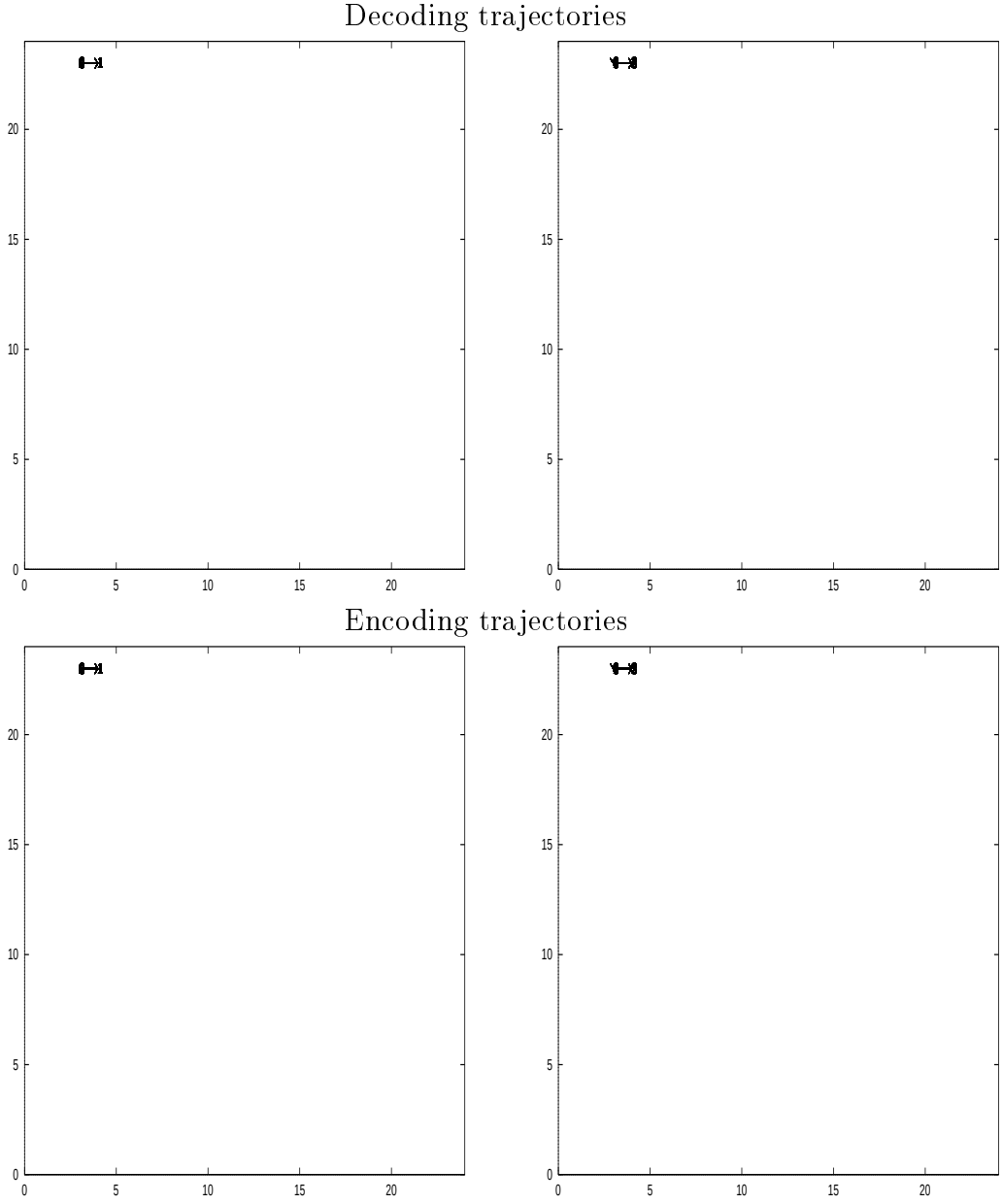

Figure 5.86: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation and Kwasny and Kalman's training after training to a tolerance of 0.16. Active sentences on the left, passive on the right.

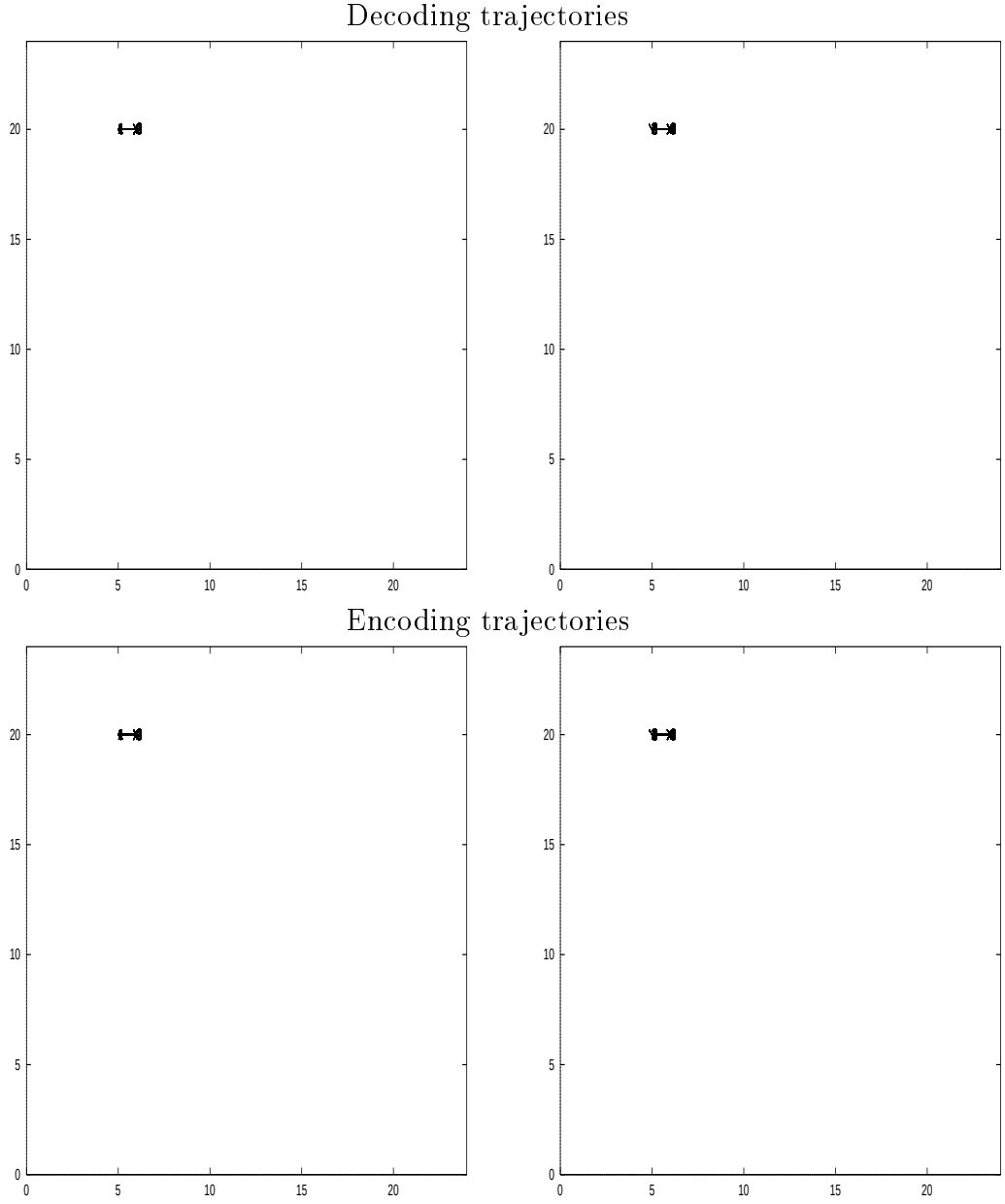

Figure 5.87: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation and Kwasny and Kalman's training after training to a tolerance of 0.04. Active sentences on the left, passive on the right.

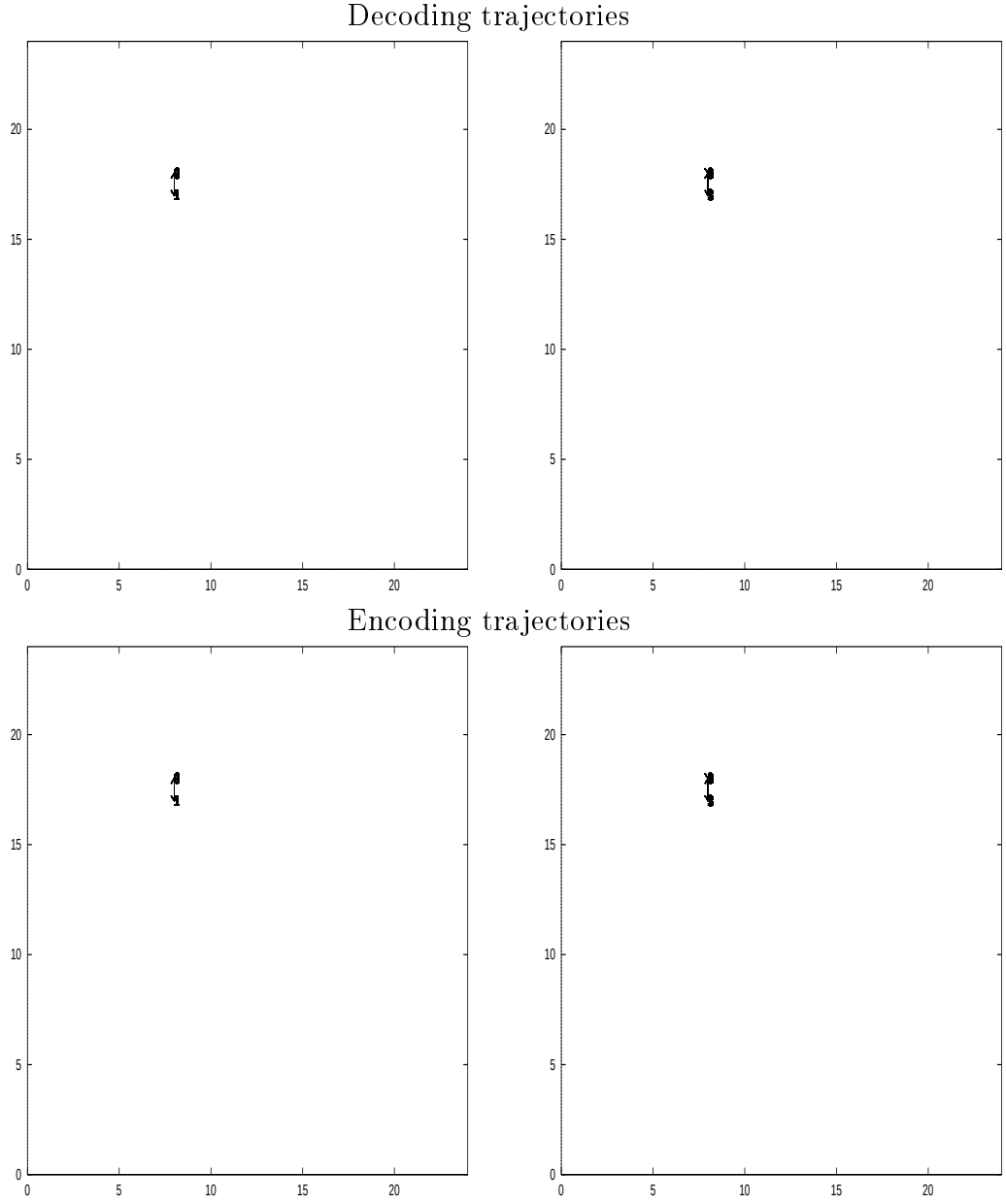

Figure 5.88: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 1 unit representation and Kwasny and Kalman's training after training has completed. Active sentences on the left, passive on the right.

5.3.4 Utilisation of hyperspace

This section presents the results of computing the utilisation of hyperspace by the SRAAMs trained to learn 20 and 130 sequences presented in Chapter 4, Section 4.5.3. The graphs presented here depict the maximum, average and minimum activation of the units in the hidden layer patterns produced during encoding and decoding the sequences in the training set for each network. They thus depict the area of hyperspace utilised by the SRAAMs in organising the hidden-layer states.

Networks trained by back-propagation

Figures 5.89 to 5.90 show the range of activations generated by the hidden layer during encoding and decoding for the networks trained by back-propagation using sigmoidal units and the 1 unit representation. Generally speaking the range of activations generated for each unit in the hidden-layer spans the bulk of the range of activations that the sigmoid function can produce, i.e. the range $(0, 1)$, with a few units where the range of activations is curtailed. The plots for the networks trained on the 2 unit representation with sigmoidal units can be found in Appendix D, Section D.1. Most of these plots are similar in form to the equivalent plots for the 1 unit representation. However for the network trained on the 2 unit representation, with 130 sequences after 100 iterations the range of values taken on by all units is constrained to a small fraction of the possible range. Note that with the encoding and decoding trajectories plotted on the SOMs this was one of the networks where the set of hidden-layer states mapped onto just two points, and thus this provides an explanation for why that happened — all the points where closely clustered together. However at the completion of training the range of values spanned suggests fairly well spread out hidden-layer states within hyperspace.

Figures 5.91 to 5.92 and Figures D.3 to D.4 from Appendix D, Section D.1 depict the range of activations produced for the networks trained by back-propagation using hyperbolic tangent units. The general pattern is similar to that for the sigmoidal units. However some of these networks had the hidden-layer states map to only two points when plotted on the SOMs, but without the narrower activation ranges to explain this as found when this happened with the sigmoidal units. However the distribution of the hidden-layer states within the ranges of activation depicted above is not known: it may be that

the points are closely clustered in two groups, one near the maximum and the other near the minimum patterns of activation when the collapse to two points occurs on the SOM.

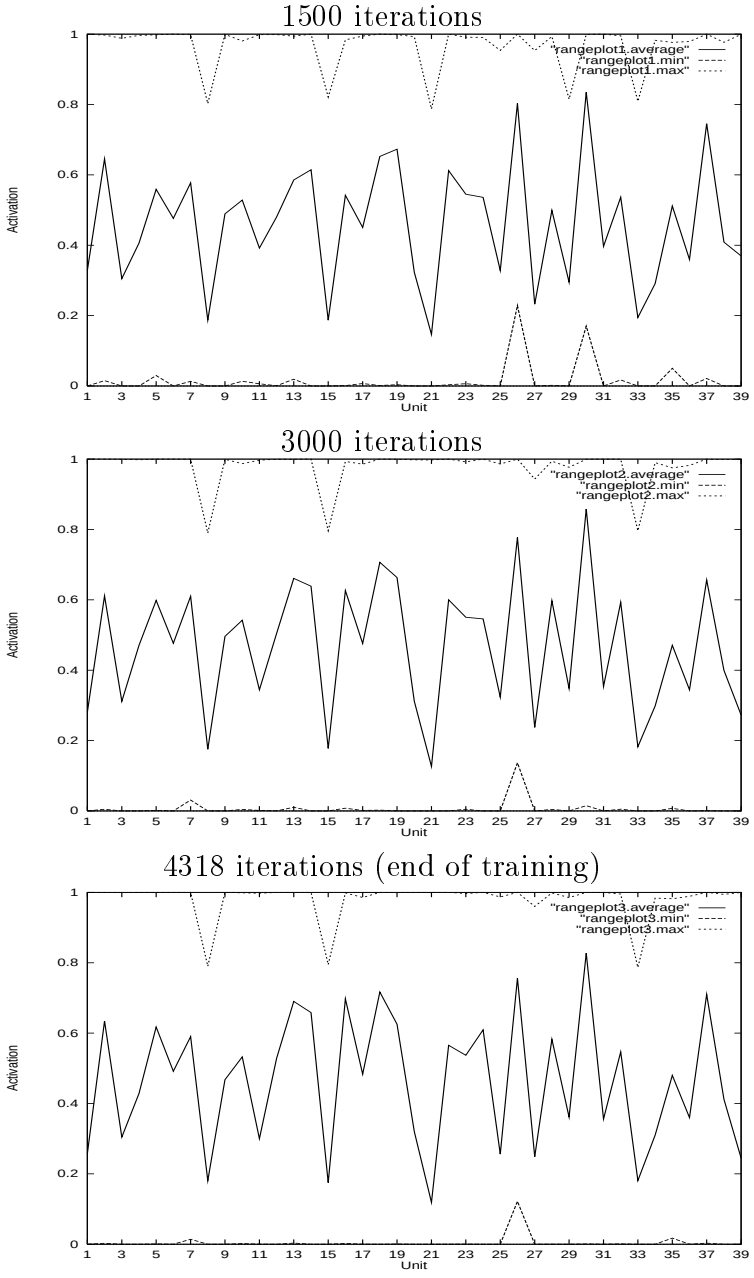

Figure 5.89: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 20 sequences with sigmoidal units and the 1 unit representation using back-propagation.

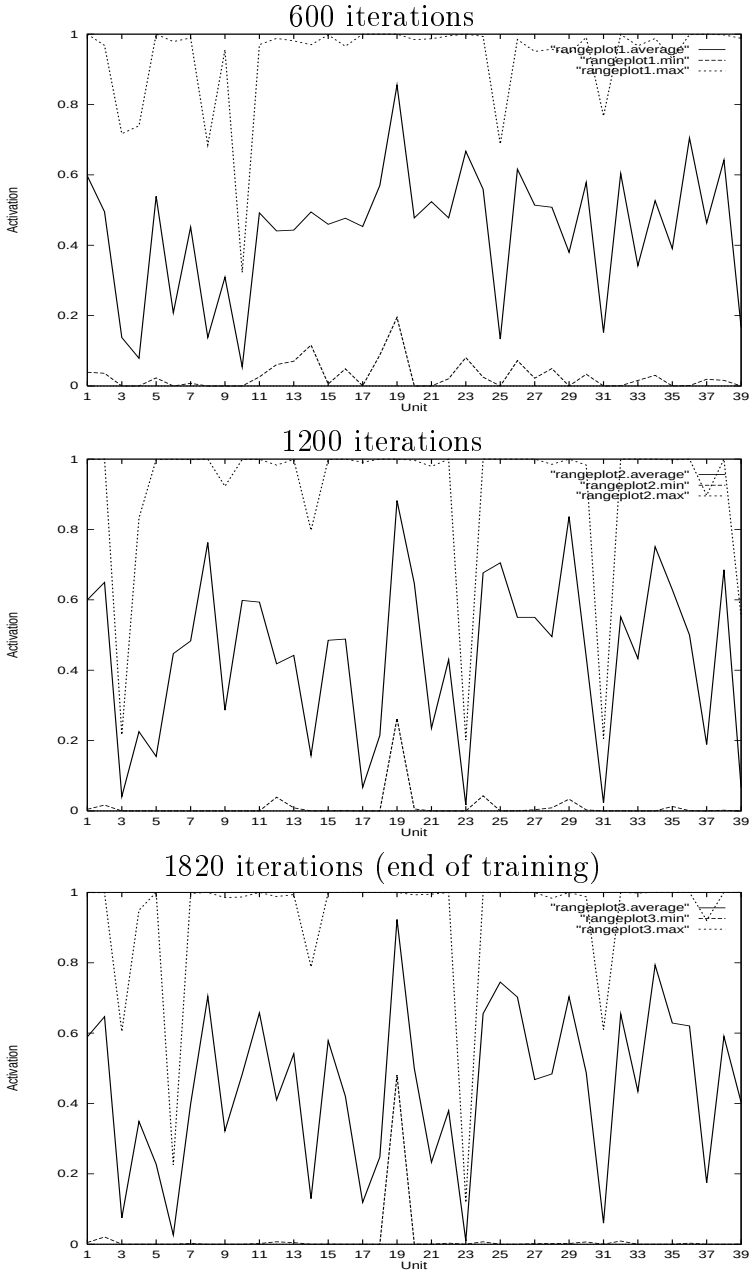

Figure 5.90: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 130 sequences with sigmoidal units and the 1 unit representation using back-propagation.

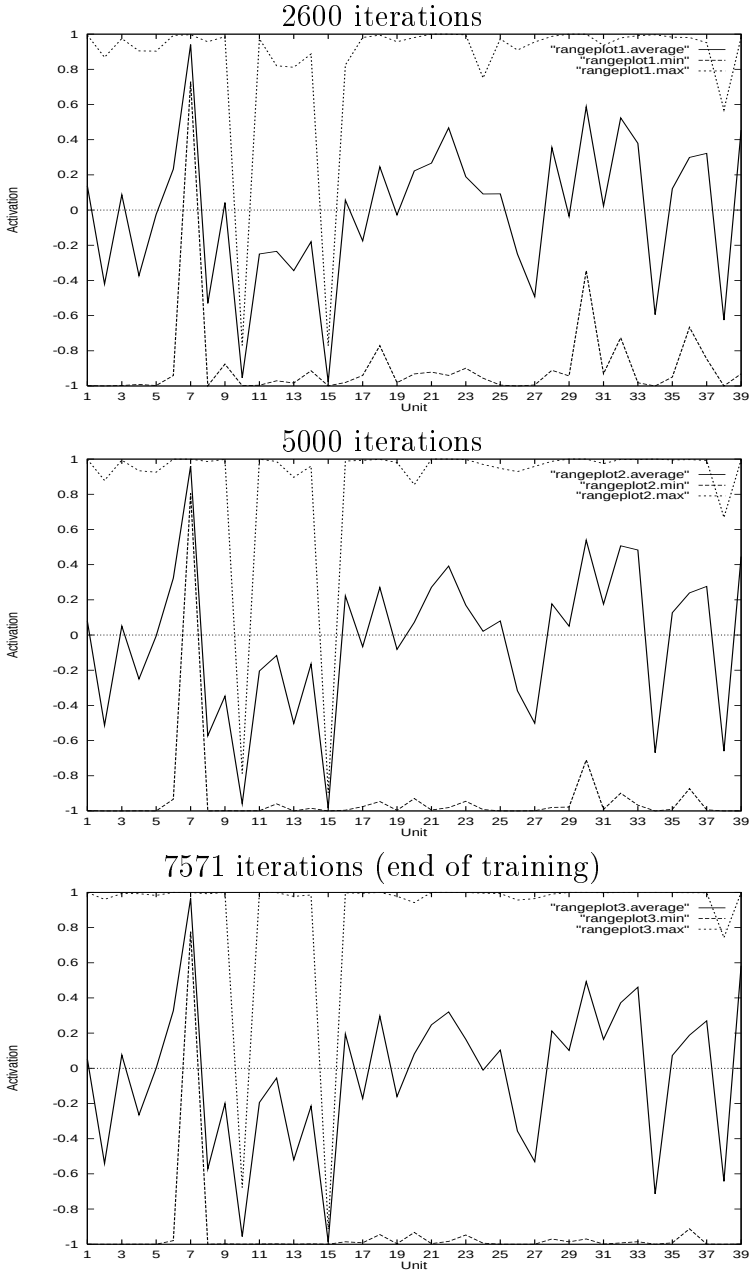

Figure 5.91: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 20 sequences with hyperbolic tangent units and the 1 unit representation using back-propagation.

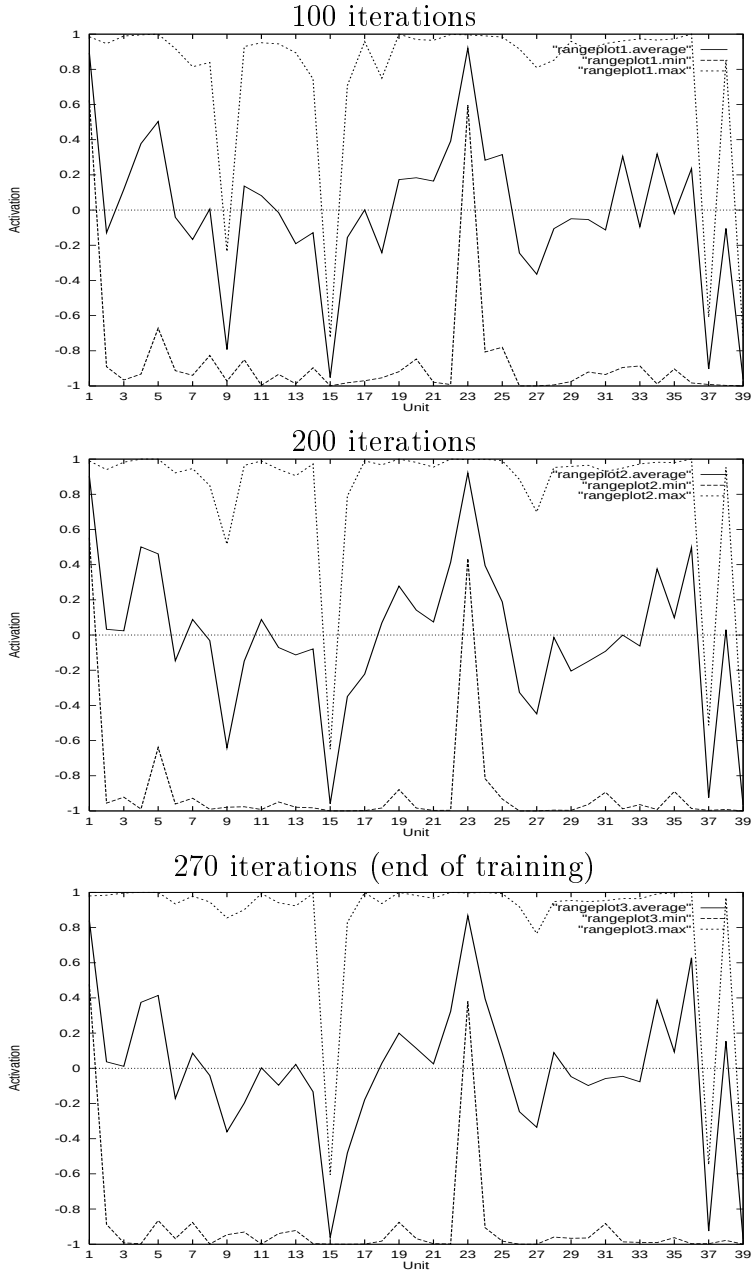

Figure 5.92: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 130 sequences with hyperbolic tangent units and the 1 unit representation using back-propagation.

Networks trained by Kwasny and Kalman's method

Figures 5.93 to 5.94 depict the range of activations generated during encoding and decoding for the networks trained with Kwasny and Kalman's method on the 1 unit representation. The results for the 2 unit representation are presented in Appendix D, Section D.2. Here the range of activations generated is much lower than for most of the back-propagation networks. Without exception the encoding and decoding trajectories of these networks mapped to just two points in the previous section, again in line with the explanations for this phenomenon given for some of the networks trained by back-propagation. However the narrower range of activations may be due to the smaller size of the SRAAM used for Kwasny and Kalman's method rather than the training method *per se.* To test this possibility, the range of activation generated during encoding and decoding was computed for the 39 unit SRAAMs trained by Kwasny and Kalman's method in Chapter 3. Figure 5.95 presents the results. The range of activations generated for the 1 unit representations is obviously narrower than for the back-propagation networks above, but wider than for the networks trained via Kwasny and Kalman's method. For the 2 unit representation the range of activations was wider than the 1 unit representations but still on close inspection slightly narrower than for the networks trained with back-propagation. More units have a narrow range than with the back-propagation networks and the rest of the units do not span the full range of the activation function as often as with back-propagation. Thus this is evidence that the hidden-layer patterns are packed somewhat more closely together for Kwasny and Kalman's method than for back-propagation. It may be that the distribution of the hidden-layer patterns is tighter than the range of activations suggests, but a more detailed analysis would be needed to confirm this.

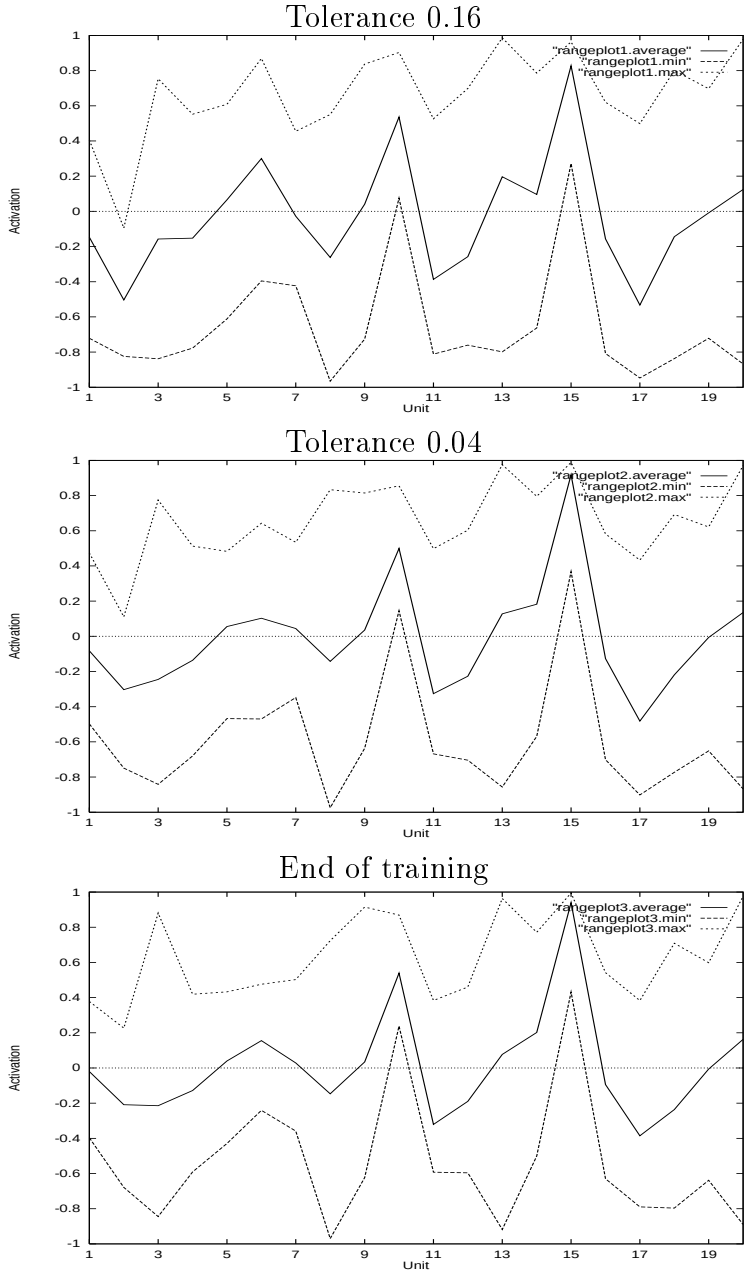

Figure 5.93: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 20 sequences with the 1 unit representation using Kwasny and Kalman's training method.

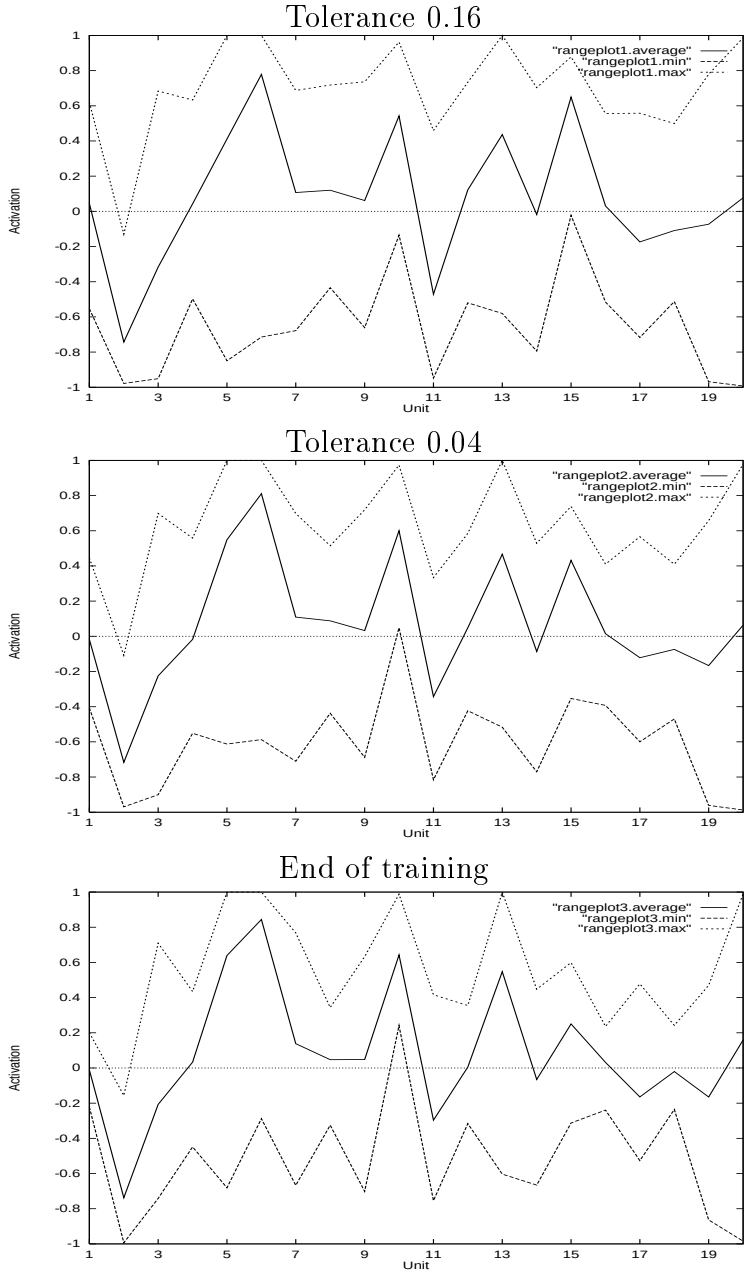

Figure 5.94: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 130 sequences with the 1 unit representation using Kwasny and Kalman's training method.

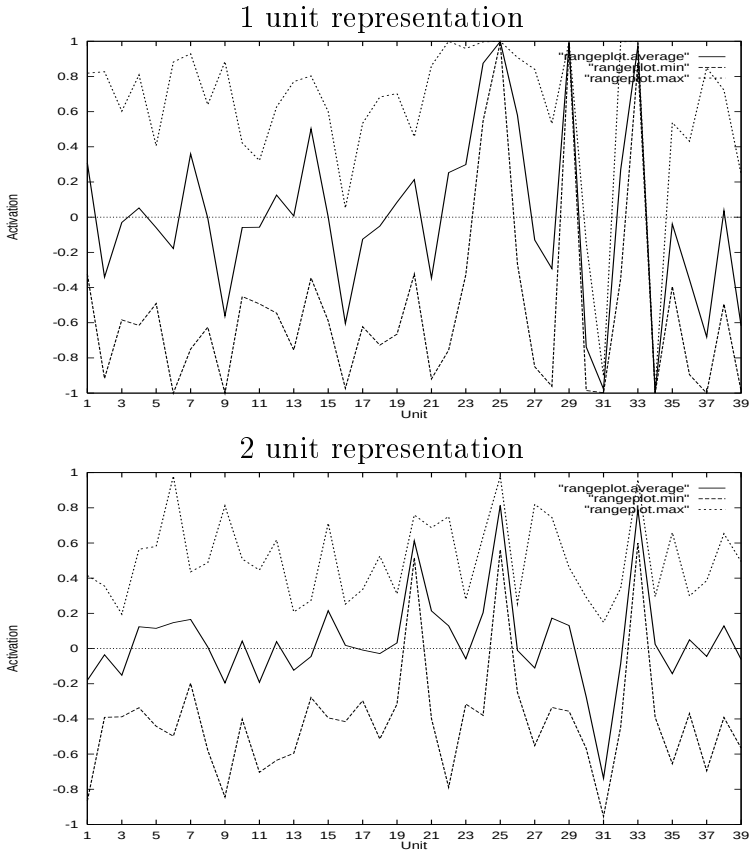

Figure 5.95: The range of hidden-layer activations generated during encoding and decoding, for the SRAAMs with 39 unit hidden layers from Chapter 3, trained on 130 sequences using Kwasny and Kalman's training method at the end of training.

5.4 Discussion

The analyses presented above produced the following findings:

- The SRAAMs cluster the sequences by two strategies; clustering by whether the sequence is an active or passive sentence and clustering by the last symbol of the sequence. With the back-propagation networks the former strategy is dominant and with the networks trained by back-propagation the latter is dominant. The latter strategy is also more evident as the number of sequences learned increases.

- The encodings produced by the SRAAM are sensitive to noise. With back-propagation a 10% perturbation of a single unit in the encoding is sufficient for significant errors to be induced on top of those produced in encoding and decoding processes. With Kwasny and Kalman's method, a 1% perturbation of a single unit is sufficient for significant extra errors to be produced.

- The representations produced by the SRAAM thus seem to be both fragile and, broadly speaking, distributed; lesioning one unit is often sufficient to cause errors across all the slots, though the error levels generated will vary. The most deeply embedded information is usually the most severely affected by the lesioning.

- The trajectories produced by the SRAAM displayed the following characteristics: hidden layer states for the same position in different sequences cluster together; there is a tendency for the trajectories to map to just two points, especially for networks trained by Kwasny and Kalman's method; the similarity in the encoding and decoding trajectories of the active sentences is usually higher than with passive sentences, except where the trajectories all map to just two points.

- The range of activations produced during encoding and decoding suggests the hidden layer states produced by networks trained by back-propagation utilise most of the available hyperspace, but with the networks trained by Kwasny and Kalman's method the available hyperspace is not as extensively utilised.

These findings provide an explanation for the difficulties in training. With Kwasny and Kalman's method, the hidden layer states get packed into a small

area of hyperspace and the network therefore has to make its distinctions between sequences, and between words within sequences, on patterns which are very similar. Thus it is not surprising that deeply embedded information is corrupted. This explanation is backed up by both the sensitivity to noise, the tendency of the encoding and decoding trajectories to map to just 2 points and the narrow range of activations used. With back-propagation the available hyperspace seems better utilised as evidenced by the wider range of activations generated during encoding and decoding and the decreased tendency of the trajectories to map to just 2 points. The sensitivity to noise is lower than with Kwasny and Kalman's method but still fairly high — only a 10% perturbation of a single unit is required to cause significant errors during encoding and decoding. This can be explained by the fact that the hidden layer states for the same position in different sequences cluster together; the states for the final position (the point at which encoding finishes and decoding starts) will be closely clustered together even if otherwise the hidden-layer states are spread out. Thus errors introduced into the patterns in this state are likely to corrupt the patterns and may move the pattern outside of the cluster.

When taken together with the results from Chapter 4 the picture is of solutions that are difficult to find (at least with back-propagation), a rugged error landscape and partial solutions that are sensitive to noise both in the weights and in the hidden-layer patterns. The tendency for the hidden-layer patterns to cluster together may explain why the solution is difficult to find. With the hand-derived solutions the activations produced in the hidden layer during encoding and decoding are all close to the extremes (and in some cases the mid-point) of the sigmoid or hyperbolic tangent functions by design. This results in hidden-layer patterns that are distributed around the boundaries of the available hyperspace. Thus the patterns are more spread out than those produced during training. This suggests that if some way could be found to keep the hidden-layer patterns separate it may improve training. However, on the face of it, the performance of the networks suggest the opposite — Kwasny and Kalman's training is by far the more successful of the two methods employed and it produces hidden layer patterns that are closely clustered together. These points can be reconciled however. That the range of hidden-layer activations, the clustering strategies and organisation of the encoding and decoding trajectories show no strong trends during training suggests that the basic strategy for organising the hidden-layer patterns is adopted early in training and then refined with extra training. It may be

that Kwasny and Kalman's method is better at making this (or any) strategy work than back-propagation and that if the basic strategy is changed to try and keep the patterns separate, easier training, better performance and more robust representations might result. This hypothesis is backed up by the good performance of Kwasny and Kalman's method when started from the hand-derived solutions with noise added to the weights. The starting point in this case will already exhibit some of the features of the hand-derived solution — a solution which has widely spaced hidden-layer patterns — and the training will thus refine the basic strategy of that solution. To test this hypothesis the range of activations produced by the hand-derived solutions for SRAAMs with 20 hidden units which had 10% noise added and were trained with Kwasny and Kalman's method were computed. The results are plotted in Figure 5.96. As can be seen the range of activations produced is far wider than with the networks trained from a random starting point. It is also worth noting though that the range of activations is narrower for some units than would be the case with the original hand-derived solution suggesting a different error-free solution has been found[2]. The tendency of Kwasny and Kalman's method to otherwise cluster the hidden-layer states together from early in training also explains why it fails to find the solution from the normal random distribution of weights around zero. This tendency may get in the way of what is needed to find an error-free solution.

Finally, the fact that the representations are *distributed* **and** *fragile* is in direct contrast to the common assumption that distributed representations are robust representations (e.g. Sharkey and Reilly (1992) describe distributed representations as being "more resistant to disruption" than localist representations). The results reported here show that a distributed representation need not be robust and may actually be highly sensitive to noise (e.g. the representations developed using Kwasny and Kalman's training), and thus demonstrates that one cannot assume that a representation is robust simply because it is distributed. This is not to say that distributed representations cannot be robust, merely that their distributedness does not necessarily guarantee it.

[2]Recall that the solutions employed internal representations consisting of "1"s, "0.5"s and "0s" or "1"s, "0"s and "-1"s depending on whether sigmoidal or hyperbolic tangent units were in use. The graph clearly indicates a truncation of the range of activations for several units.

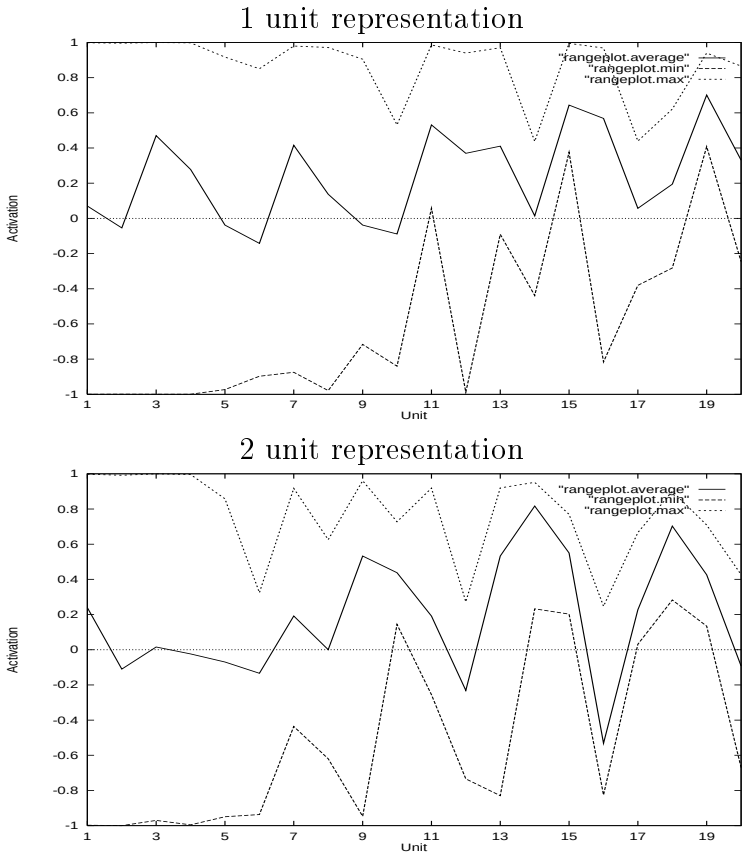

Figure 5.96: The range of hidden-layer activations generated during encoding and decoding, for the 20 unit SRAAMs which were trained from the weights from hand-derived solutions with 10% noise added using Kwasny and Kalman's method.

Chapter 6

Implications: The SRAAM and Holistic Computation

6.1 What has been achieved?

Chapter 1 listed the following aims for this thesis:

1. To clarify the nature and definition of holistic computation by answering the following questions:
   - What is holistic computation?
   - In the light of Balogh's critique of Chalmers' work (Balogh, 1994), does the work which claims to demonstrate holistic computation in fact do so?
   - Are distributed or holistic representations necessary for holistic computation?
   - Is holistic computation unique to connectionism?
2. To investigate how well the representations generated by the Sequential RAAM can support the holistic processing of symbol structures as follows:
   - An attempt to recreate Chalmers' (1990a) experiment using the Sequential RAAM.

- Analysing the behaviour of the SRAAM in the above experiment to investigate whether the technique can scale up and whether it can be improved.

The first of these aims was addressed in Chapter 2. It was argued there that holistic computation had been defined in a vague and sometimes confused manner in the literature and proposed a new definition to clarify the issues involved, thus making a start at least on the critical examination of the concept of holistic computation argued for in Chapter 1. Its main conclusions were:

- A holistic computation is "any computational process that can act on all of the constituents of an object simultaneously out with the need to search to access those constituents". This definition makes it explicit what constitutes a holistic computation and answers the first question.

- Most types of neural network perform a trivial form of holistic computation. The input vectors are transformed holistically into the output vectors. However the bulk of the literature on holistic computation has as its main focus the holistic processing of symbolic structures, referred to here as "holistic symbol processing".

- Neither holistic computation nor the holistic symbol processing require a holistic representation, contrary to the often implicit assumptions of much of the literature, answering the third question.

- Confluent inference is a separate phenomenon from holistic symbol processing, and by itself involves only the trivial form of holistic computation mentioned above.

- It follows from the above that Chalmers did demonstrate holistic symbol processing as does most of the RAAM based work being performed. However Chrisman and Ho and Chan demonstrate only confluent inference and not holistic symbol processing. This is the answer to the second question.

- In principle at least, holistic computation and holistic symbol processing are not restricted to connectionist systems, which answers the final question.

By drawing out these points, it is hoped the work presented in Chapter 2 will lead to a better understanding of holistic computation and holistic symbol processing and therefore what some connectionists have been trying to exploit. At any rate the nature of holistic computation has been clarified and the questions listed above answered.

The second aim was addressed in Chapters 3, 4 and 5. Chapter 3 presented an attempt to recreate Chalmers' experiment but using the SRAAM in place of the RAAM, whilst Chapters 4 and 5 analysed the performance of the SRAAM. The following are the main findings from these chapters:

- The SRAAM was unable to learn to encode and decode 130 of the sequences from Chalmers' set with a hidden-layer of up to 39 units, although one of the training runs managed to encode and decode over 85% of the entire set of 250 sequences without error. However the inability to learn to encode and decode its training set without error and the 50.8% or lower performance of the rest of the training runs suggests that the SRAAM would not be viable for larger tasks even so.

- The partial solutions found by the SRAAM were sensitive to noise both in the weights and the hidden-layer activations.

- The SRAAM was developing a distributed but fragile representation.

- It was shown that an error-free solution is known to exist for SRAAMs of 20 or more hidden units, but that under back-propagation the area of weight-space it and any nearby error-free solutions that exist are accessible from is small. Under Kwasny and Kalman's method the area was much larger, but still apparently inaccessible from the normal starting points of weights distributed uniformly in a range centred around zero.

- It was also shown that given a set of sequences of maximum length L, made up from a set of M symbols, one can construct a working SRAAM with a hidden layer of $L \times \lceil \lg(M) \rceil$ hidden units, if the symbols are represented by orthogonal patterns.

The overall message from these findings is rather negative for the SRAAM in its current form, since it was unable to learn the task and suffered from high sensitivity to noise. The findings did suggest that it may be possible to improve the performance of the SRAAM, however. The fact that error-free

solutions do exist for 20 or more hidden units is one indication that this may be possible. Furthermore, the SRAAM seems to settle on a strategy for organising the hidden-layer patterns early in training and then subsequently refines it, as evidenced by the lack of a strong trend during training in the clustering strategies or the organisation of the encoding and decoding trajectories when measured at three points in training. It was noted that the encodings for the complete sequences were clustered together (as evidenced both by sensitivity to noise and by the encoding and decoding trajectories) even if the remaining hidden-layer states appeared spread out. If some way could be found to keep the hidden-layer states well separated (closer in organisation to the hand-derived solutions than the partial solutions being learned) it may help training. That Kwasny and Kalman's network had good performance on finding error free solutions when noise had been added to the hand-derived solutions and that the resulting hidden-layer states were more spread out than for the partial solutions seems to confirm that if the basic strategy spreads the hidden-layer states out in hyperspace, then it may be possible to learn the encoding and decoding task without error. It appears that the SRAAM in its current form is not as effective a vehicle for holistic symbol processing as the RAAM in that it requires much larger representations and is sensitive to noise, but it might be possible to improve the SRAAM if a way can be found to keep the hidden-layer patterns separated during training. Thus it appears that in its current form the SRAAM will not scale up well as it does not even match the performance of the RAAM, but there is scope for improving its performance.

6.2 Novel features of this work.

There are some novel aspects to this work when compared to earlier work on holistic computation and earlier analyses of connectionist representations.

Firstly whilst analyses of encoding and decoding trajectories have been performed before these have either involved plotting the hidden layer patterns produced directly, which is impractical for hidden layers greater than 3 units in size, or have plotted the top 2 or 3 principal components of the patterns which risks losing information if they do not account for all of the variation in the data. This work's novel use of a SOM to plot the trajectories is intended to address these deficiencies in earlier work, since the SOM can take into account all of the variation in the data when making its map. When taken

along with the range of activations used in each unit, this provides more information than either of the earlier methods.

Secondly the author is not aware of any work which specifically sets out to test the sensitivity of a compositional connectionist representation (such as the RAAM or SRAAM) to disruption of the activations of the units that form the representation, as is done with the representational lesioning here. The technique itself includes a novel adaptation of Balogh's single unit lesioning technique to allow the sensitivity to noise to be accurately measured. Earlier analyses have only looked at whether a representation is localist or distributed and have ignored whether it is sensitive to noise or robust perhaps because of the common assumption that distributed representations are robust. The sensitivity to noise of a representation is important since a fragile representation may have problems scaling to large, complex tasks, and since neural networks can rarely achieve perfectly correct output. As noted in Chapter 5 the fact that the SRAAM developed a representation that was both distributed and fragile invalidates the assumption of robustness for distributed representations.

6.3 Implications

6.3.1 Holistic Computation

The main implications of the work presented in this thesis for exploiting holistic computation are that holistic symbol processing does not require holistic representations; that it is not restricted to connectionist systems (at least in principle); that confluent inference is a separate phenomenon, and that a lot more work is needed to find out how best to effectively exploit holistic computation. With regard to this last point, the work on the SRAAM suggests that it is not the best vehicle for holistic symbol processing, contrary to the suggestions of Kwasny and Kalman that it may have advantages over the RAAM, although there may be modifications which will improve its performance and thus make it more attractive.

6.3.2 Connectionism

The work presented here suggests that whilst connectionist systems may in principle be well suited to holistic symbol processing (since they process their

input vectors holistically), holistic symbol processing is not unique to connectionism. Further the current state of holistic symbol processing suggests more work is needed to exploit holistic symbol processing effectively within connectionist systems. Much of the work that has already been done has been "in principle" work that demonstrates the possibility of holistic symbol processing. There is a need to move beyond this sort of work and to demonstrate complex tasks being performed holistically in order to show that (or determine whether) connectionist methods can be employed effectively for holistic performance of symbolic tasks. The work here has barely touched on this issue, save in so far as it has examined whether the SRAAM might scale up well to complex tasks, where it seems that in its current form at least it probably will not. It remains the case that holistic symbol processing is an interesting possibility that has yet to achieve the power and success of traditional symbolic methods.

On the other hand the work here does illustrate a methodology for evaluating the connectionist representations that have been or are being developed to support holistic symbol processing, which is to not only see whether some task can be successfully performed but to analyse both the weights and the hidden layer patterns produced to see whether there are limitations to the technique, and to see what scope there is for improving performance. An effort to find out what makes the other techniques work (or fail) will improve the understanding of connectionist methods of producing compositional representations and connectionist holistic processing techniques. This would form an important contribution towards understanding how to exploit holistic computation in connectionist systems.

6.3.3 Symbolic approaches

Holistic symbol processing is something traditional symbolic methods do not support, yet it is a new form of symbol processing. However as noted earlier, it is an implication of the discussion in Chapter 2 that this need not be the case in principle. Thus it may be worthwhile investigating whether symbolic methods might be modified to support holistic symbol processing. What would be needed is some way of making the information required to perform the task directly accessible to the processes performing the task. This may require employing vector-based representations as in connectionist networks. If it can yield performance improvements it may be worth investigating.

6.4 Areas for Future Work

Whilst the work in this thesis has achieved the aims set out above, there are a number of ways in which the work could be extended to further the understanding of holistic computation, connectionist representations and the SRAAM itself. This section presents some ideas for further work in line with these aims.

6.4.1 Modifications to the SRAAM

The basic message from the work in this thesis is that the SRAAM, as it currently stands, is not an effective vehicle for holistic symbol processing and that other methods would be preferred for exploiting holistic computation. However it has also been argued that there may be scope for improving the performance of the SRAAM. It may therefore be worth investigating this possibility to see if the SRAAM can be turned into an effective vehicle for holistic symbol processing or to see what lessons might be learned with respect to improving other techniques. A number of suggestions are presented below:

- Often it is the more deeply embedded information that the SRAAM fails to encode and decode correctly. One strategy to improve performance would be to reinforce information added to SRAAM encodings in the early stages of encoding in some way to counter the effect of adding information later on. This could be achieved by adding another feedback loop to the SRAAM, so that rather than just feeding back the hidden-layer from the previous time-step, the hidden-layer from the previous time-step but one is also fed back.

- The analysis suggested that if the hidden-layer patterns could be kept well separated during training it might improve the performance of the SRAAM. One way of testing this hypothesis would be to adapt the context-biasing technique developed by French (1994), for use with SRAAMs to see if this improves training and performance. French modified back-propagation training so that for each pattern, the hidden layer pattern would be modified to increase the distance between it and the previous hidden layer pattern in proportion to how far apart the current target output and the previous one are. The weights between the input layer and hidden layer were modified to "lock in" the

modifications to the hidden layer pattern. The end result was that well-separated distributed representations were developed for the data the network was trained on. This technique was developed on feed-forward networks but could be adapted for the SRAAM fairly easily, to attempt to produce well-separated and well-distributed hidden-layer patterns. If this works, it would reduce interference between the representations and may thus improve the performance of the SRAAM without altering the architecture. This would also confirm the conclusions about the SRAAM presented in this thesis.

- In Chapter 4 it was pointed out that the encoder and decoder are two layer networks, and may have limited power as a result. Adding an additional hidden layer to either or both of the subnetworks, will increase the power of them, and may allow them to learn longer and more complex sequences.

6.4.2 Further analysis of the SRAAM

The analysis presented in this thesis was aimed at understanding how the SRAAM works in order to find out why it failed to learn to encode and decode Chalmers' sentences without error and whether it may be possible to improve the SRAAM's performance. It has shed light on why the SRAAM failed to learn the task, suggesting that it is due to the tendency of the SRAAM to pack the hidden-layer patterns closely together in hyperspace. However there are unanswered questions as to why this tendency exists and under what conditions it is likely to cause problems. Since other researchers have used the SRAAM successfully on some tasks, an investigation into precisely what causes this behaviour and under what conditions it will be a problem may therefore be warranted. Such an investigation may involve further use of the techniques presented here and, if possible, formal analyses of the SRAAM's properties which would complement the experimental analyses used here. Formal analyses may also yield deeper insights into the SRAAM's behaviour than the experimental analysis presented here. The insights gained from such an investigation would be of importance not only to those who wish to use or improve upon the SRAAM, but also to the development of other techniques derived from the RAAM and of course the RAAM itself.

Some preliminary work for the investigation mentioned above would be to find out under what conditions the encoding and decoding trajectories

get mapped to just two or three points when back-propagation is used as the training method, as occurred with the work presented above. It was not possible to fully explain why this was happening with the information made available by the analyses presented here. What is needed is to find out under exactly what conditions this occurs with back-propagation and exactly how the hidden-layer patterns are distributed within hyperspace when this happens. An extensive investigation would involve training runs starting from several different random sets of weights and systematic variation of the learning rate, momentum, hidden-layer sizes and activation functions used.

6.4.3 Analyses of other connectionist representations

There are several connectionist representations for a researcher to choose from if they wish to attempt to exploit holistic computation in some manner. As argued in Chapter 3, many of these techniques have not had extensive investigations into their behaviour performed in order to understand what limitations they may have and what their capacity for supporting holistic symbol processing is. There is thus a need for such investigations to be performed to further understanding of the specific techniques and how they may be developed and to further understanding of how holistic computation can be best exploited. The work presented here illustrates a methodology to employ in such investigations, though where possible it should also be complemented with formal analyses of the techniques as occurred with the Labelling RAAM (Sperduti, 1993b). Considerations of whether the solution for a class of tasks exist in principle, as illustrated in the proofs presented in Chapter 4, may provide a starting point for such formal analyses.

An analysis of the RAAM along the lines suggested above would be particularly relevant, as the SRAAM is a simple variation of the RAAM, the RAAM is currently the most widely used technique for work on holistic symbol processing, and because the RAAM has been used as the basis for deriving several other techniques. Thus developing a thorough understanding of the RAAM would yield insights into both connectionist representations and holistic computation. Such an investigation should also look at what impact the use of confluent inference has on the behaviour of both the RAAM and the SRAAM. The more successful work with the SRAAM has often employed confluent inference, and it has also been used with the RAAM (Niklasson and Bodèn, 1994; Niklasson and Sharkey, 1997).

6.4.4 Improving the analysis techniques

The previous section suggested that the analysis presented here illustrates a methodology for analysing other techniques. However it is not necessarily the case that the analysis should simply be applied to each technique without change. The techniques use here have various limitations and there may be other techniques, or improvements to the techniques used here that one may wish to employ in further analyses. Some suggested improvements to the analysis are presented here:

- Although the broad patterns of clustering, hyperspace utilisation and the encoding and decoding trajectories were extracted, there remained some ambiguity about precisely how the hidden-layer states were organised in hyperspace. The self-organising maps of the encoding and decoding trajectories show the form of organisation but do not give information about where in hyperspace this takes place, hence one reason for including the plotting of the range of activations generated during encoding and decoding. Also where the trajectories mapped to just two points, much of the organisational information is lost, though a higher resolution map might retrieve this information. The plots showing the range of activations generated during encoding and decoding are only of limited usefulness, since the *distribution* of those activations is not measured, thus it only shows the area of hyperspace utilised and not the full extent to which that area is utilised. An improvement to this part of the analysis would add the average distance between each pair of hidden-layer patterns as a more direct measure of how spread out they are, and might also show the range of activations, and the average distance measure, produced for each position in a sequence. This would yield a lot more information about how the hidden-layer patterns are organised in hyperspace to complement the maps of the encoding and decoding trajectories. Higher resolution maps of the trajectories might also yield further information, as would the use of a three dimensional map, though the latter might become difficult to interpret.

- The weight lesioning analysis showed the sensitivity to having a percentage of noise applied uniformly to every weight. It might also be useful to know how sensitive the solutions are to having percentage of the weights removed. The former method was useful since it also indicate how large an area of weight-space the partial solutions were

located in but the latter forms an alternative method of measuring the network's sensitivity to damage.

6.5 Conclusions

The ability of neural networks to process their input vectors holistically has given rise to the possibility of performing symbol processing holistically. The discovery of holistic symbol processing may be one of the more important contributions of connectionist research as it is a form of symbol processing hitherto unsupported by traditional symbolic systems. It therefore both provides a reason for preferring connectionist systems to symbolic systems and the basis for an answer to the charge that connectionist symbol processing mechanisms merely reimplement classical symbolic mechanisms. However in order to be exploited to the full, a clear understanding of holistic computation, holistic symbol processing and what makes them possible needs to be developed. This thesis has sought to clarify the nature of holistic computation and holistic symbol processing. It has further sought to investigate the SRAAM's ability to generate representations that support holistic symbol processing and subsequently to understand why the SRAAM failed to learn the task given to it in Chapter 3 and what lessons can be drawn from this failure. Through these investigations, it is hoped that this thesis has contributed to advancing the understanding of holistic computation, holistic symbol processing and what makes them possible argued for above.

Appendix A

HCA Plots

This appendix contains the dendrograms for the HCA experiments described in Chapter 5. It is split into sections for networks trained via back-propagation (Section A.1) and Kwasny and Kalman's method (Section A.2).

A.1 Back-propagation networks

A.1.1 Fully Trained Networks

Figures A.1 to A.24 are dendrograms from fully trained back-propagation networks.

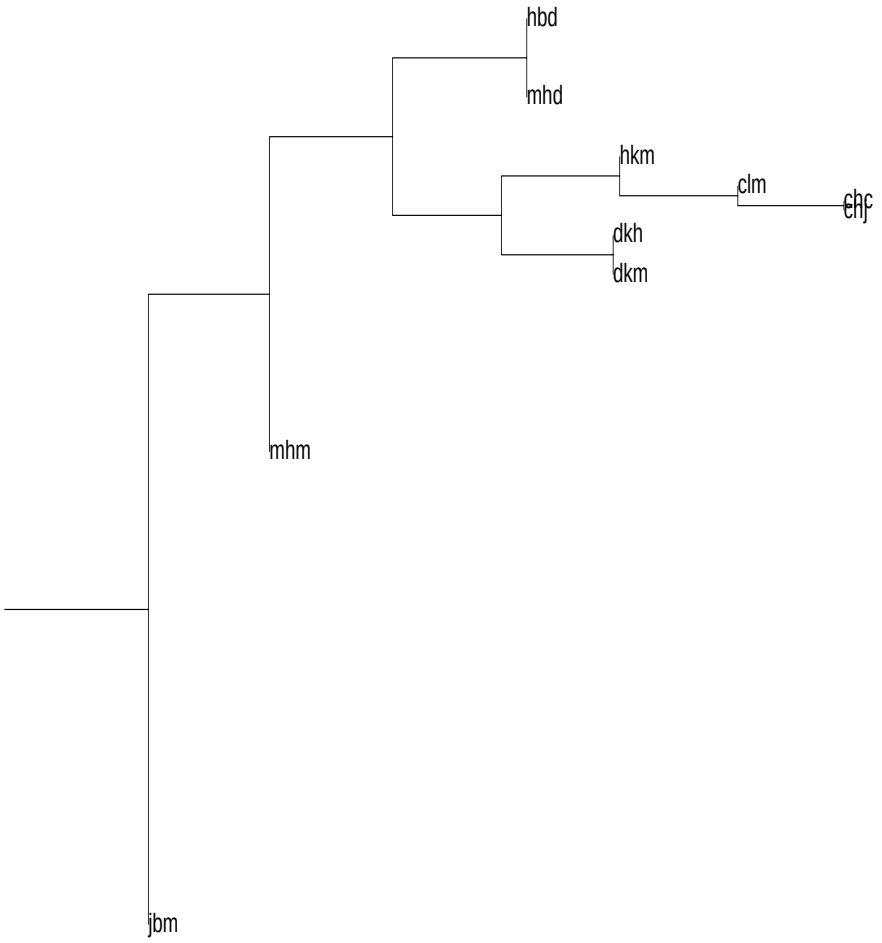

Figure A.1: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 2 unit representation.

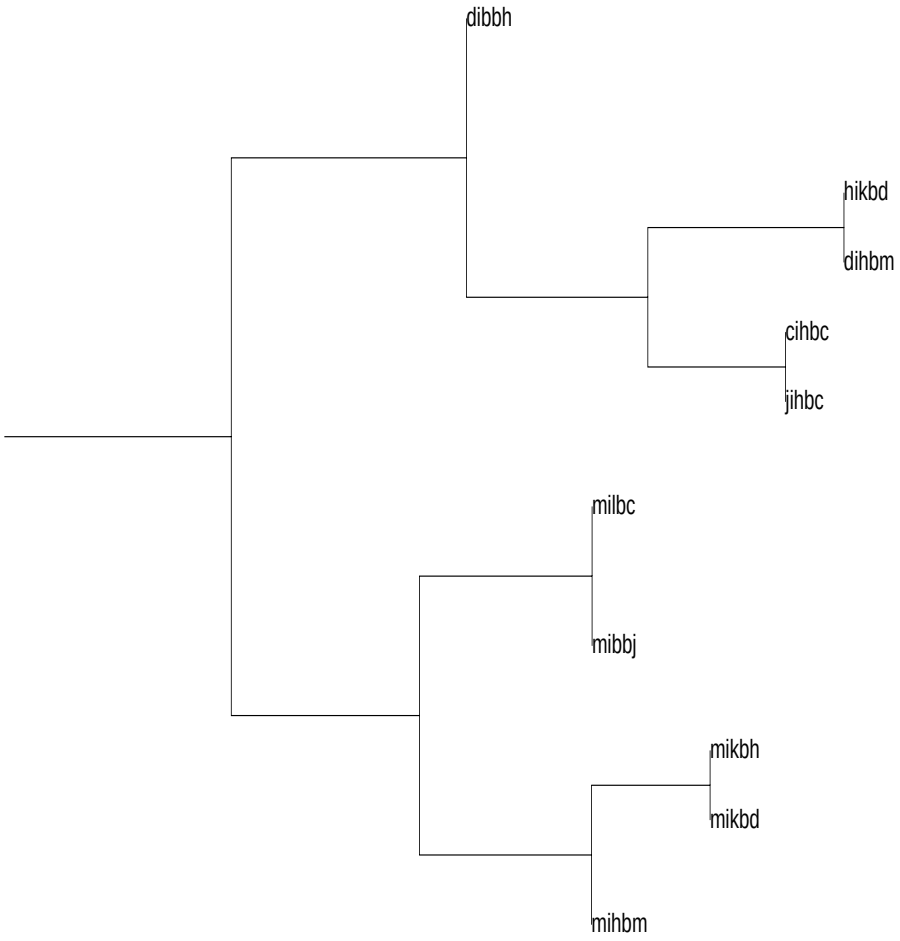

Figure A.2: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 2 unit representation.

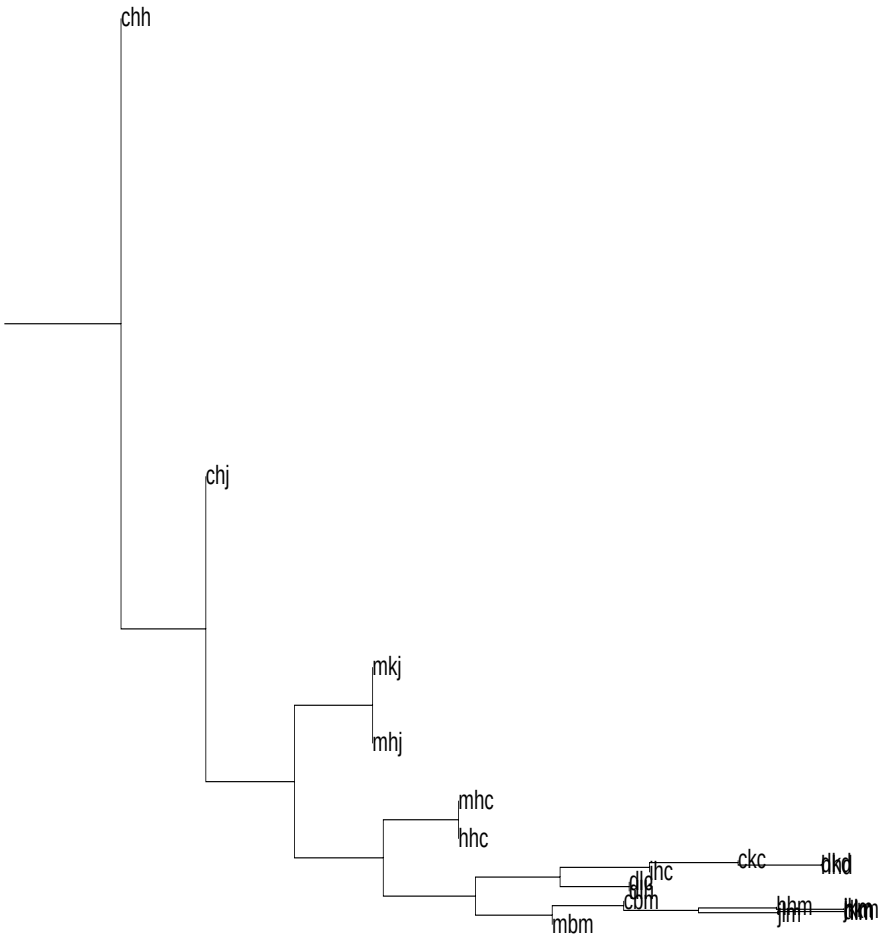

Figure A.3: Top branch of the HCA plot for the SRAAM trained on 40 sequences with back-propagation, sigmoidal units and the 2 unit representation.

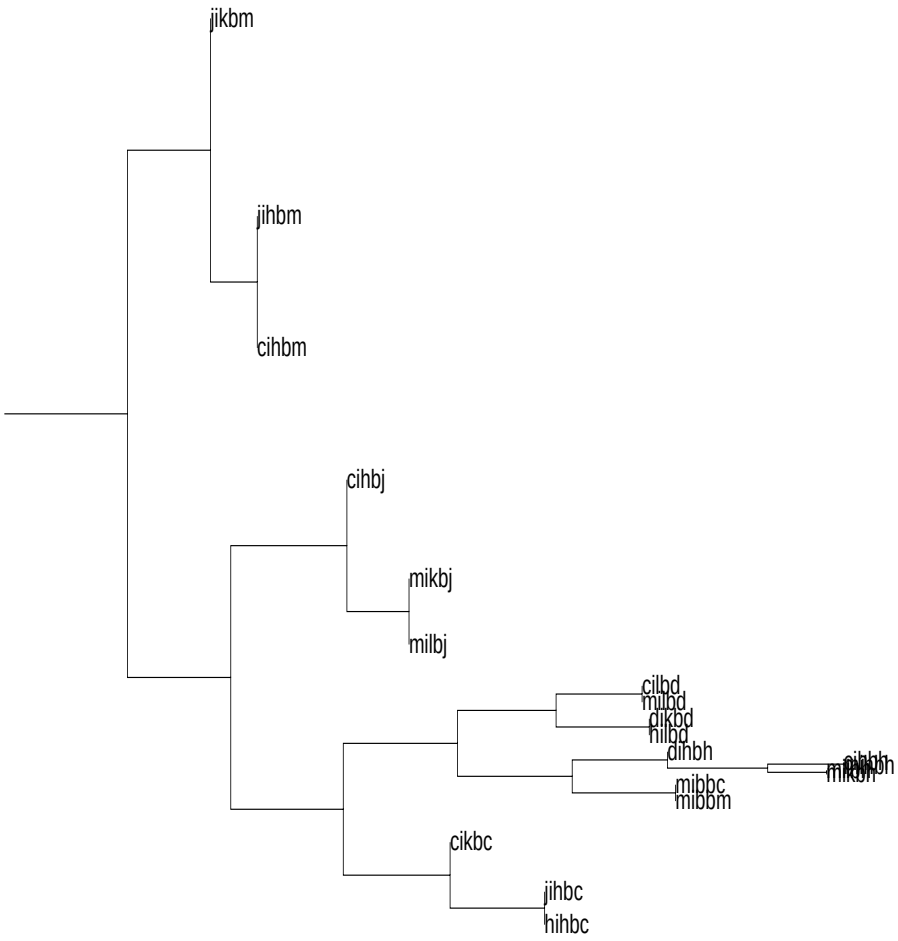

Figure A.4: Bottom branch of the HCA plot for the SRAAM trained on 40 sequences with back-propagation, sigmoidal units and the 2 unit representation.

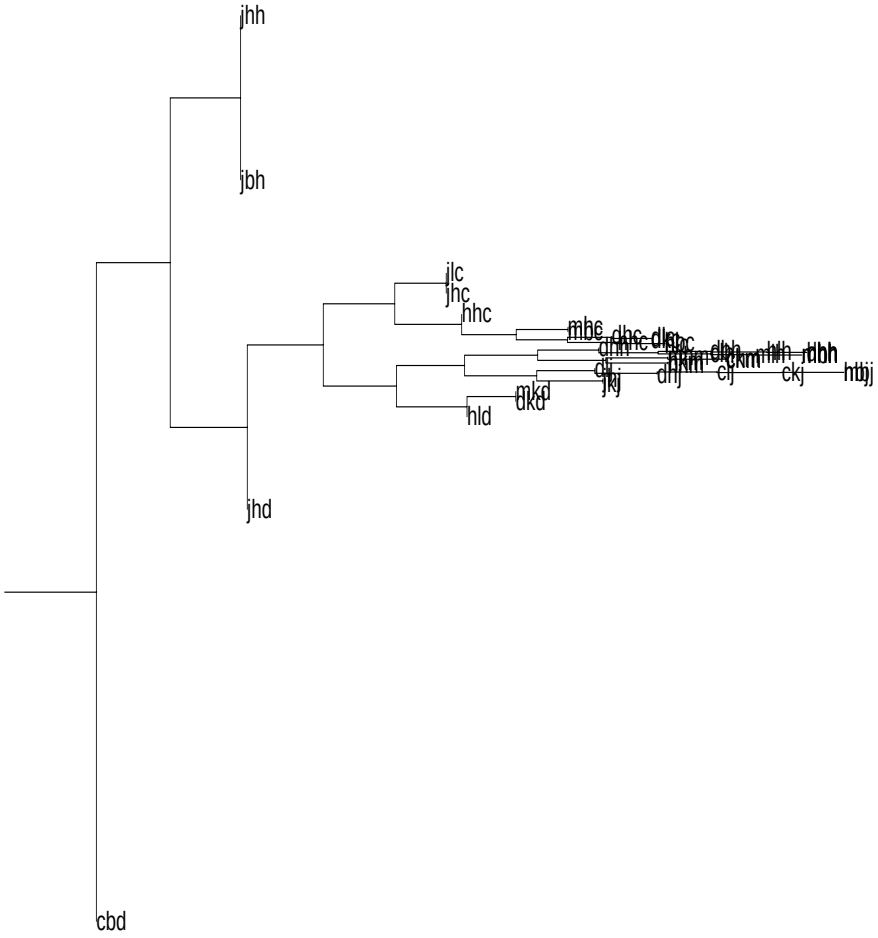

Figure A.5: Top branch of the HCA plot for the SRAAM trained on 80 sequences with back-propagation, sigmoidal units and the 2 unit representation.

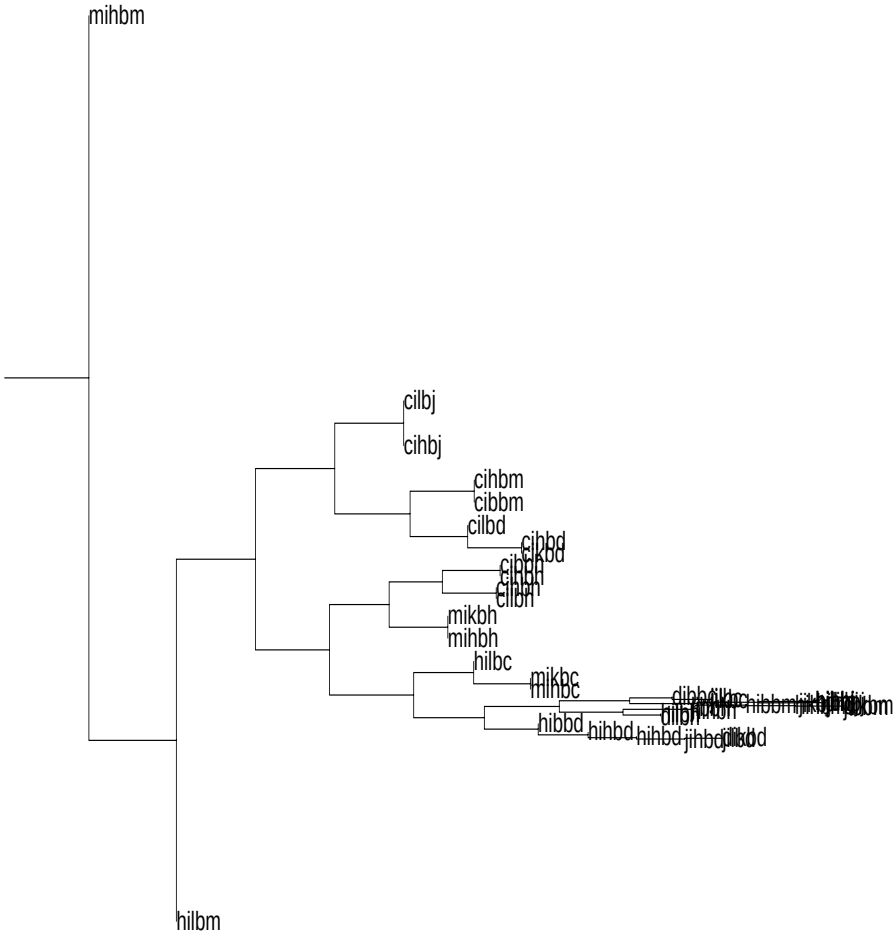

Figure A.6: Bottom branch of the HCA plot for the SRAAM trained on 80 sequences with back-propagation, sigmoidal units and the 2 unit representation.

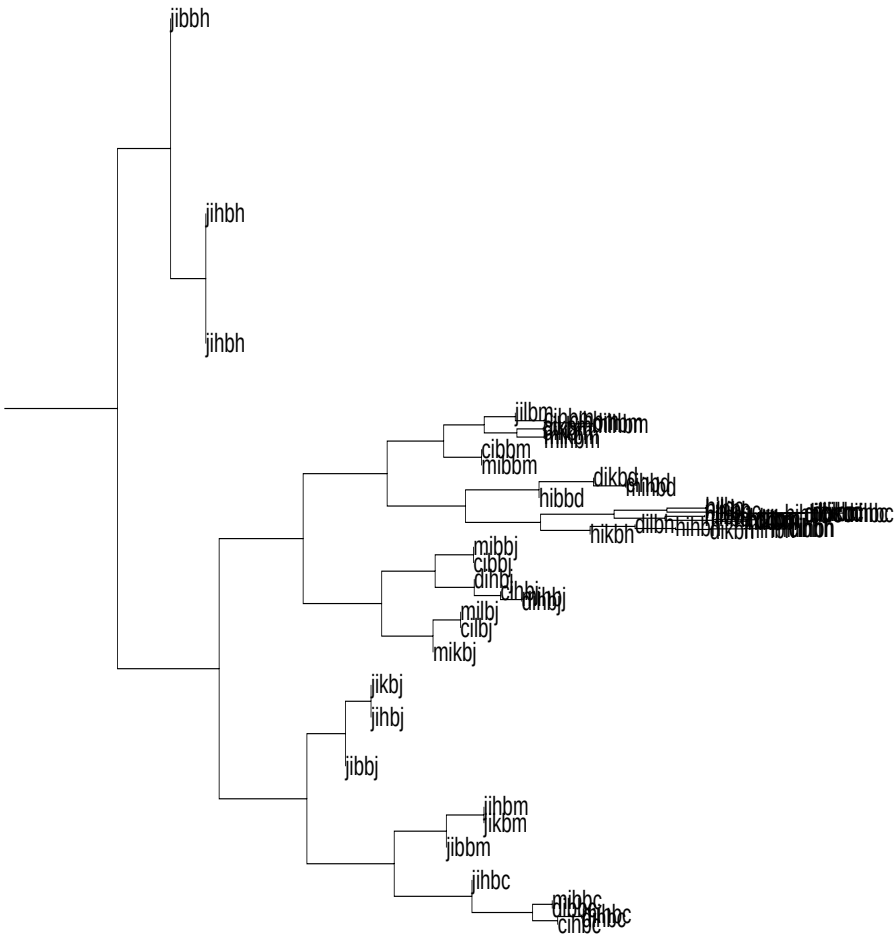

Figure A.7: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 2 unit representation.

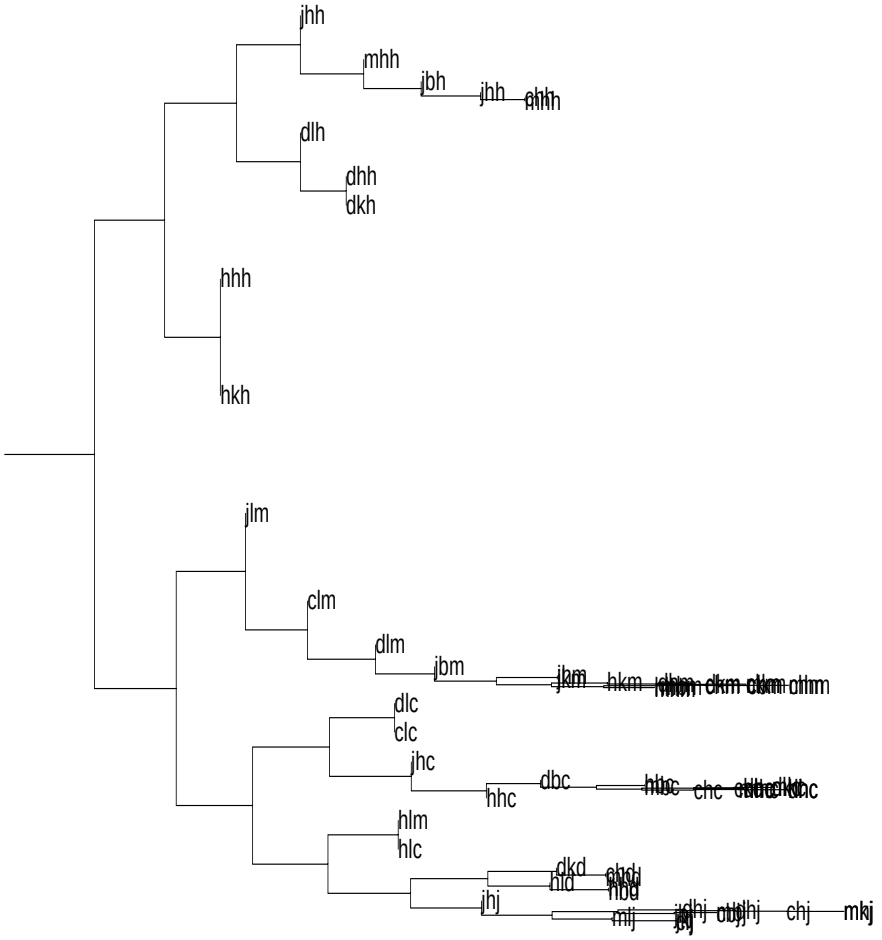

Figure A.8: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 2 unit representation.

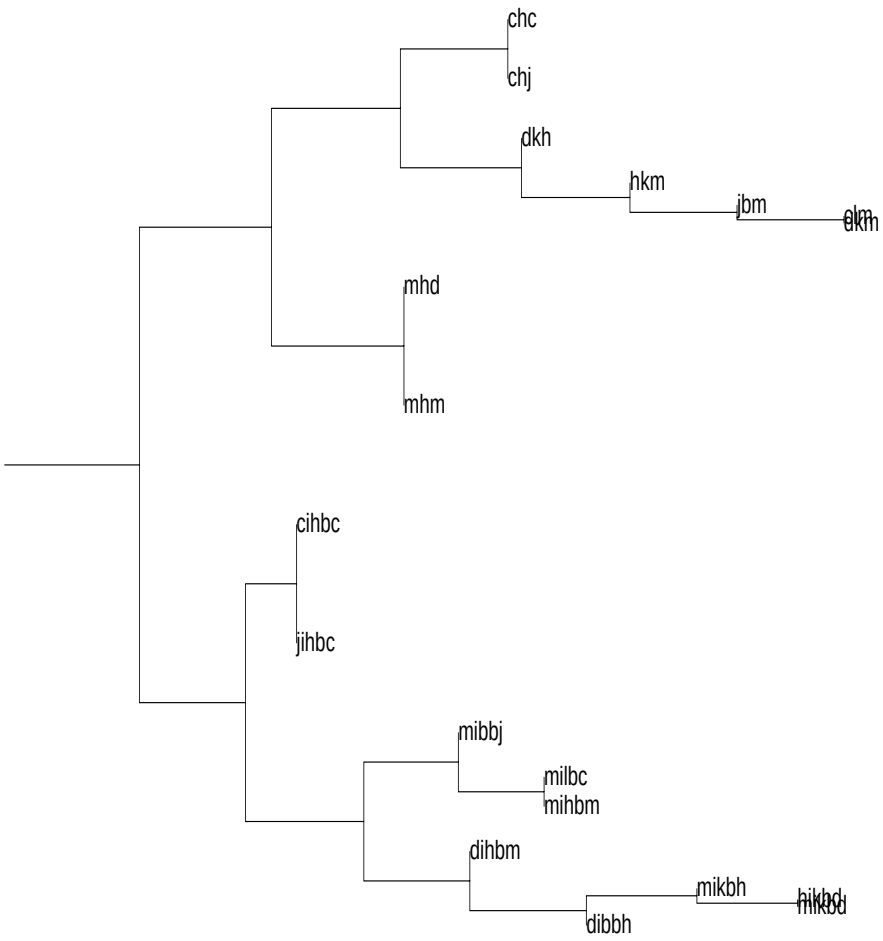

Figure A.9: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation.

Figure A.10: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation.

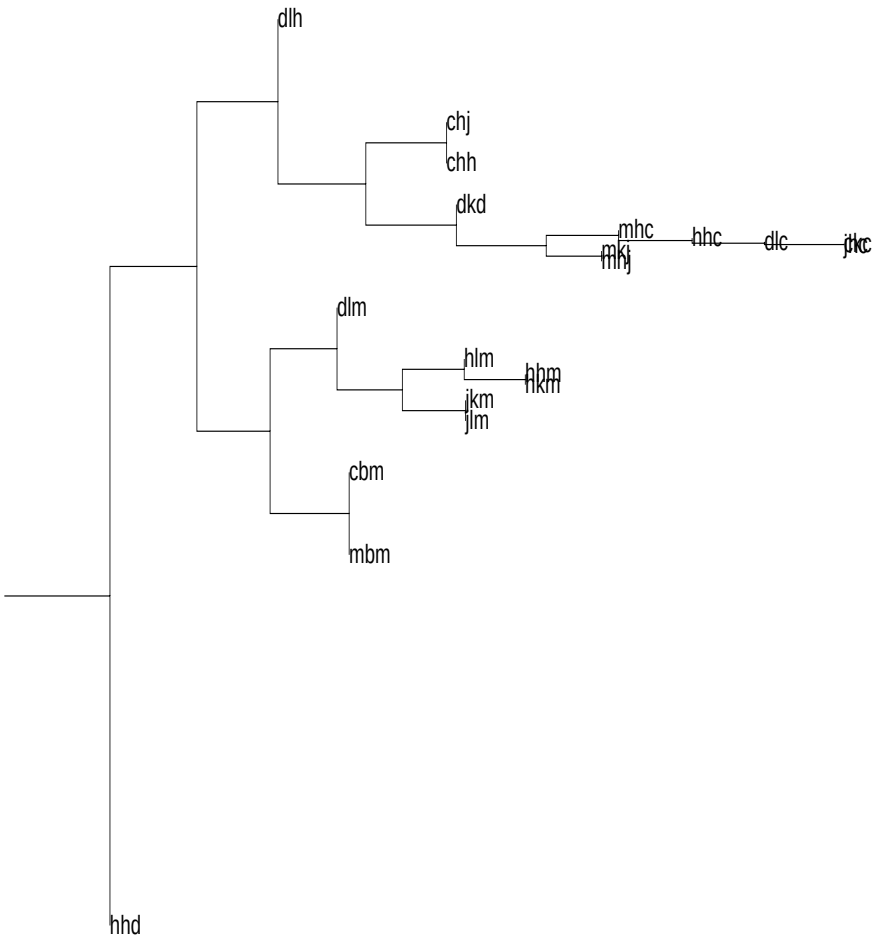

Figure A.11: Top branch of the HCA plot for the SRAAM trained on 40 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation.

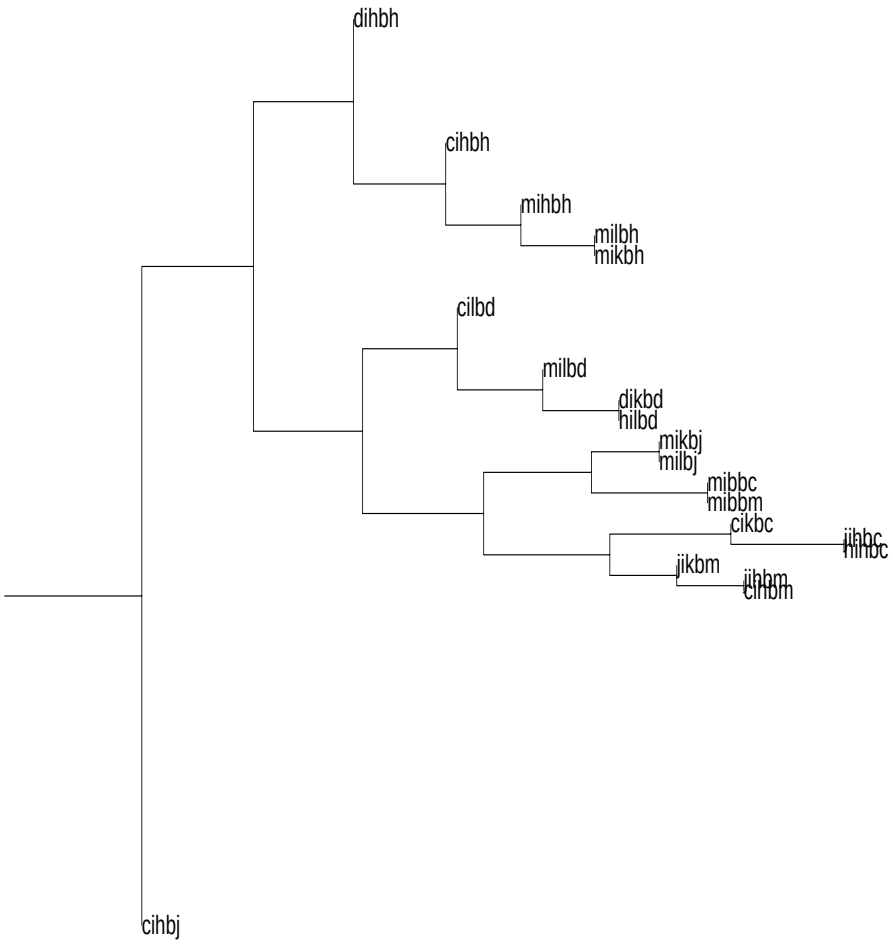

Figure A.12: Bottom branch of the HCA plot for the SRAAM trained on 40 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation.

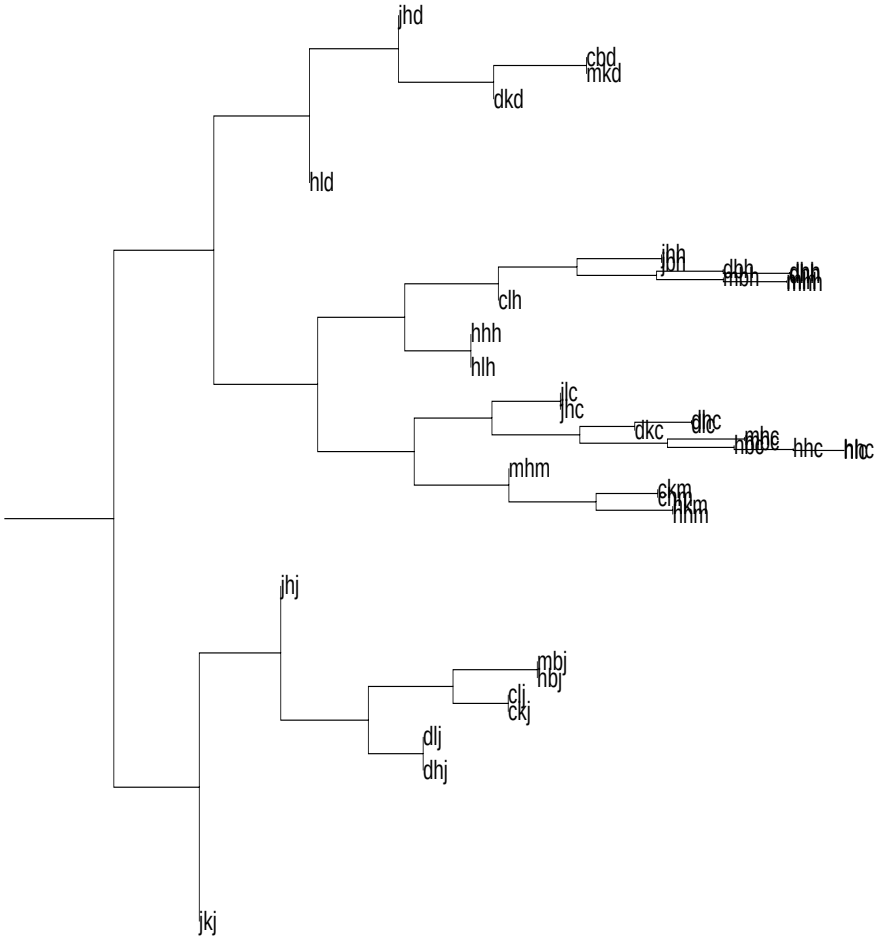

Figure A.13: Top branch of the HCA plot for the SRAAM trained on 80 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation.

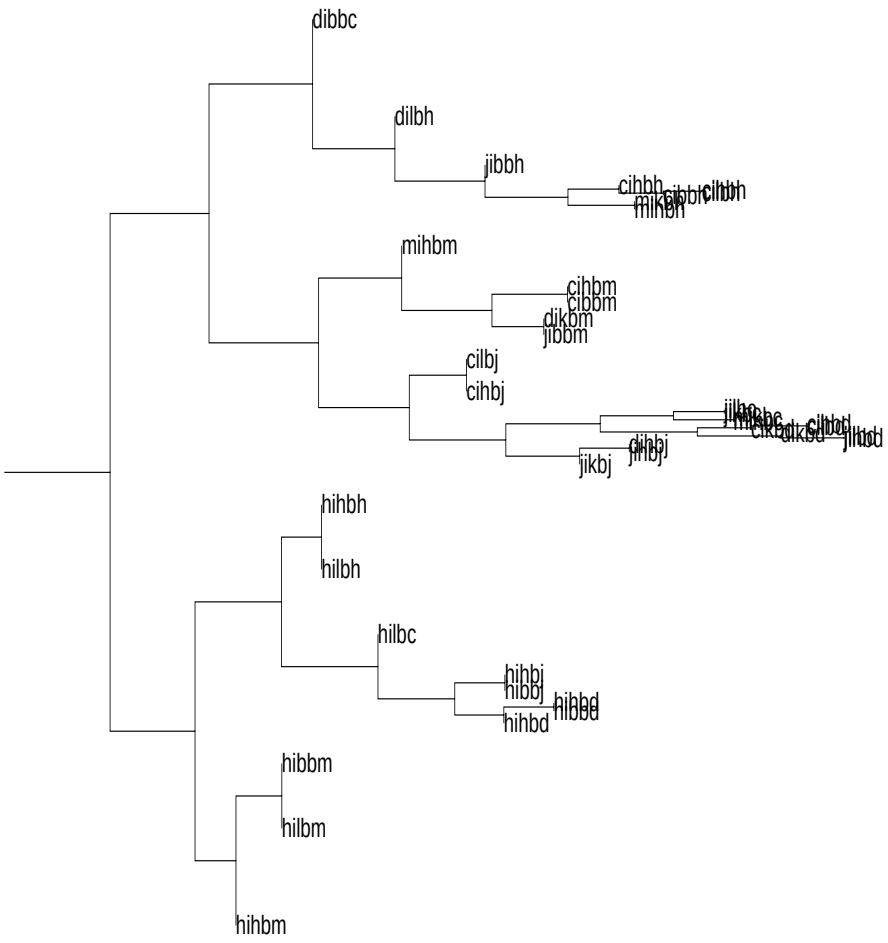

Figure A.14: Bottom branch of the HCA plot for the SRAAM trained on 80 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation.

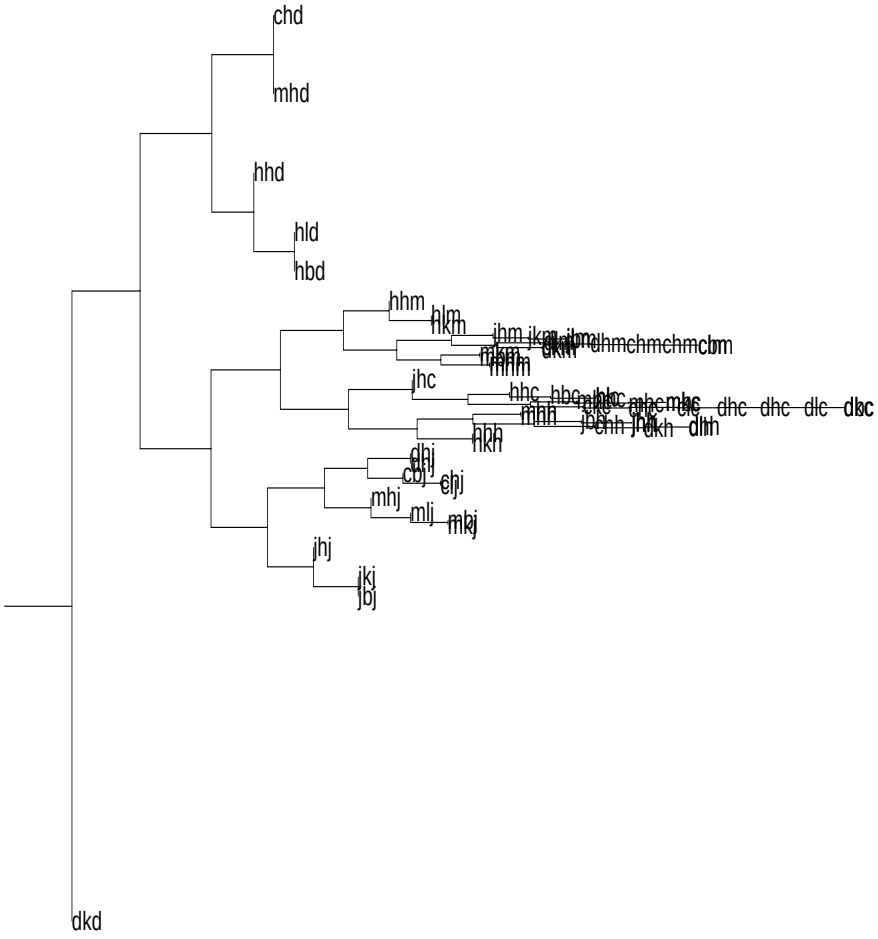

Figure A.15: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation.

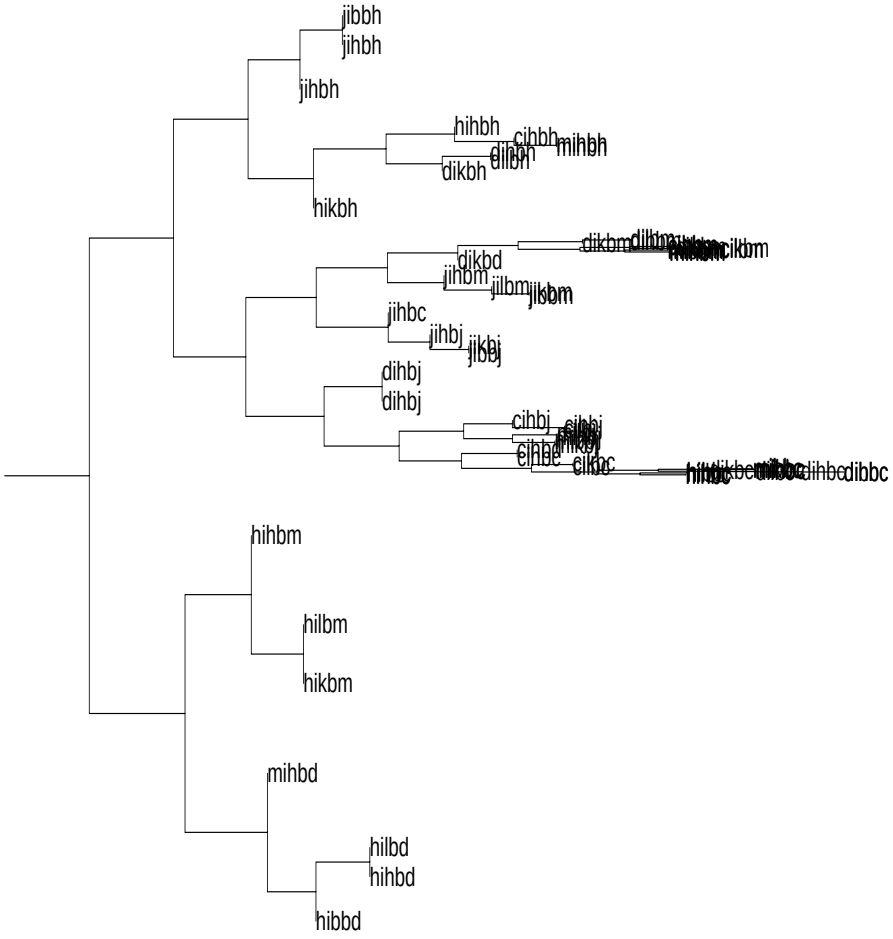

Figure A.16: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation.

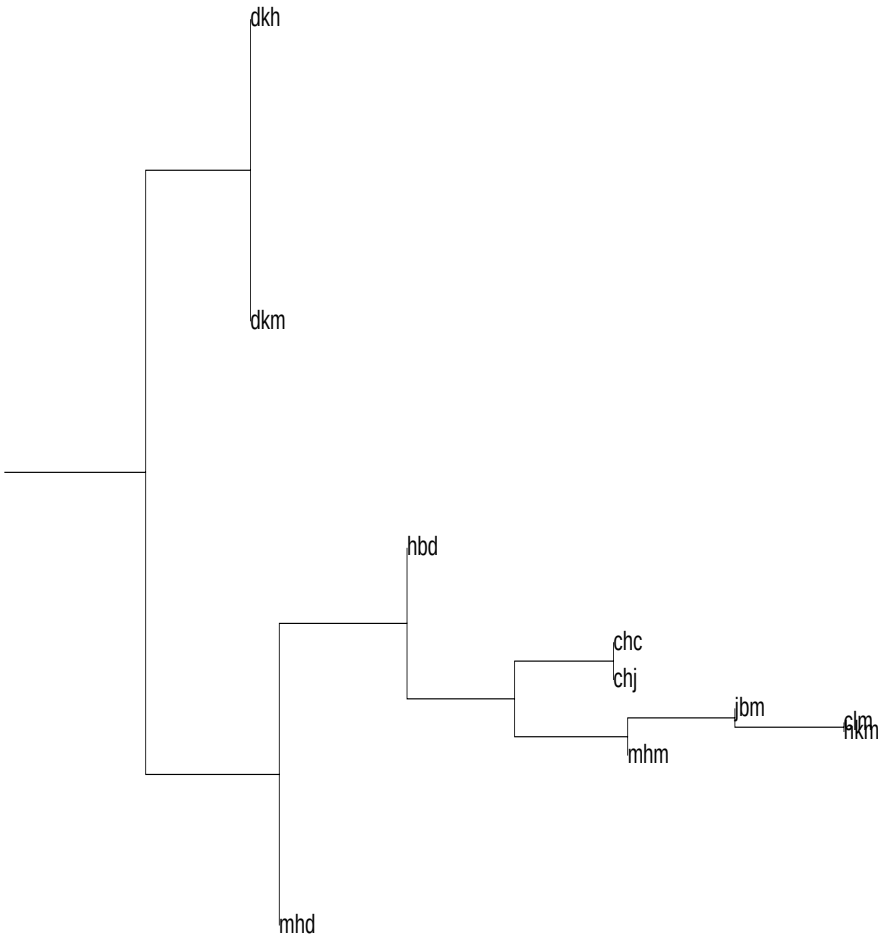

Figure A.17: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation.

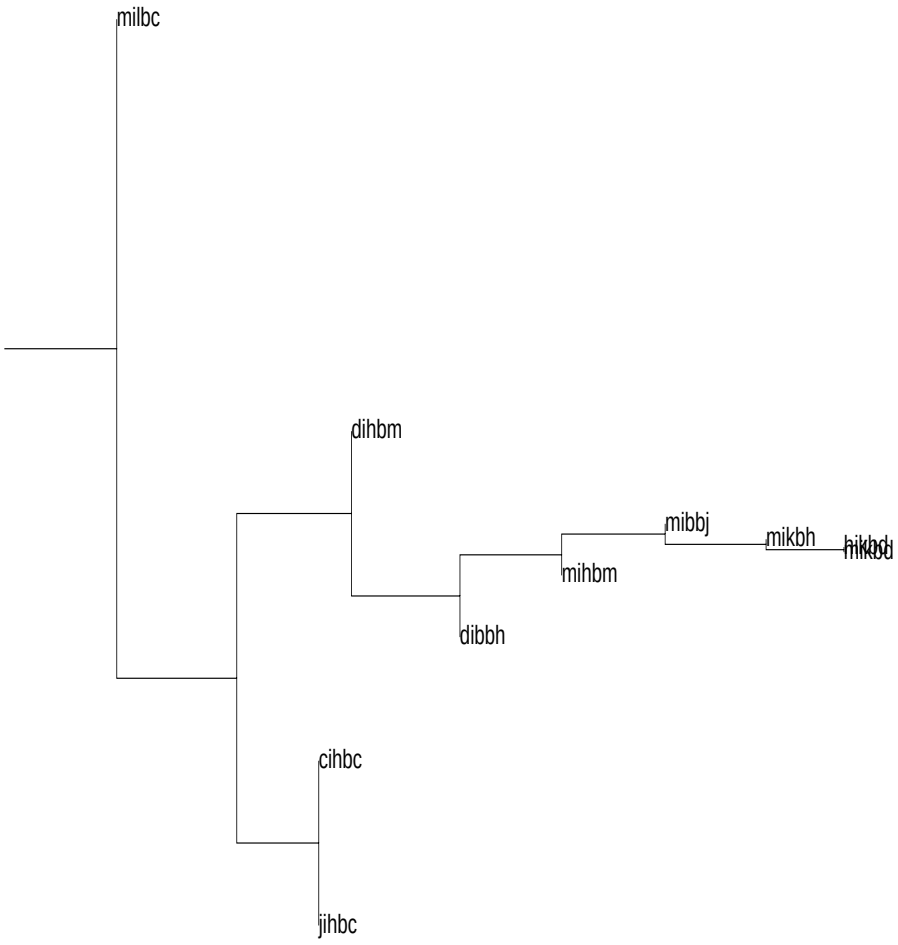

Figure A.18: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation.

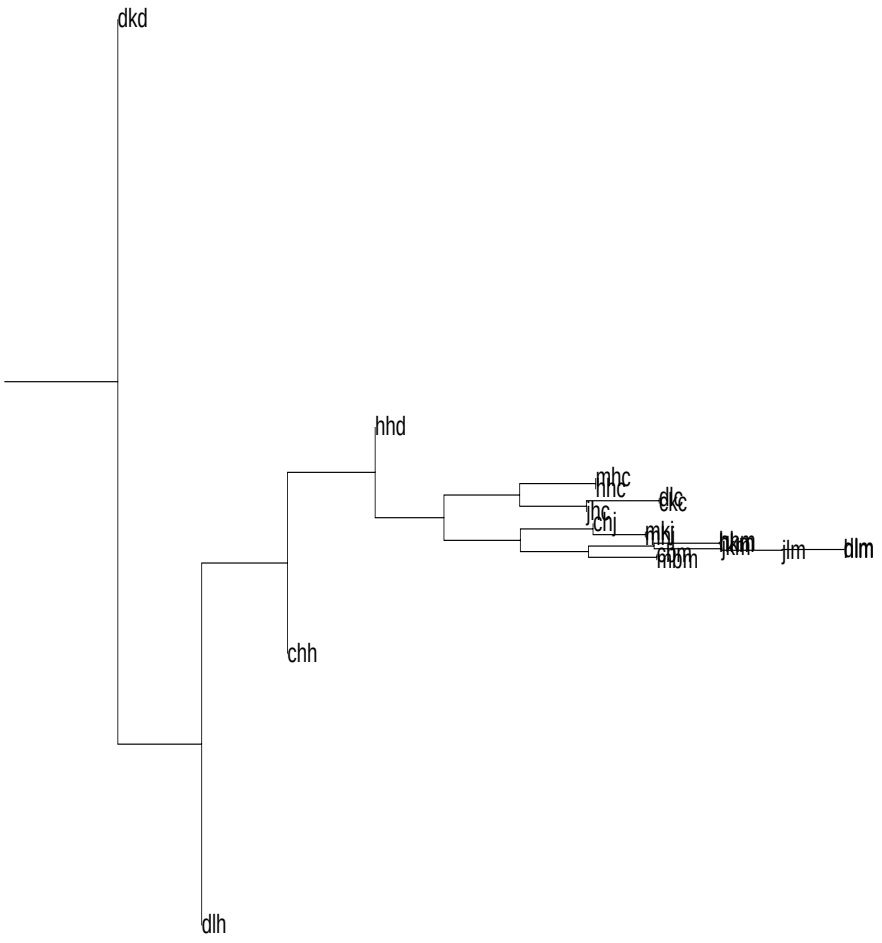

Figure A.19: Top branch of the HCA plot for the SRAAM trained on 40 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation.

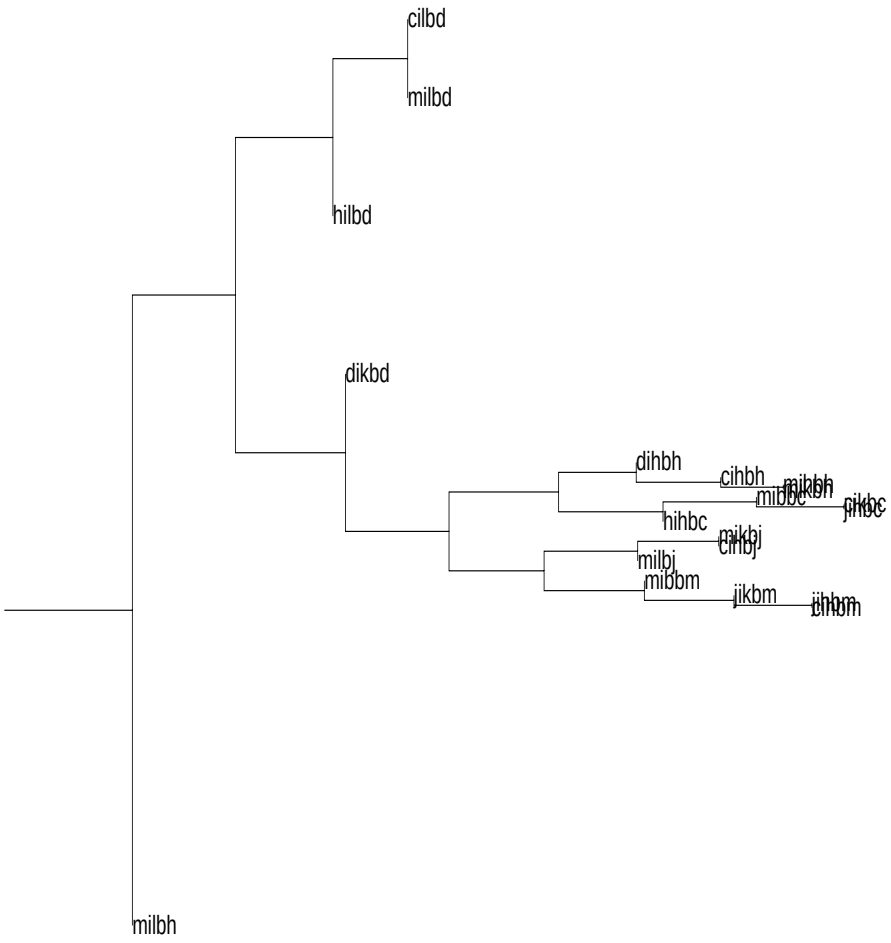

Figure A.20: Bottom branch of the HCA plot for the SRAAM trained on 40 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation.

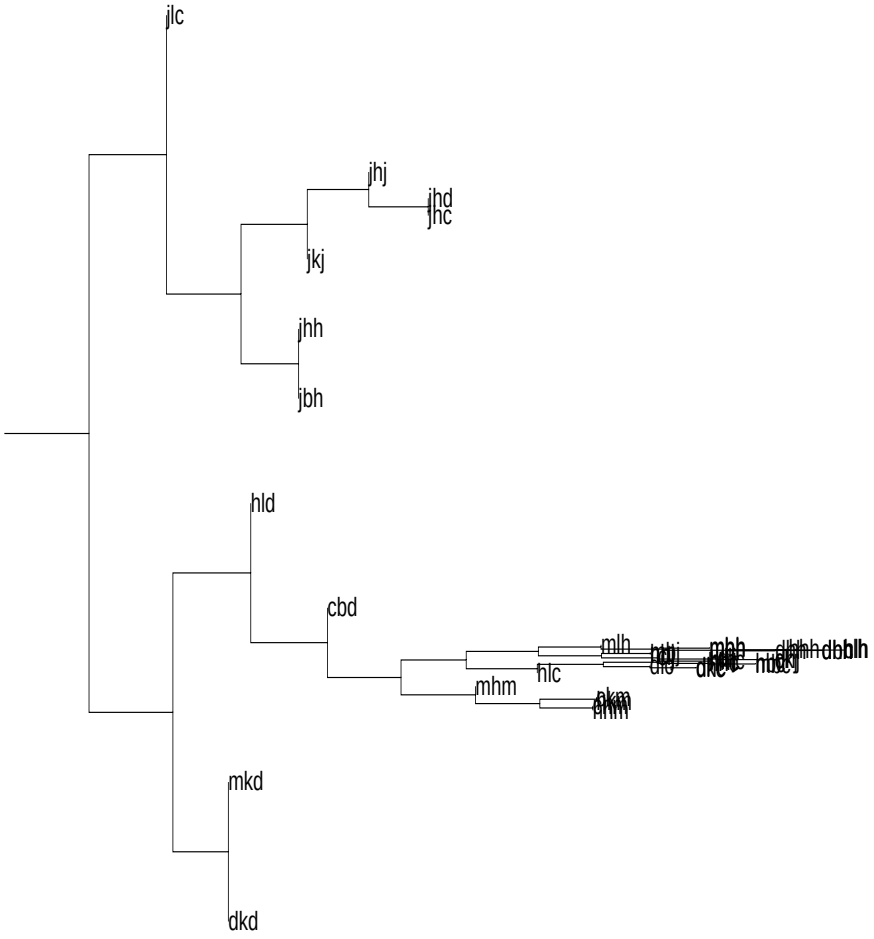

Figure A.21: Top branch of the HCA plot for the SRAAM trained on 80 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation.

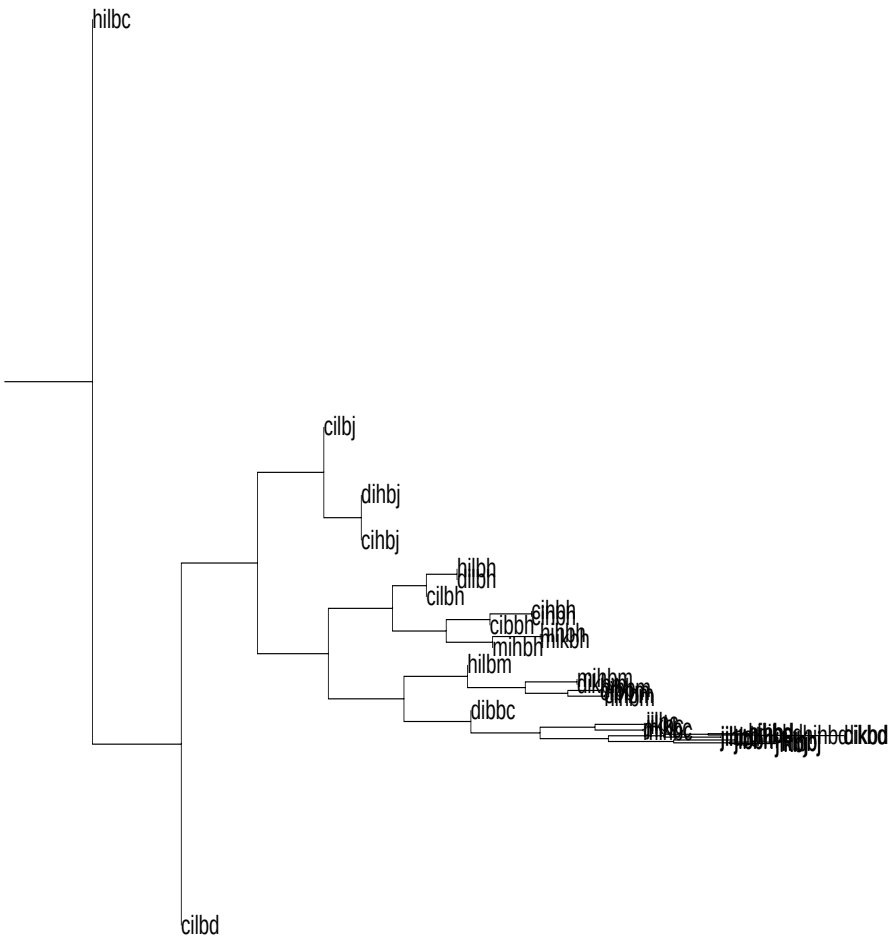

Figure A.22: Bottom branch of the HCA plot for the SRAAM trained on 80 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation.

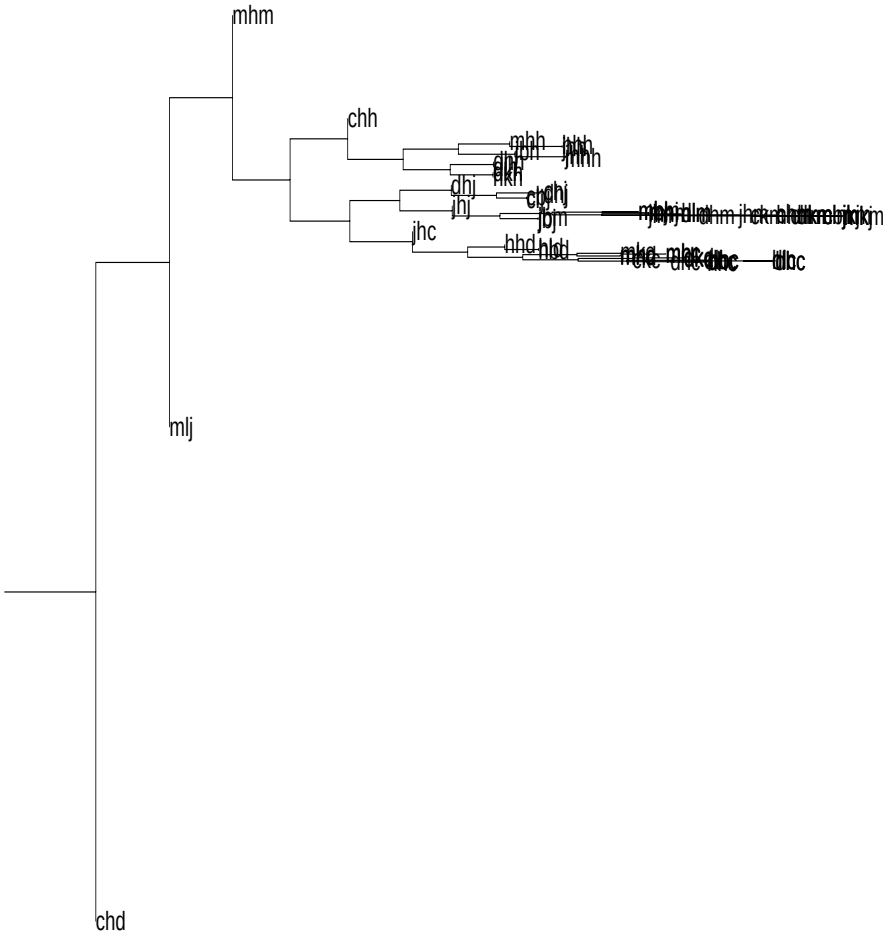

Figure A.23: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation.

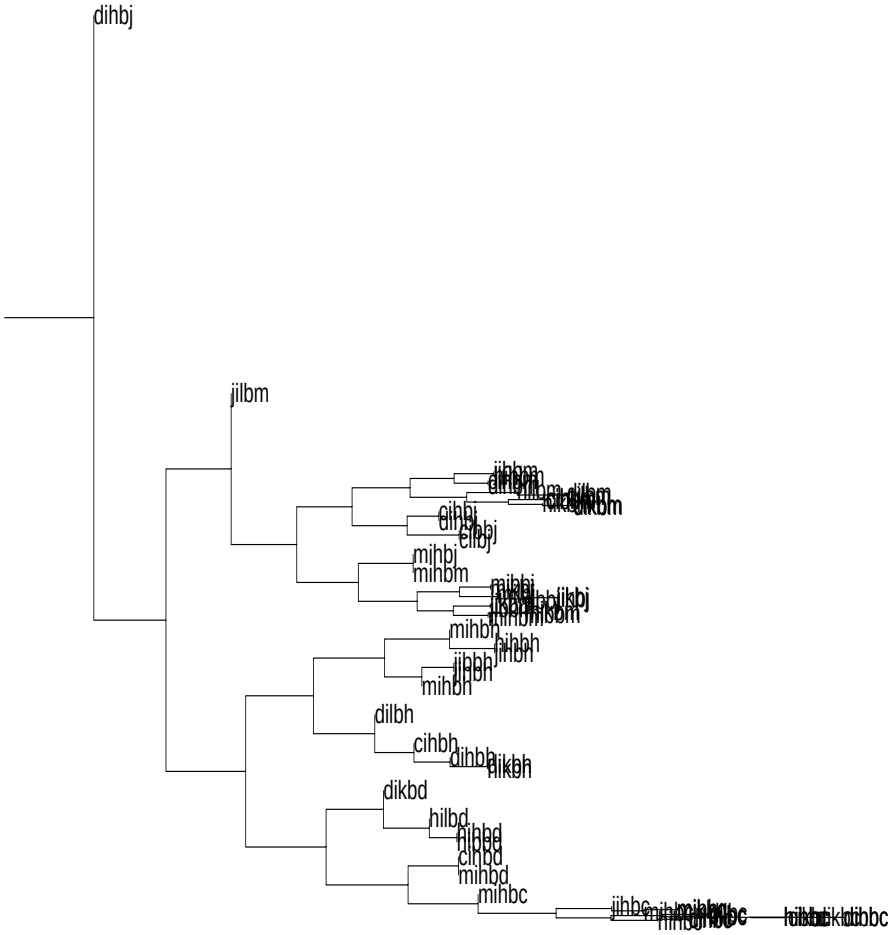

Figure A.24: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation.

A.1.2 HCA plots for partially trained networks

Figures A.25 to A.48 show the dendrograms for networks trained with back-propagation at approximately 1/3 and 2/3 of the way through training.

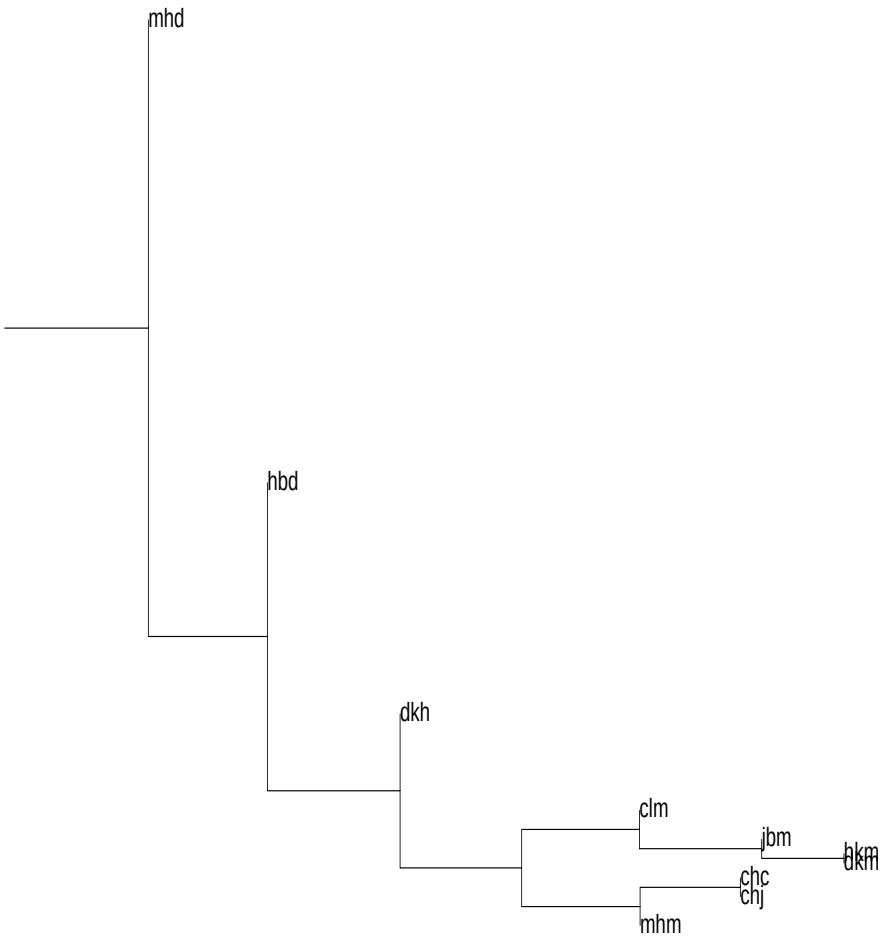

Figure A.25: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 2 unit representation after 1000 iterations.

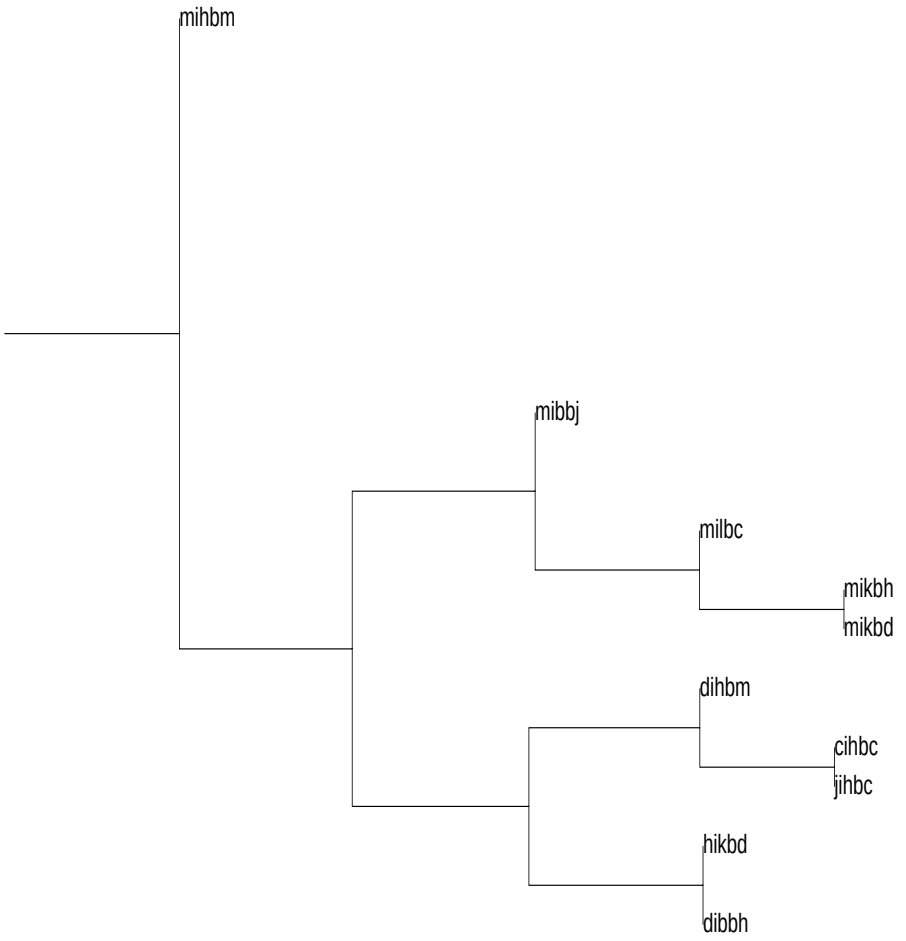

Figure A.26: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 2 unit representation after 1000 iterations.

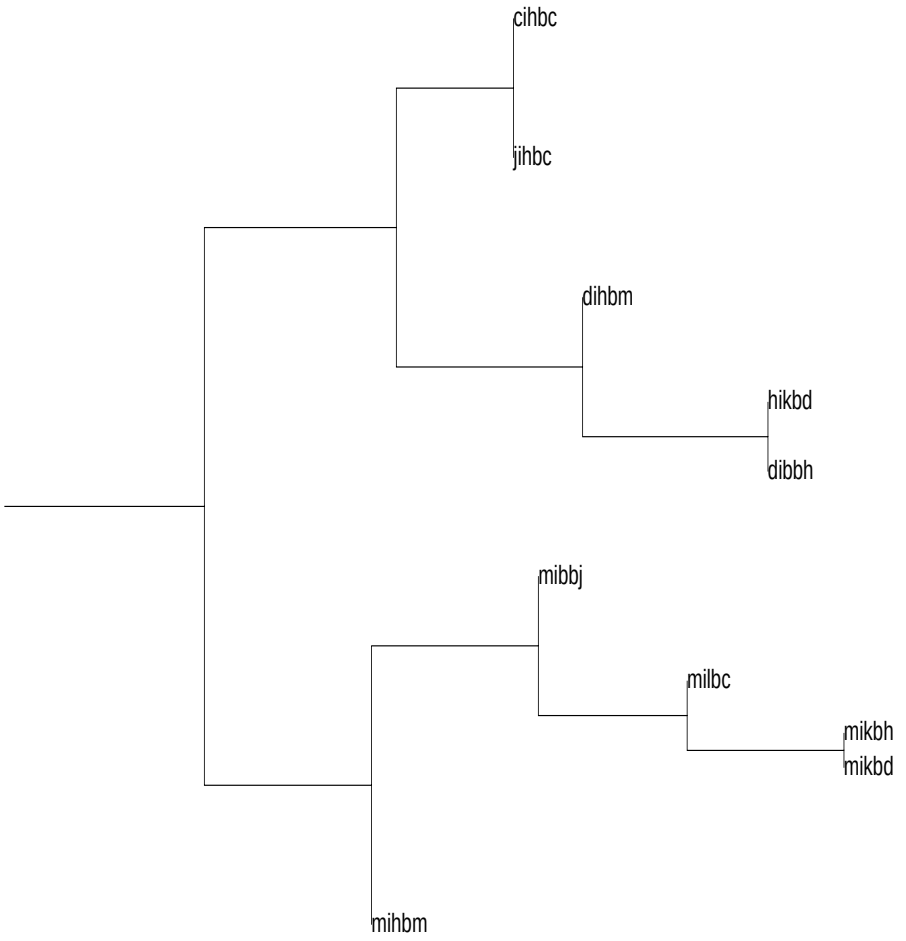

Figure A.27: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 2 unit representation after 2000 iterations.

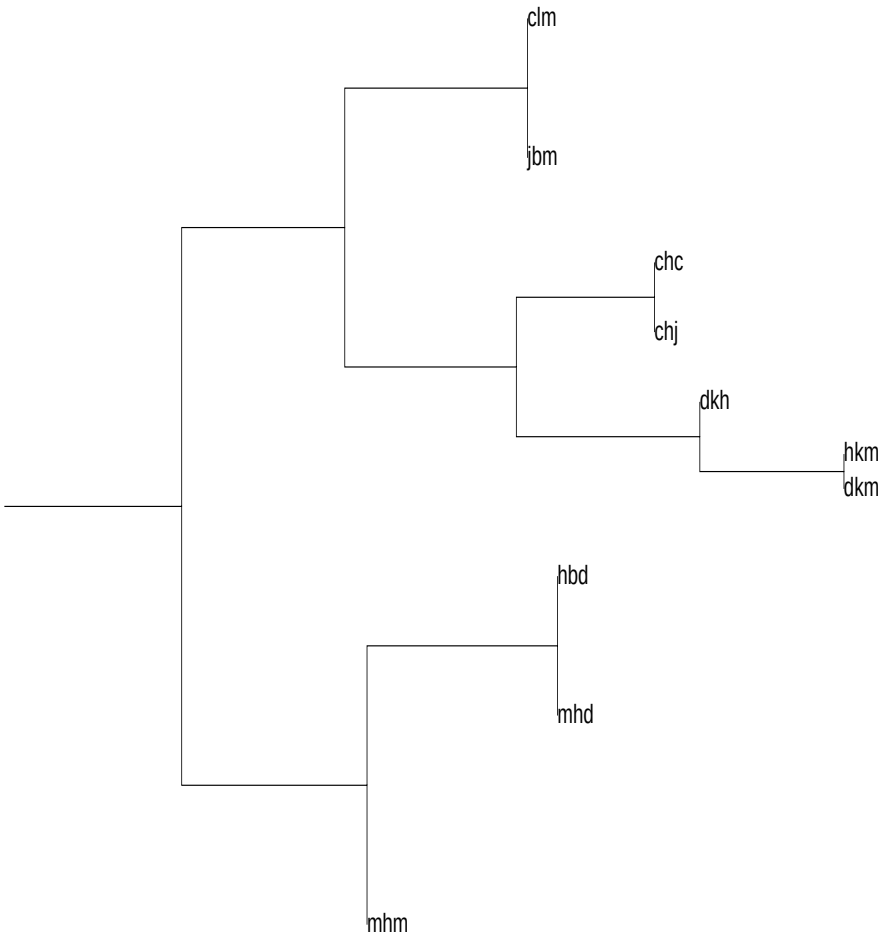

Figure A.28: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, sigmoidal units and the 2 unit representation after 2000 iterations.

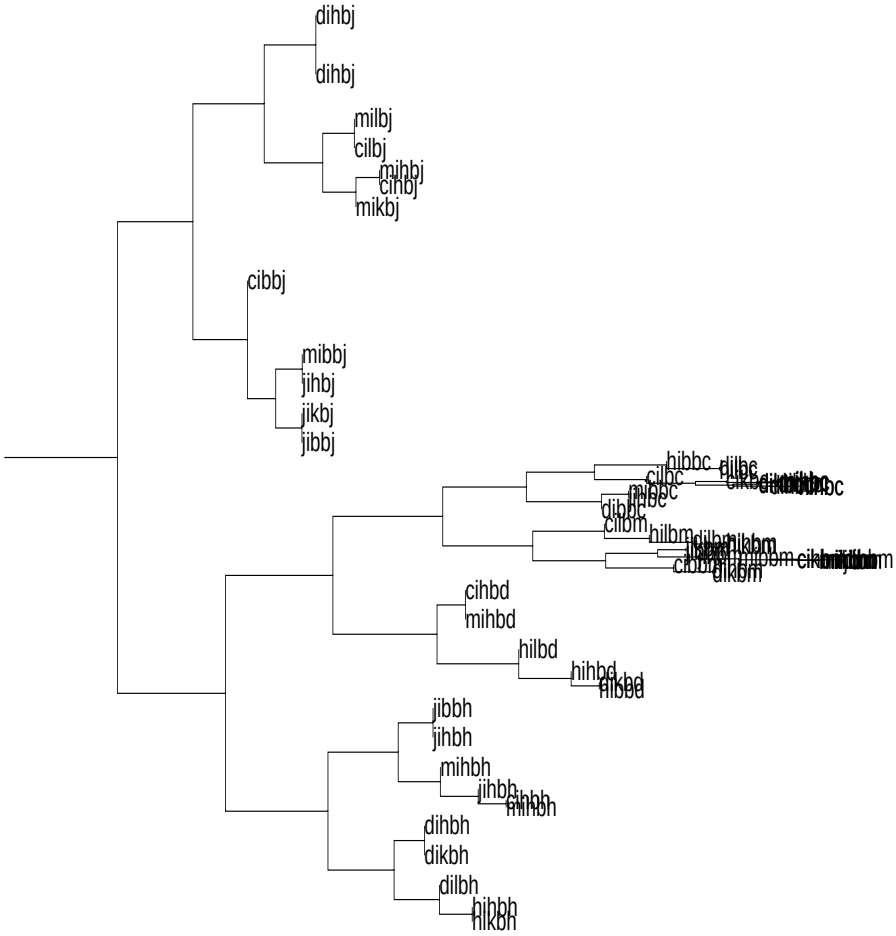

Figure A.29: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 2 unit representation after 500 iterations.

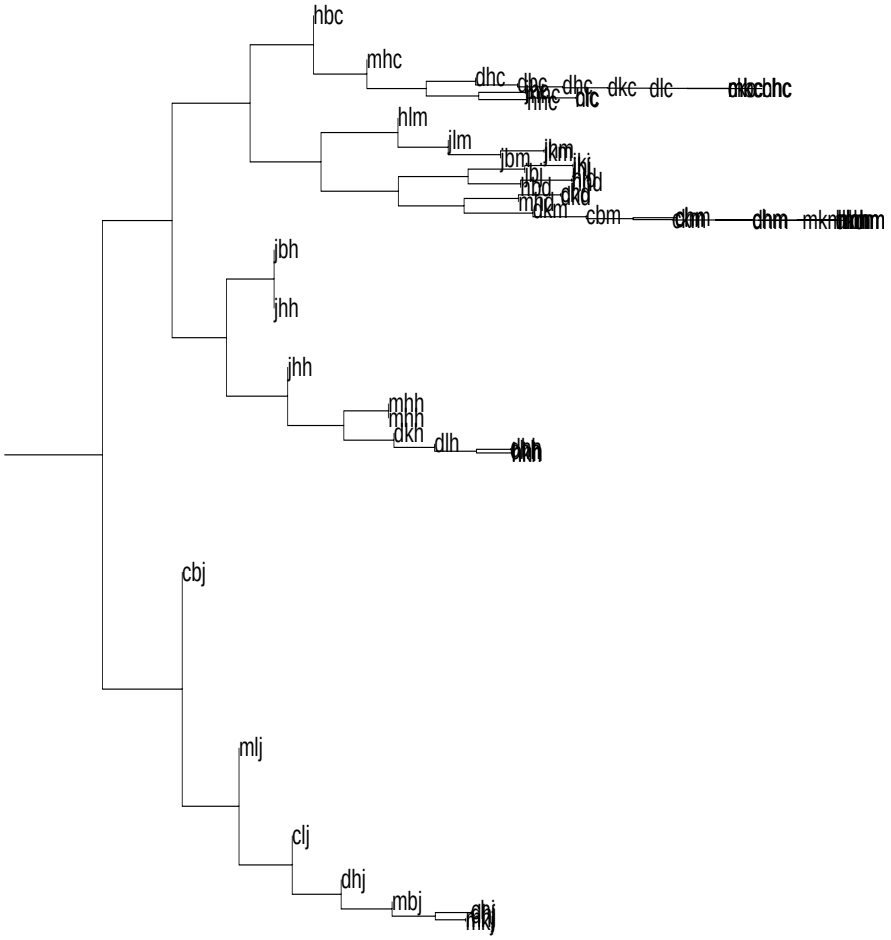

Figure A.30: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 2 unit representation after 500 iterations.

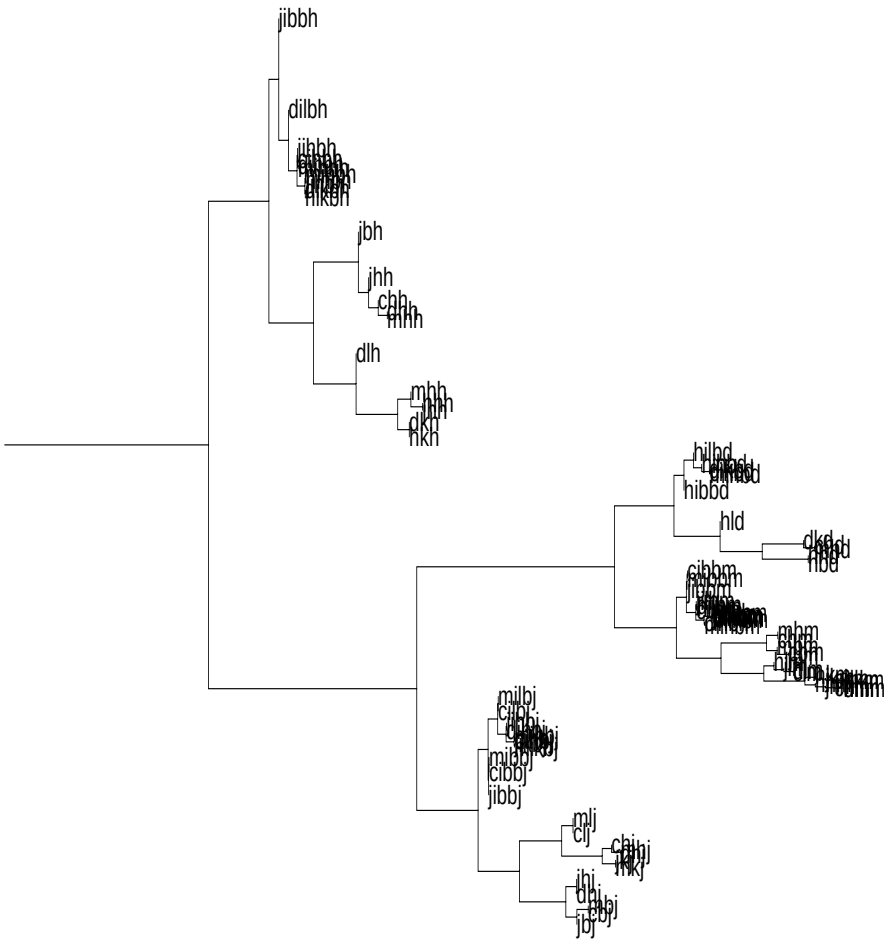

Figure A.31: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 2 unit representation after 1000 iterations.

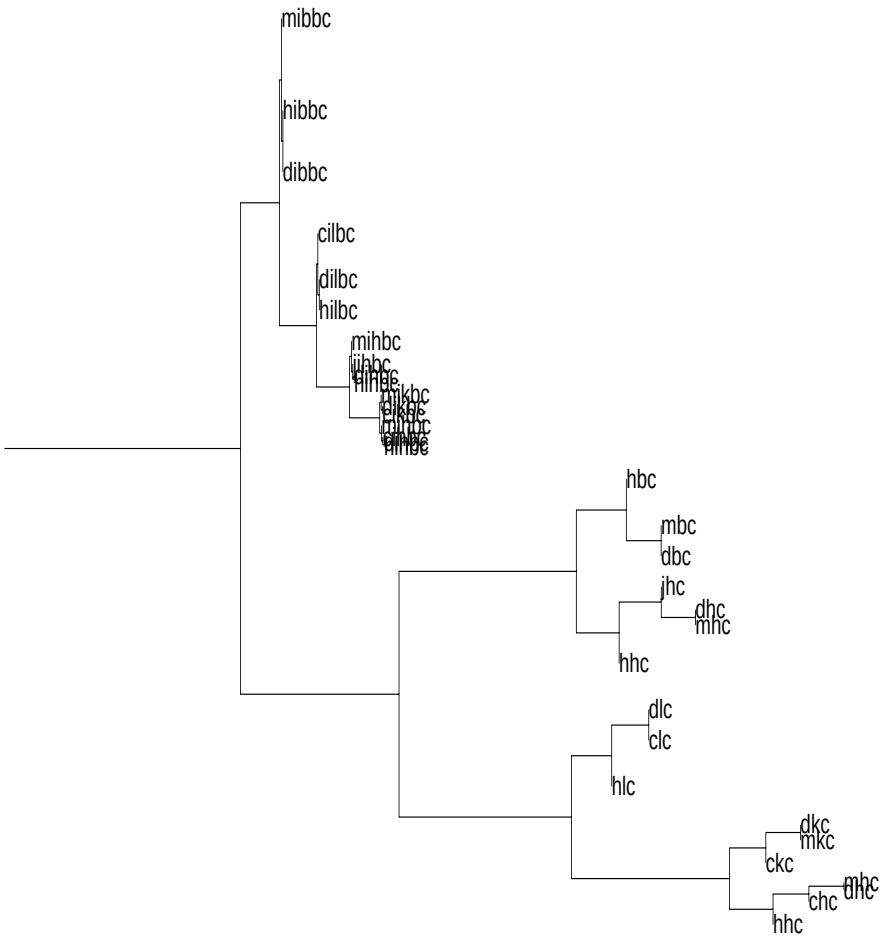

Figure A.32: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, sigmoidal units and the 2 unit representation after 1000 iterations.

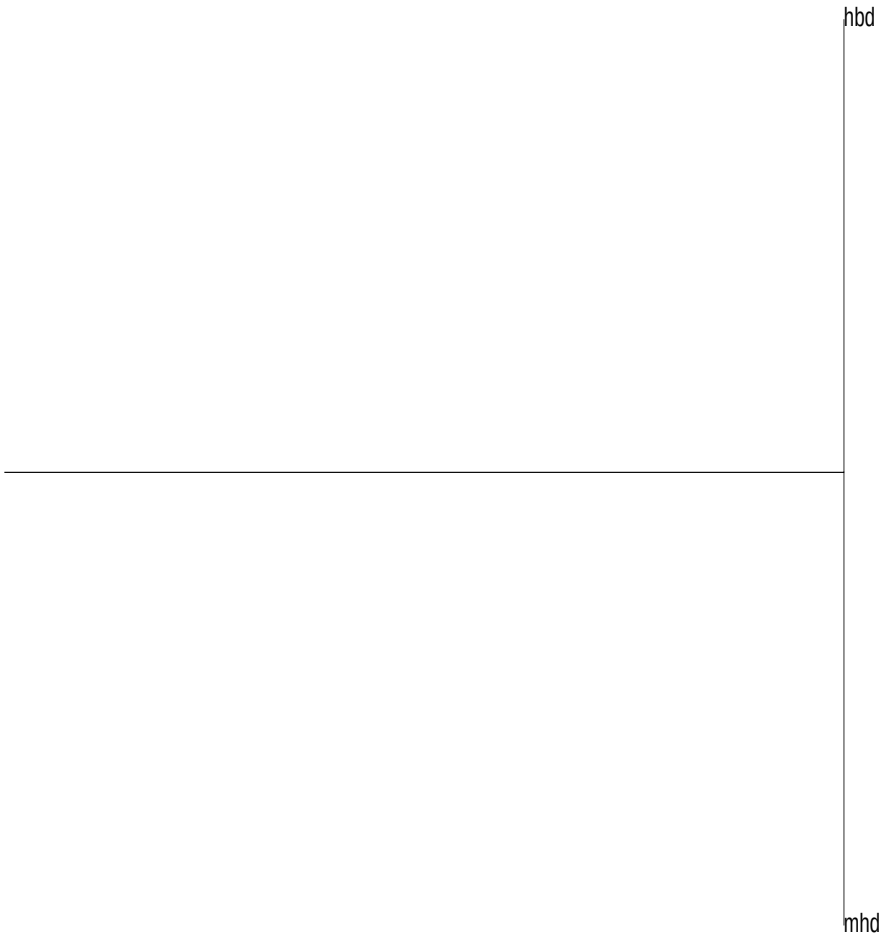

Figure A.33: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation after 2600 iterations.

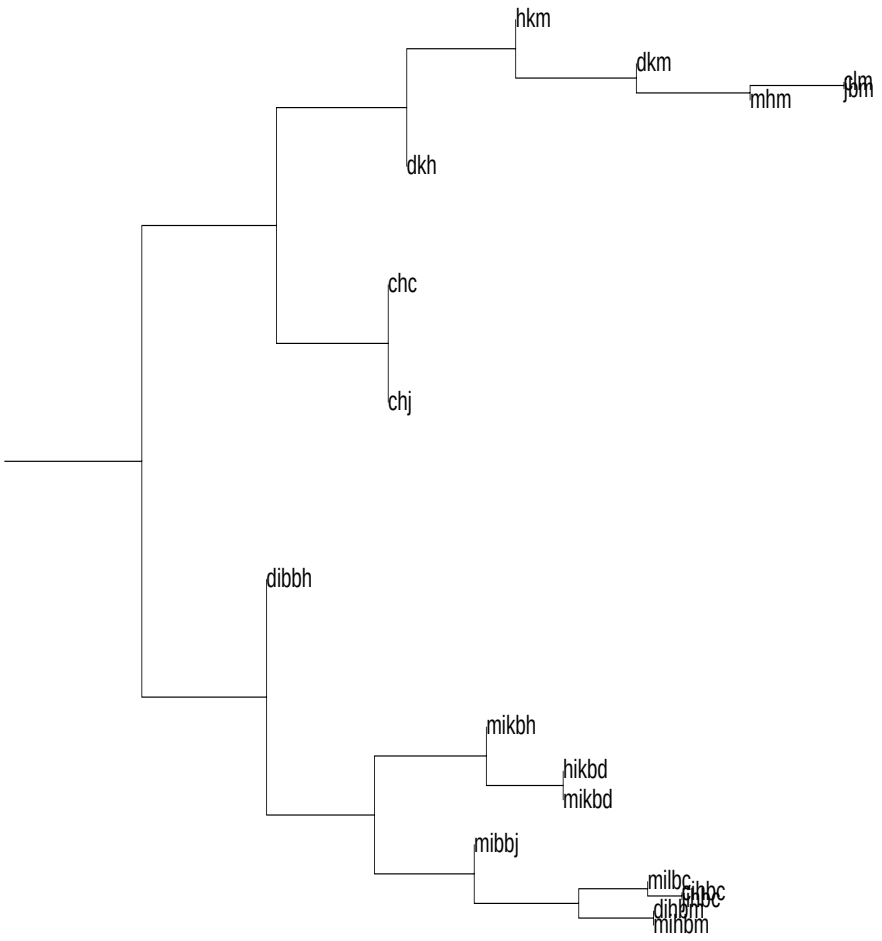

Figure A.34: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation after 2600 iterations.

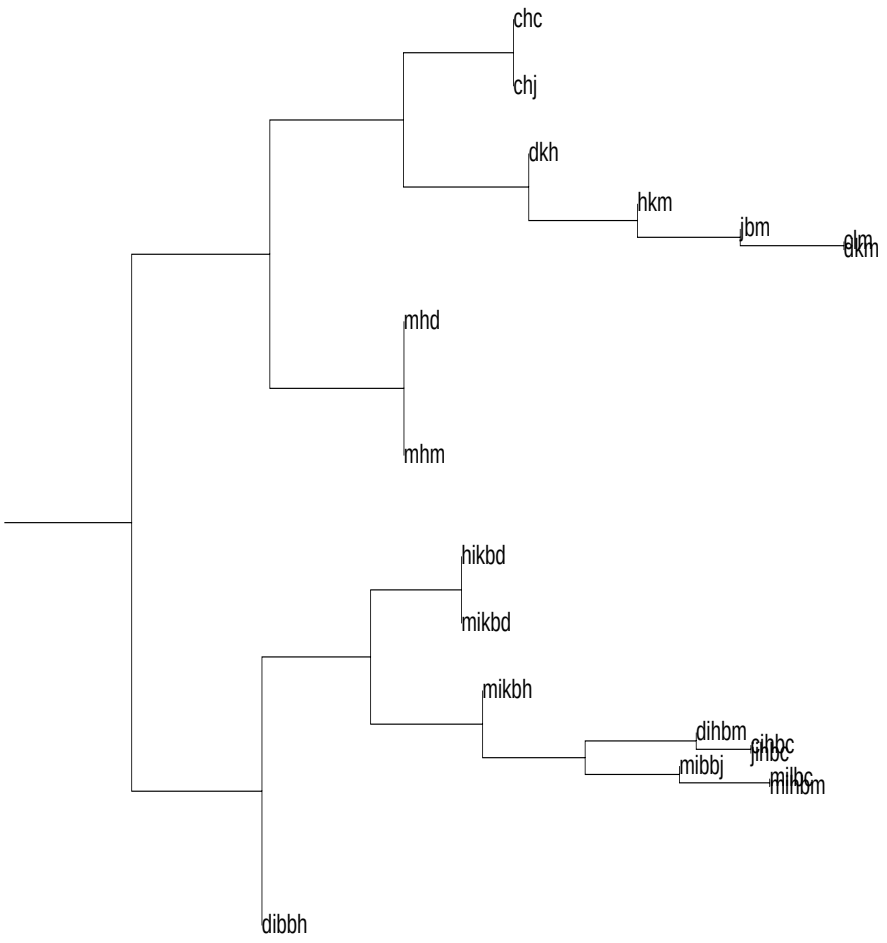

Figure A.35: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation after 5000 iterations.

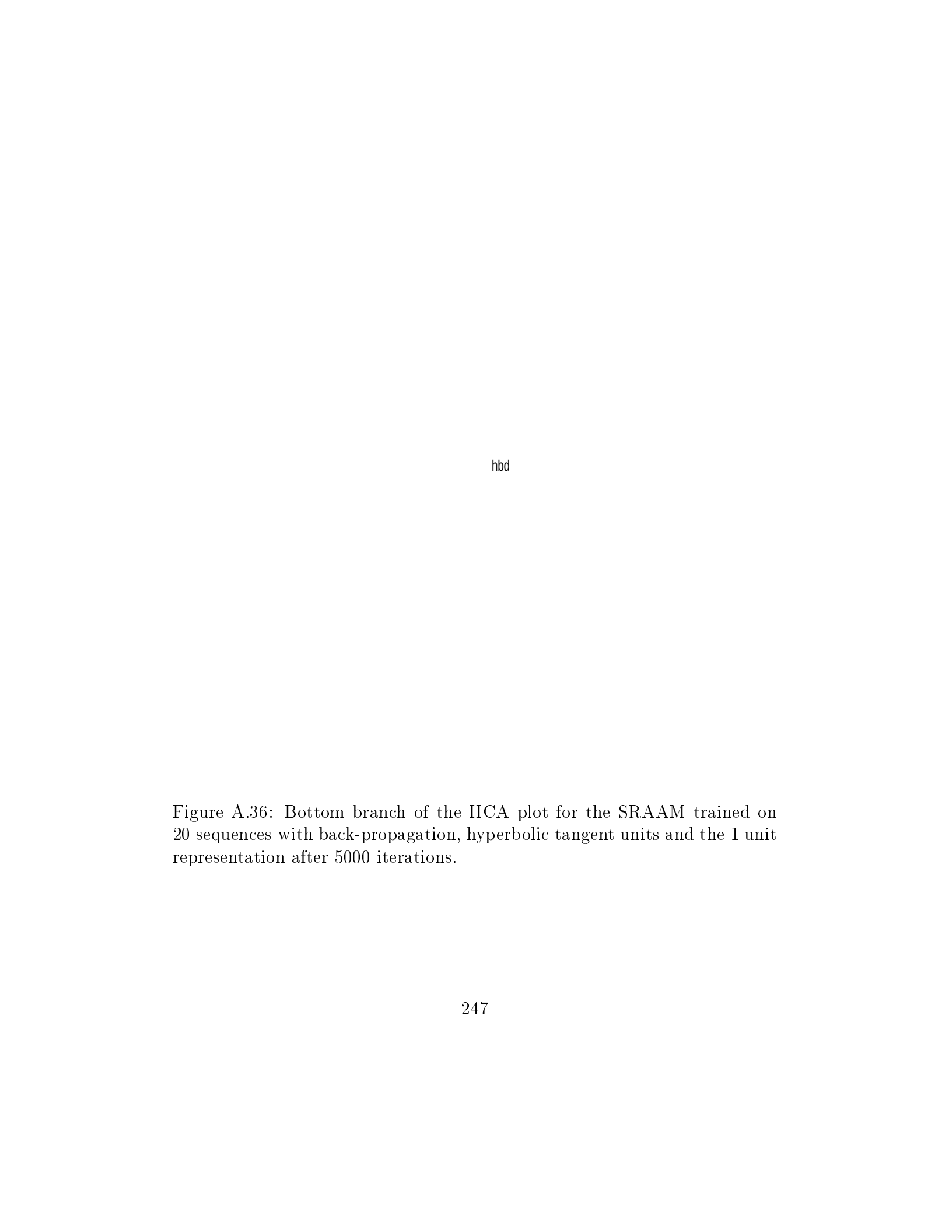

Figure A.36: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation after 5000 iterations.

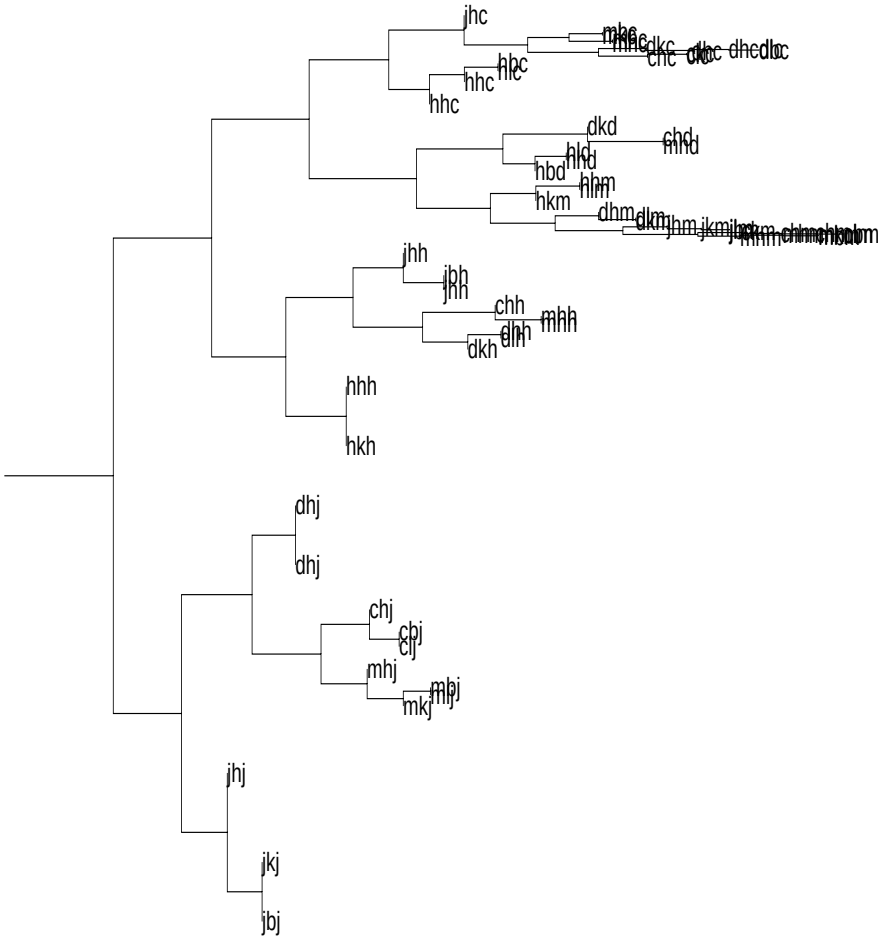

Figure A.37: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation after 100 iterations.

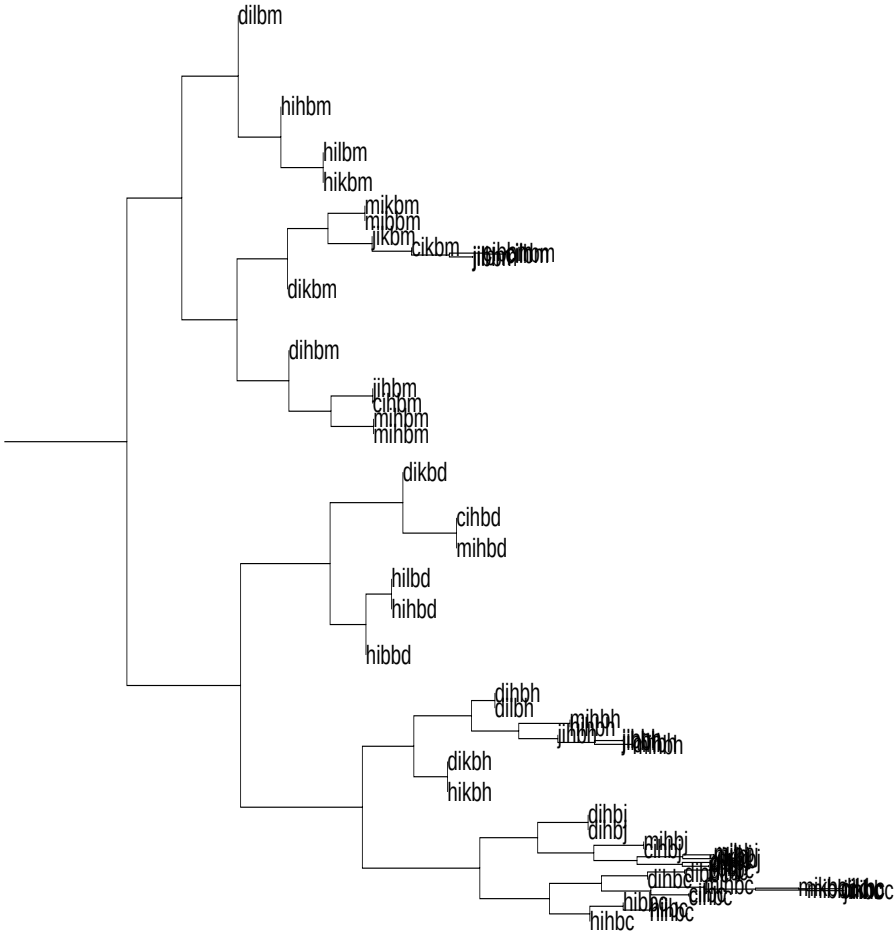

Figure A.38: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation after 100 iterations.

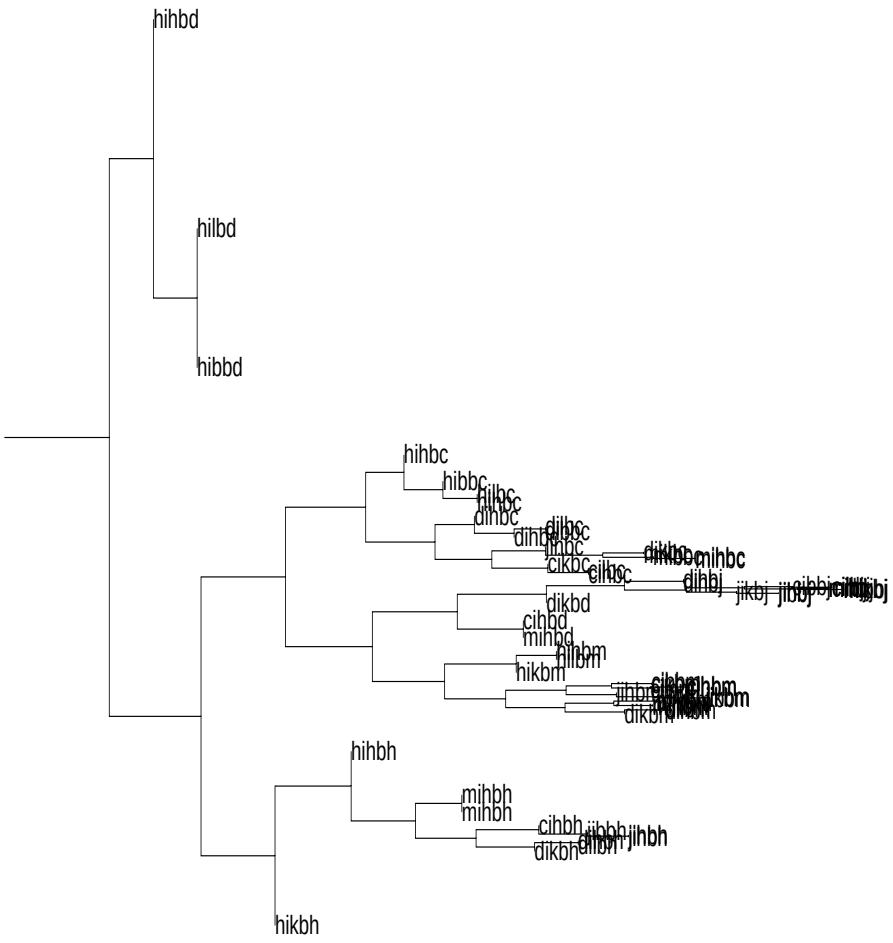

Figure A.39: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation after 200 iterations.

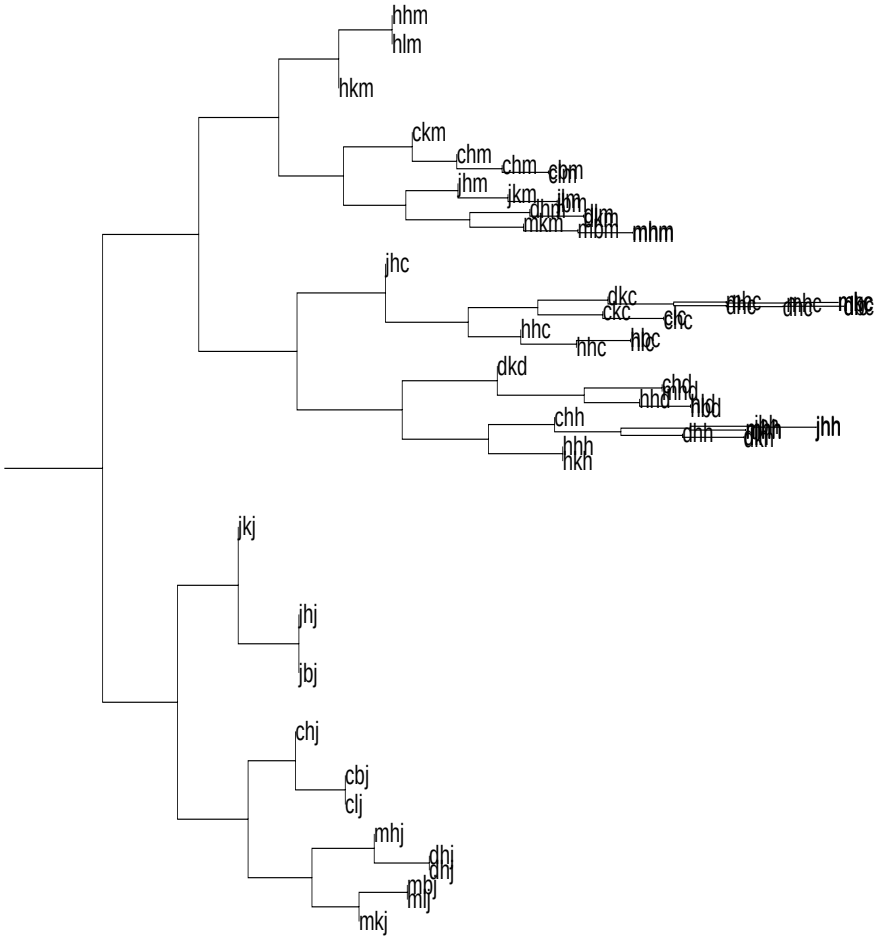

Figure A.40: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 1 unit representation after 200 iterations.

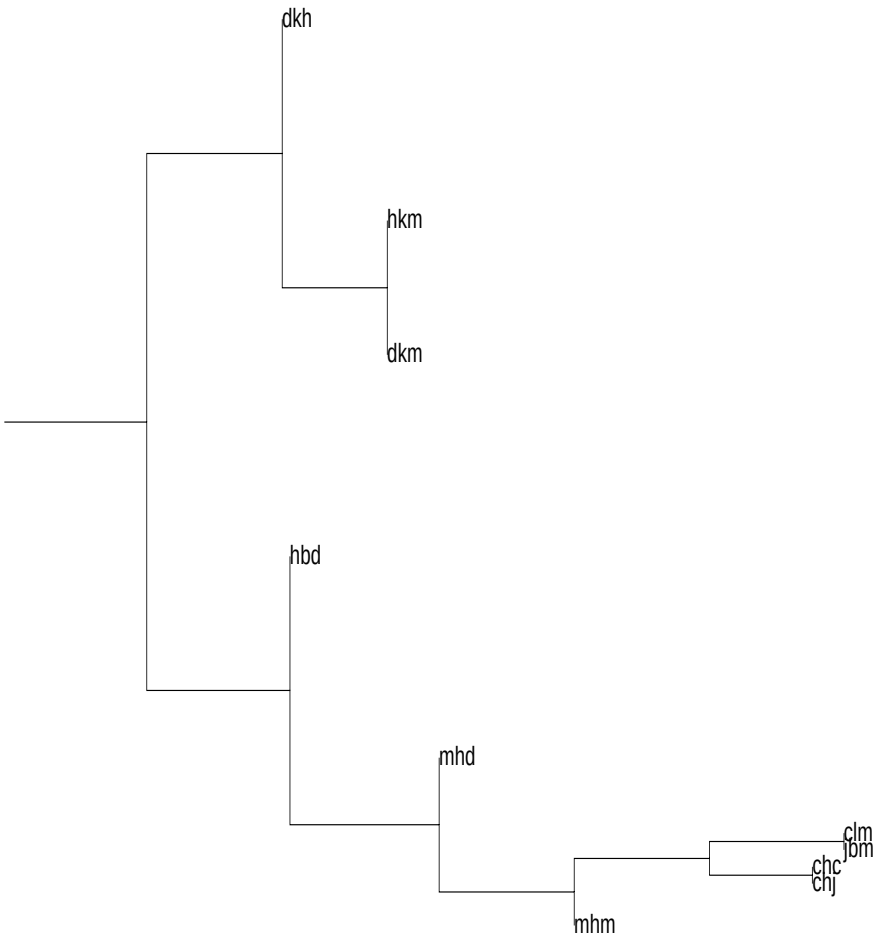

Figure A.41: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation after 1500 iterations.

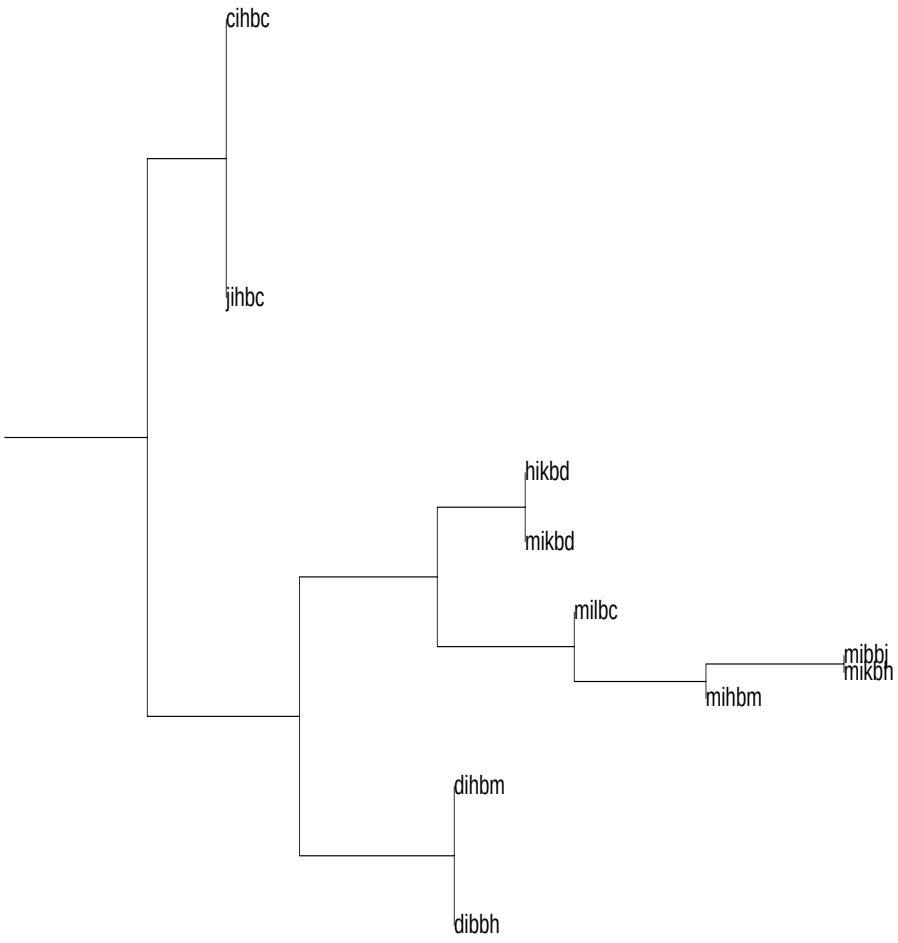

Figure A.42: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation after 1500 iterations.

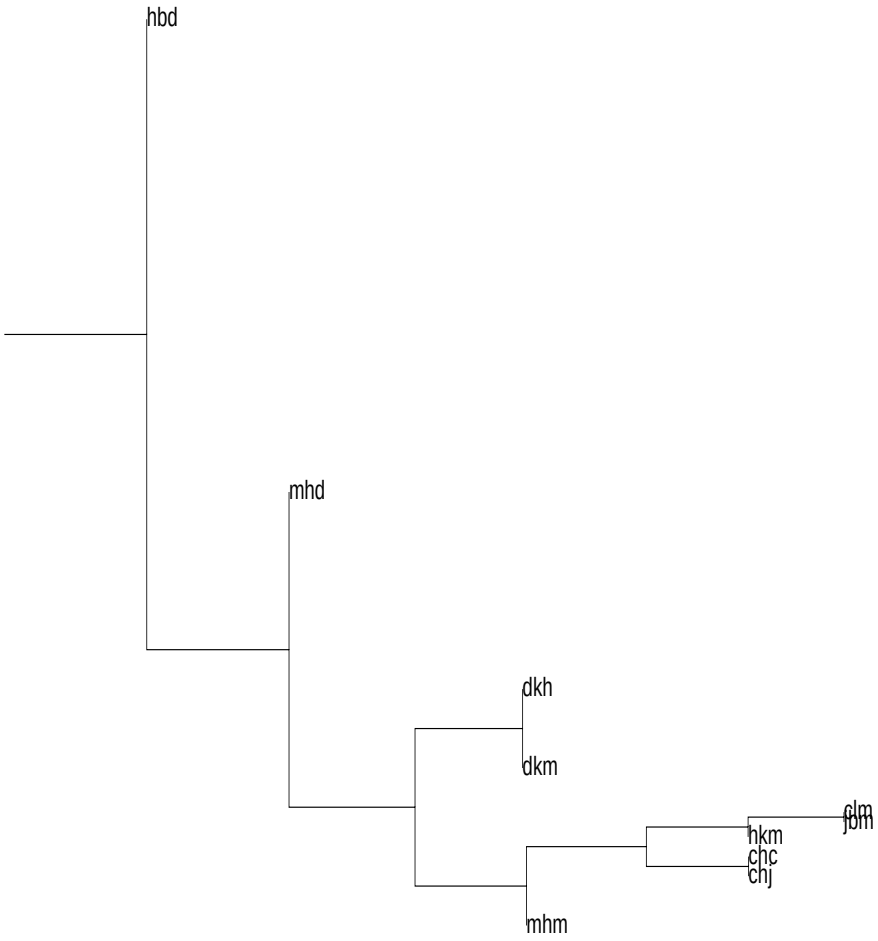

Figure A.43: Top branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation after 3000 iterations.

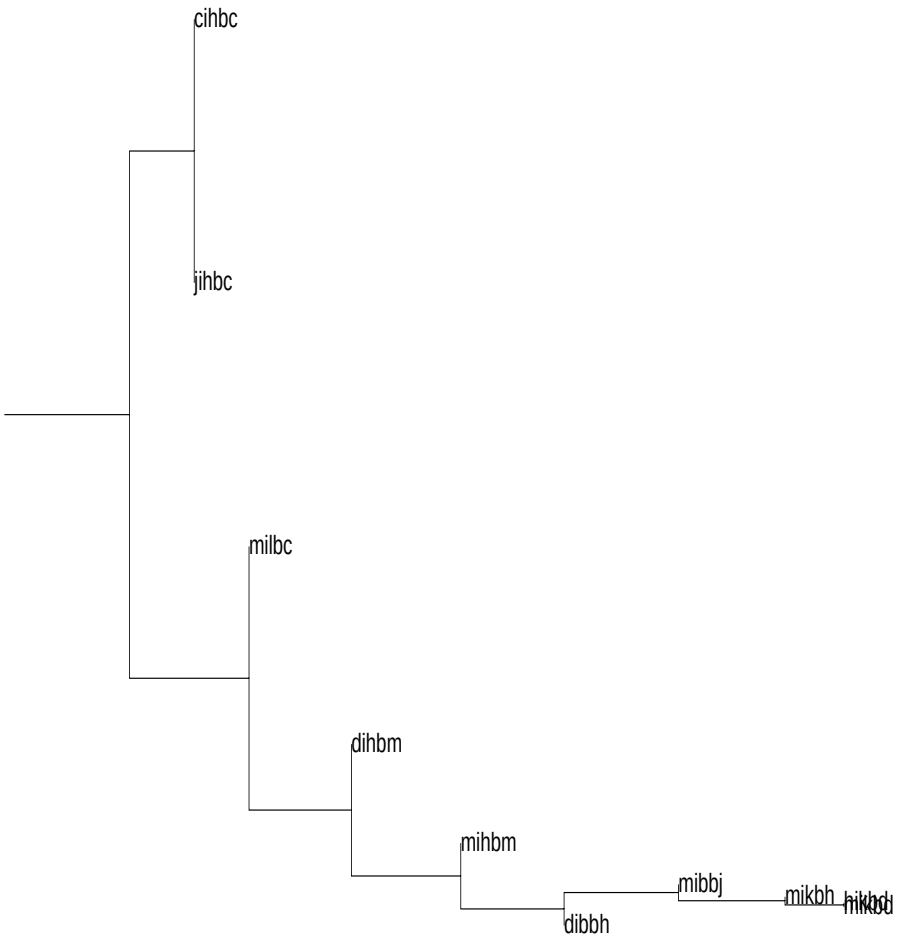

Figure A.44: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation after 3000 iterations.

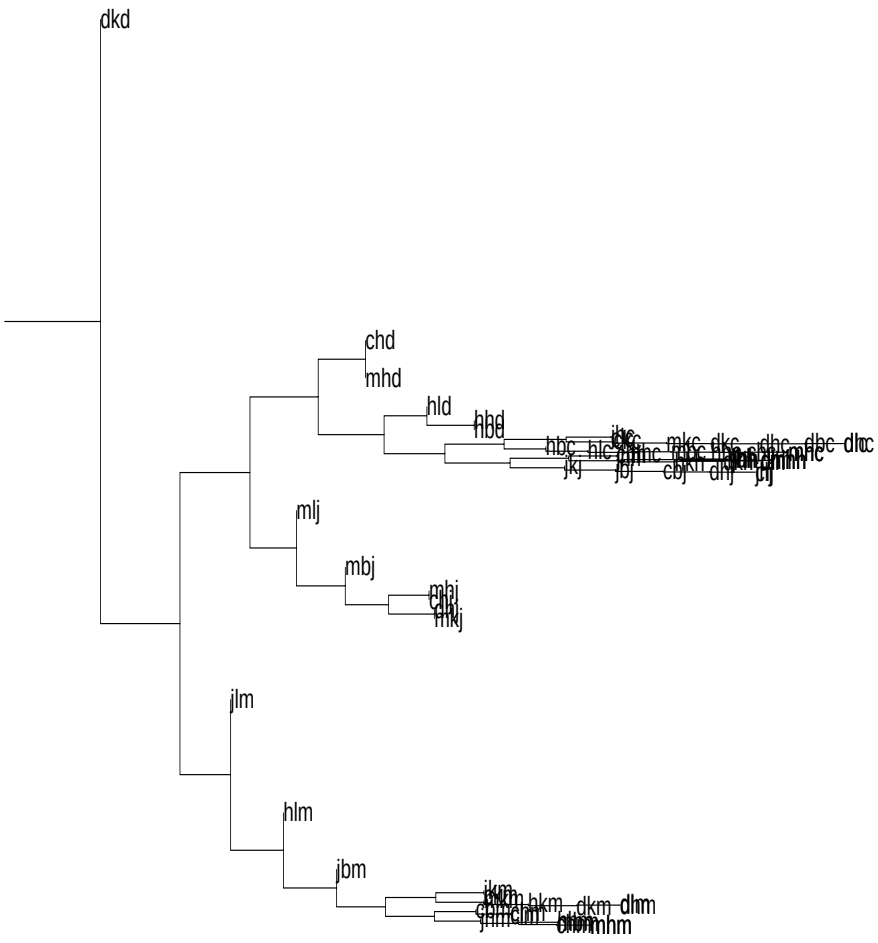

Figure A.45: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation after 100 iterations.

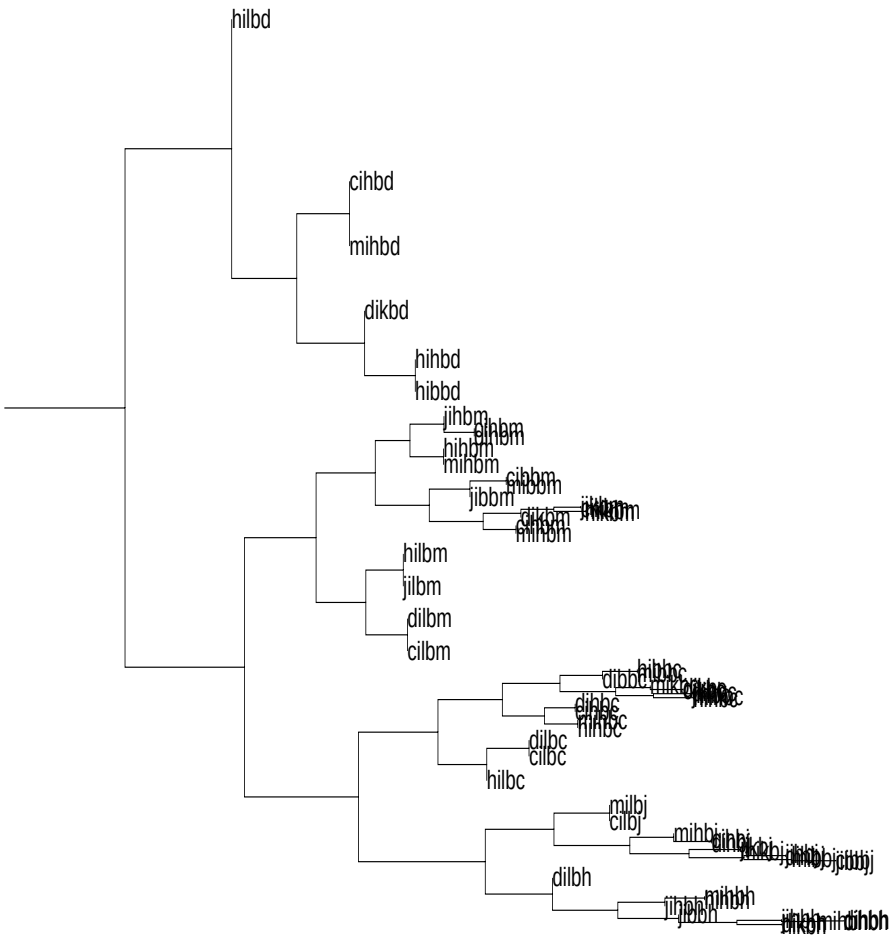

Figure A.46: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation after 100 iterations.

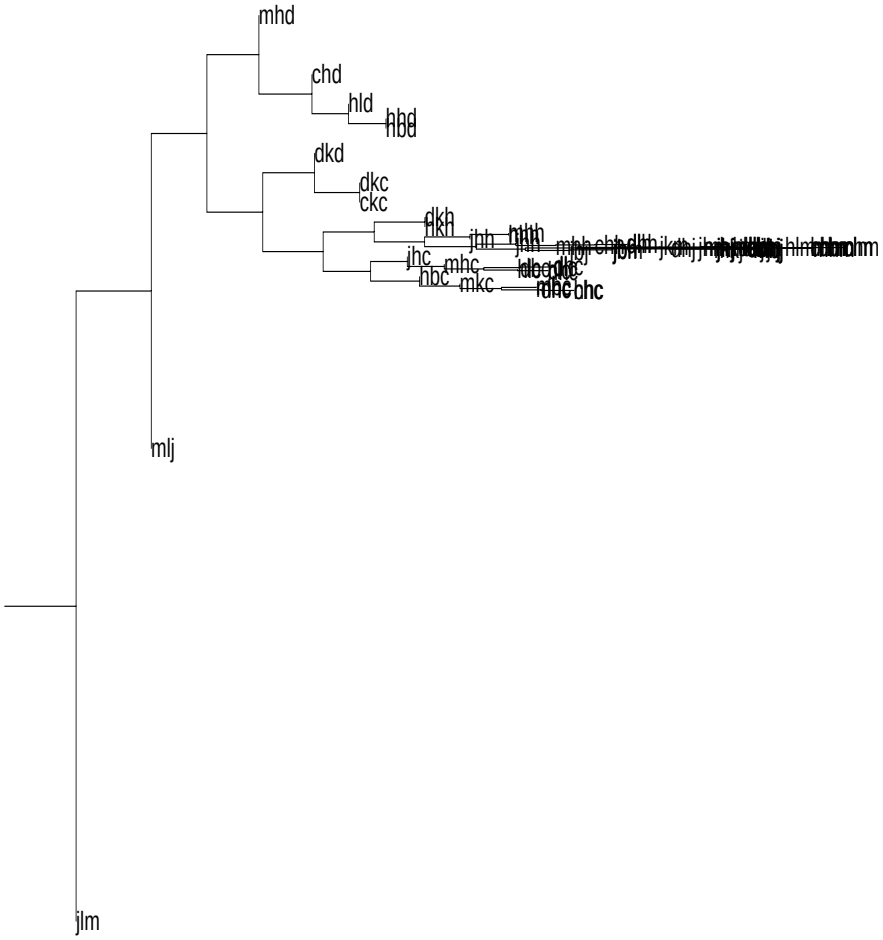

Figure A.47: Top branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation after 200 iterations.

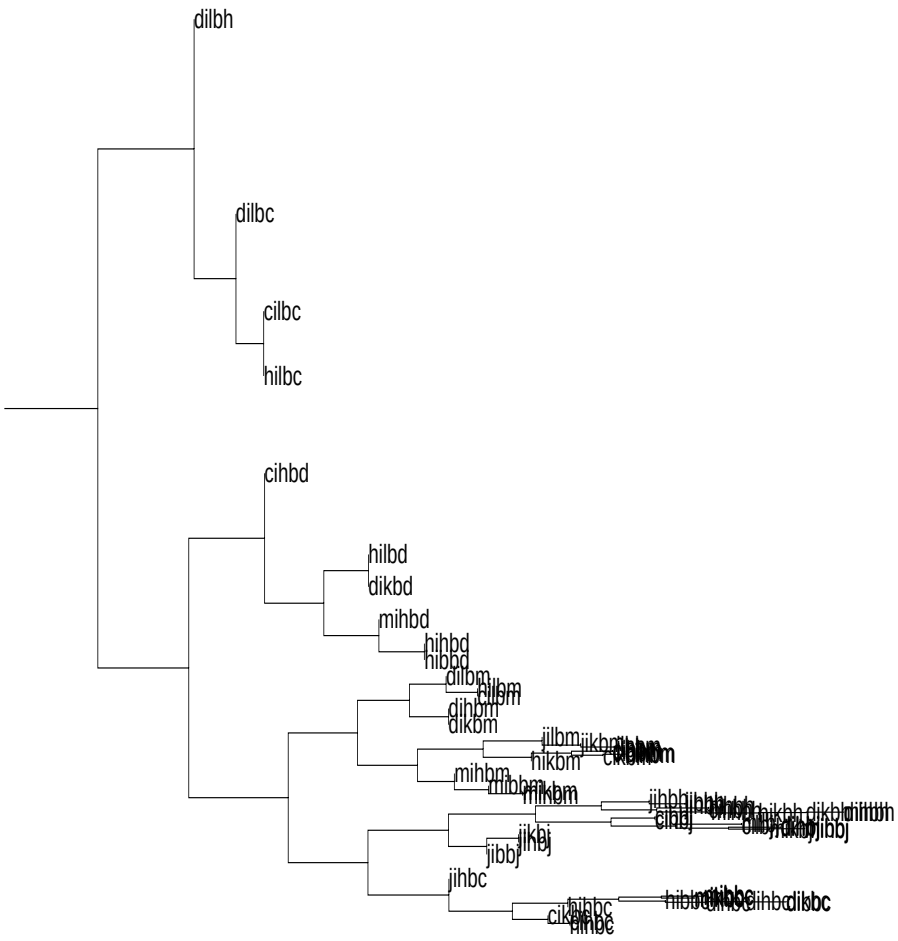

Figure A.48: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with back-propagation, hyperbolic tangent units and the 2 unit representation after 200 iterations.

A.2 Networks trained with Kwasny and Kalman's method

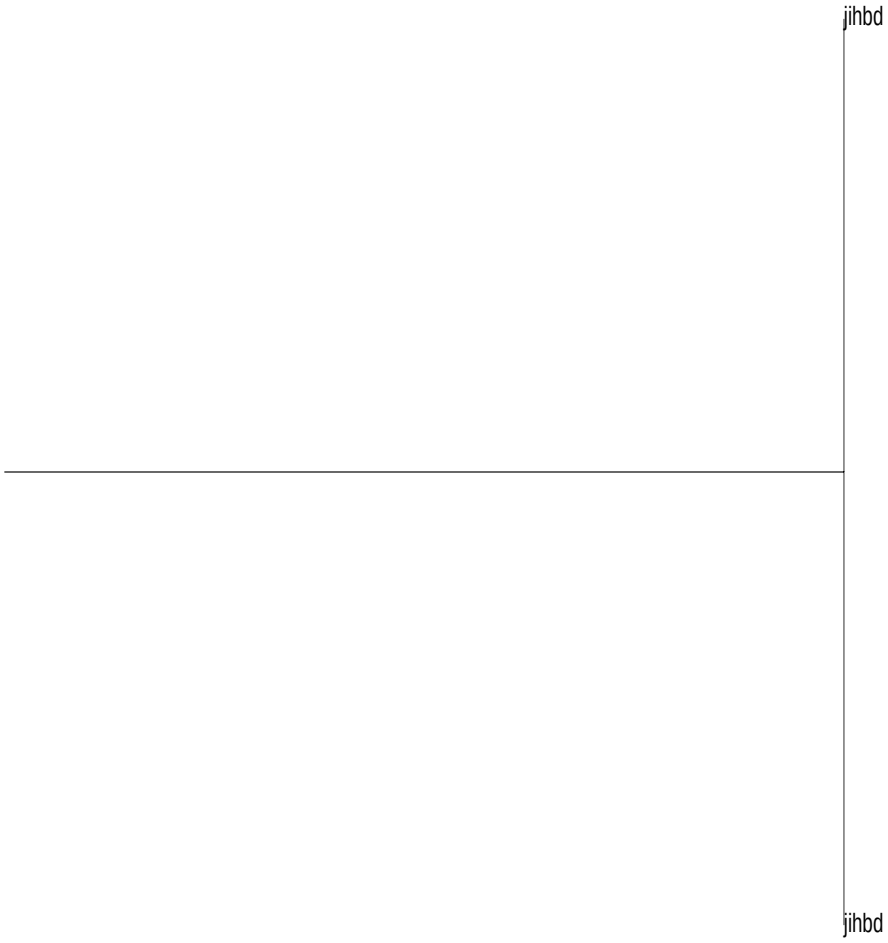

Figure A.49: Top branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 2 unit representation.

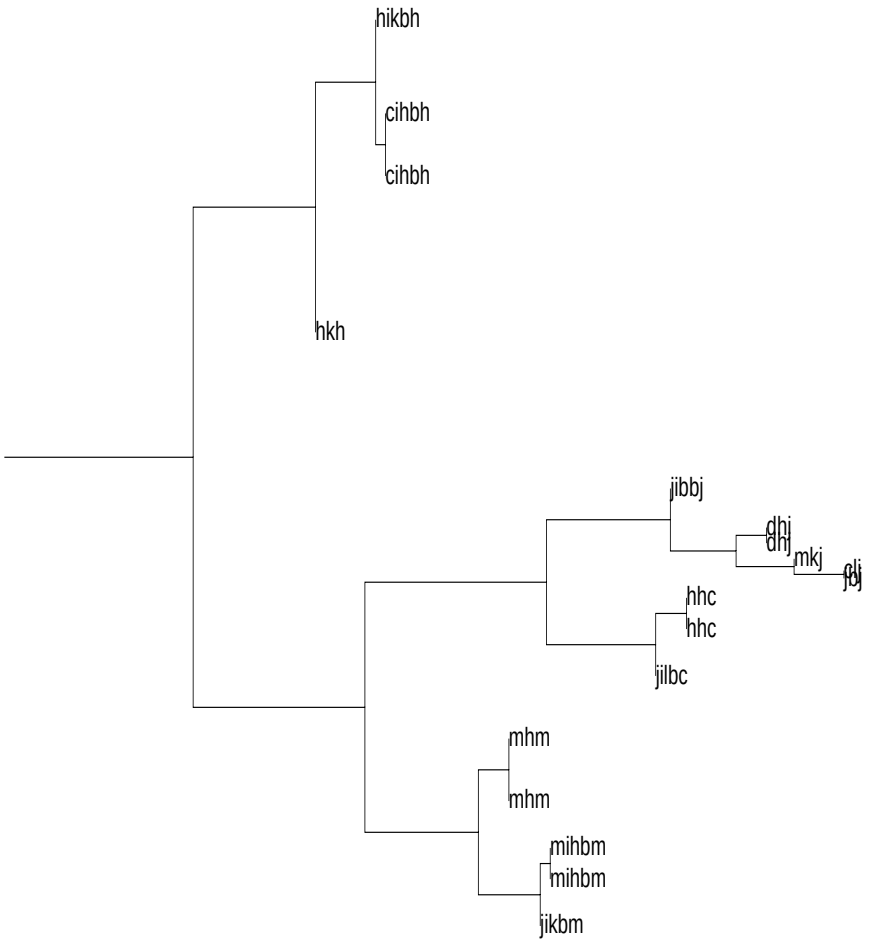

Figure A.50: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 2 unit representation.

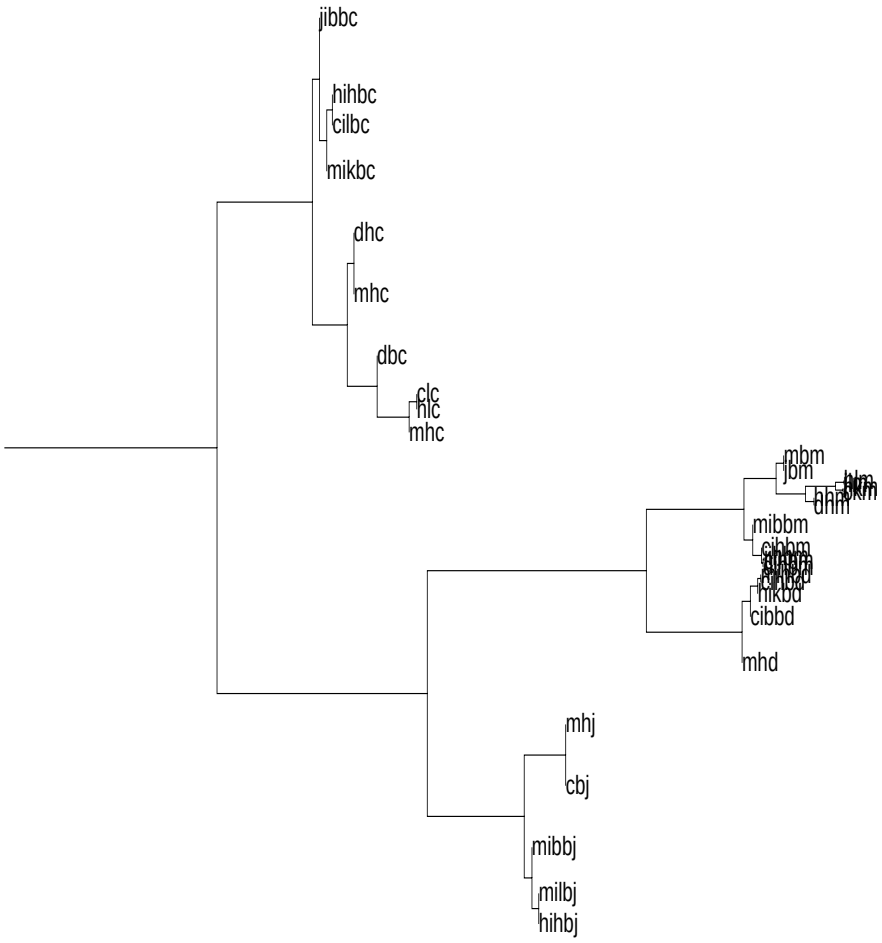

Figure A.51: Top branch of the HCA plot for the SRAAM trained on 40 sequences with Kwasny and Kalman's method and the 2 unit representation.

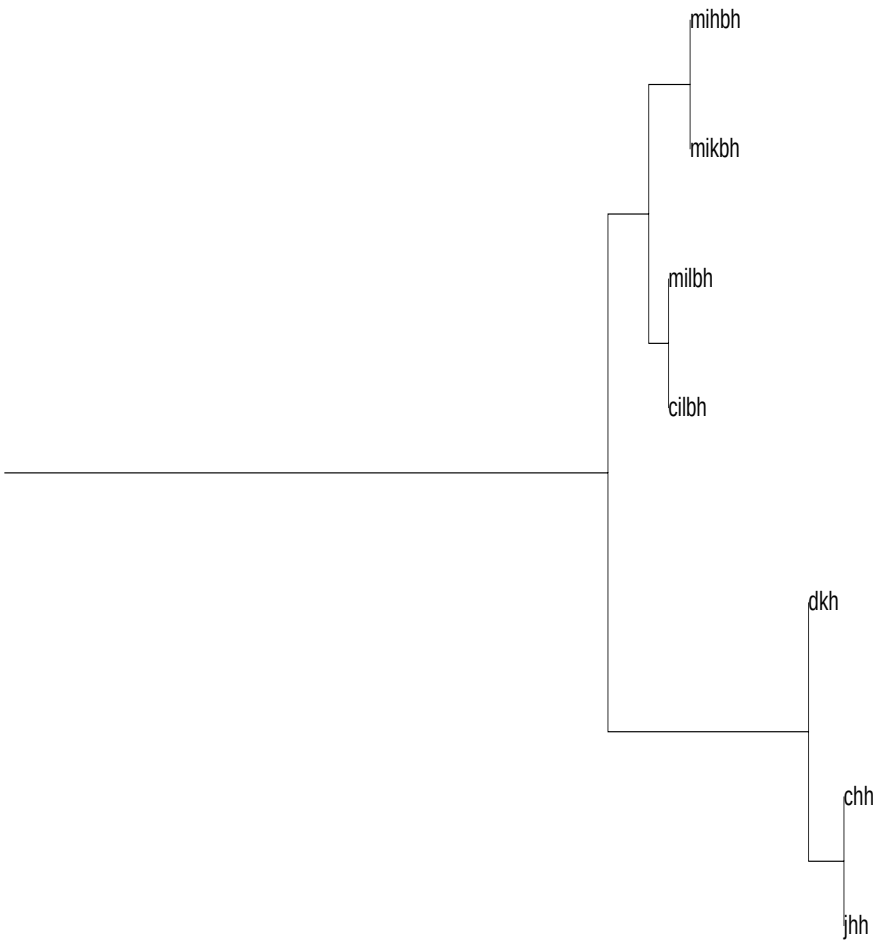

Figure A.52: Bottom branch of the HCA plot for the SRAAM trained on 40 sequences with Kwasny and Kalman's method and the 2 unit representations.

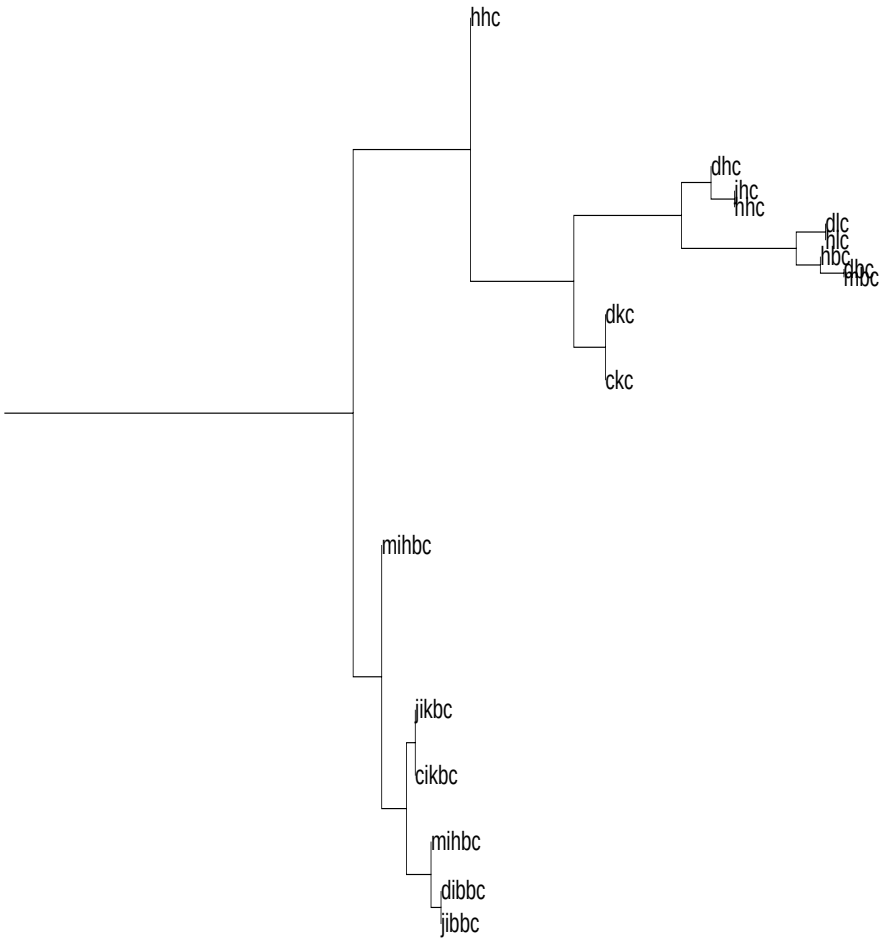

Figure A.53: Top branch of the HCA plot for the SRAAM trained on 80 sequences with Kwasny and Kalman's method and the 2 unit representation.

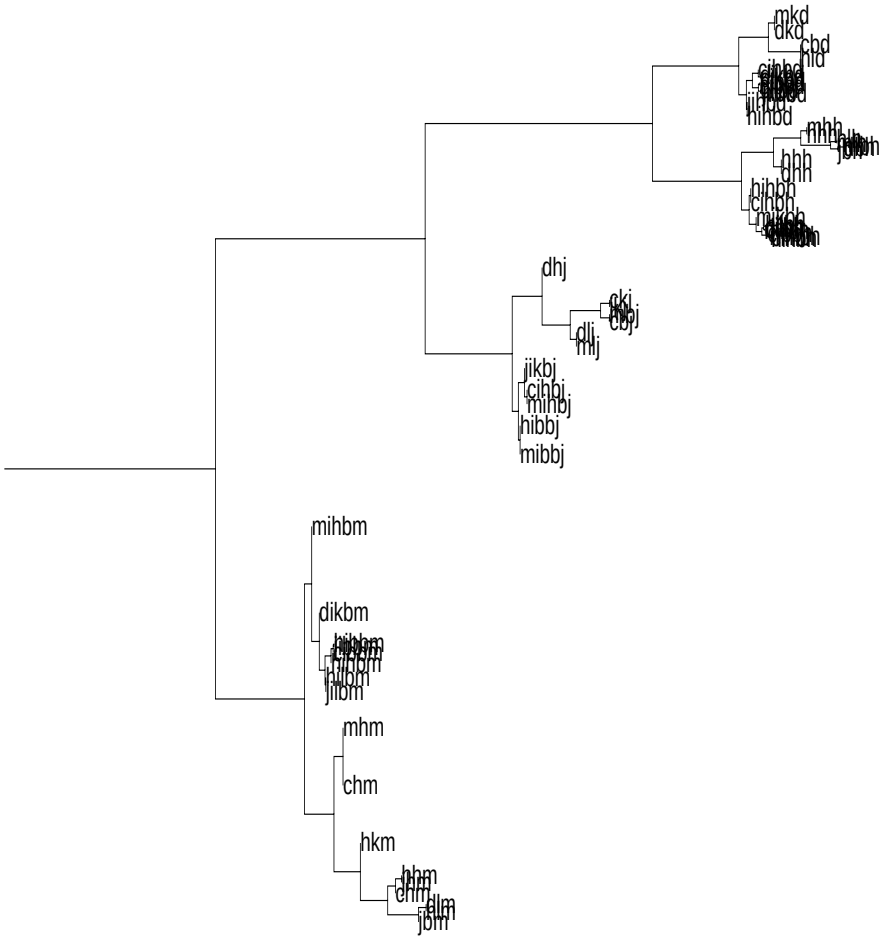

Figure A.54: Bottom branch of the HCA plot for the SRAAM trained on 80 sequences with Kwasny and Kalman's method and the 2 unit representation.

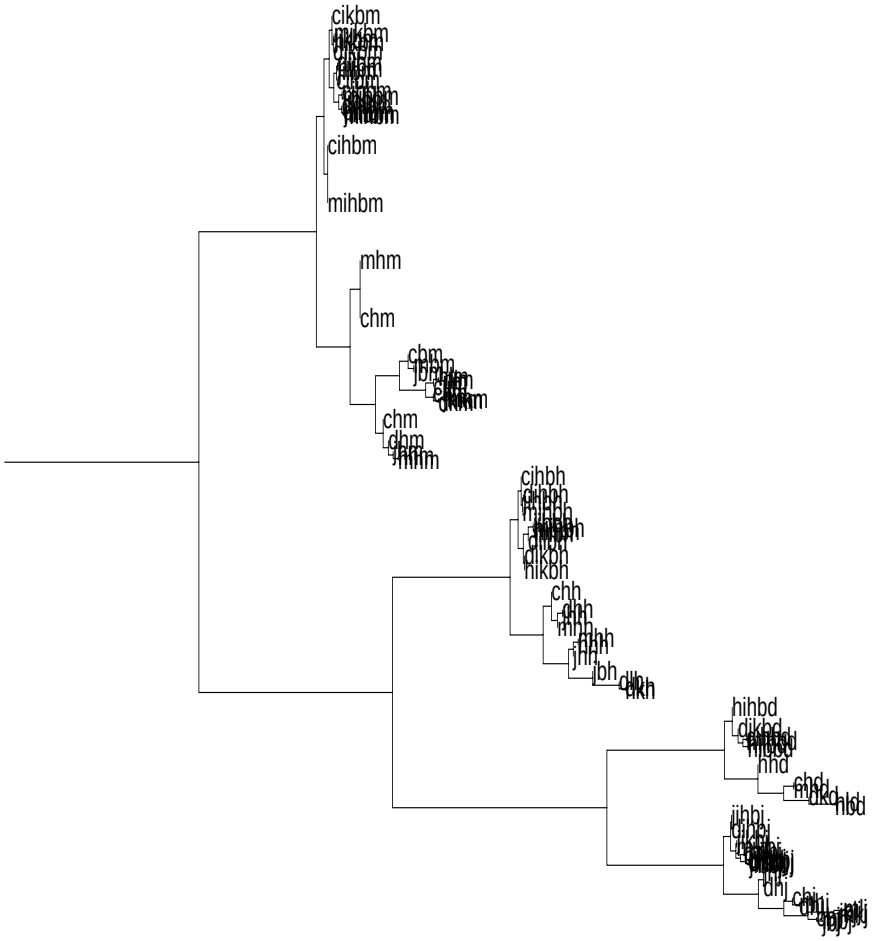

Figure A.55: Top branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 2 unit representation.

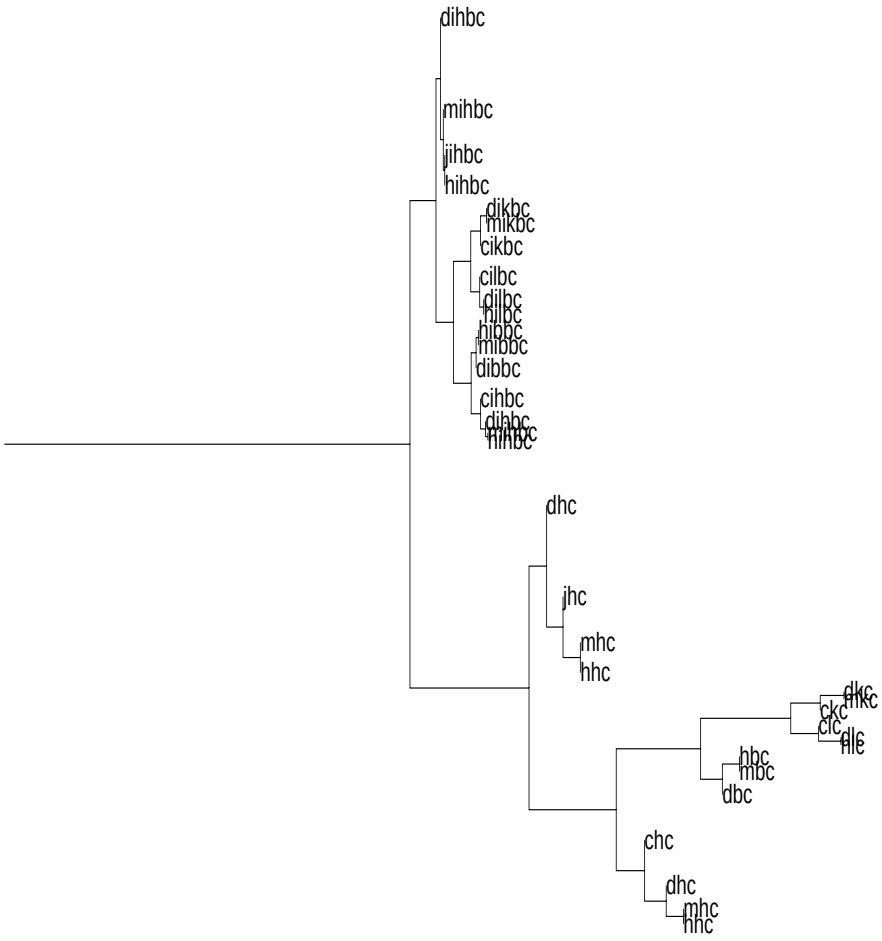

Figure A.56: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 2 unit representation.

A.2.1 Partially trained networks

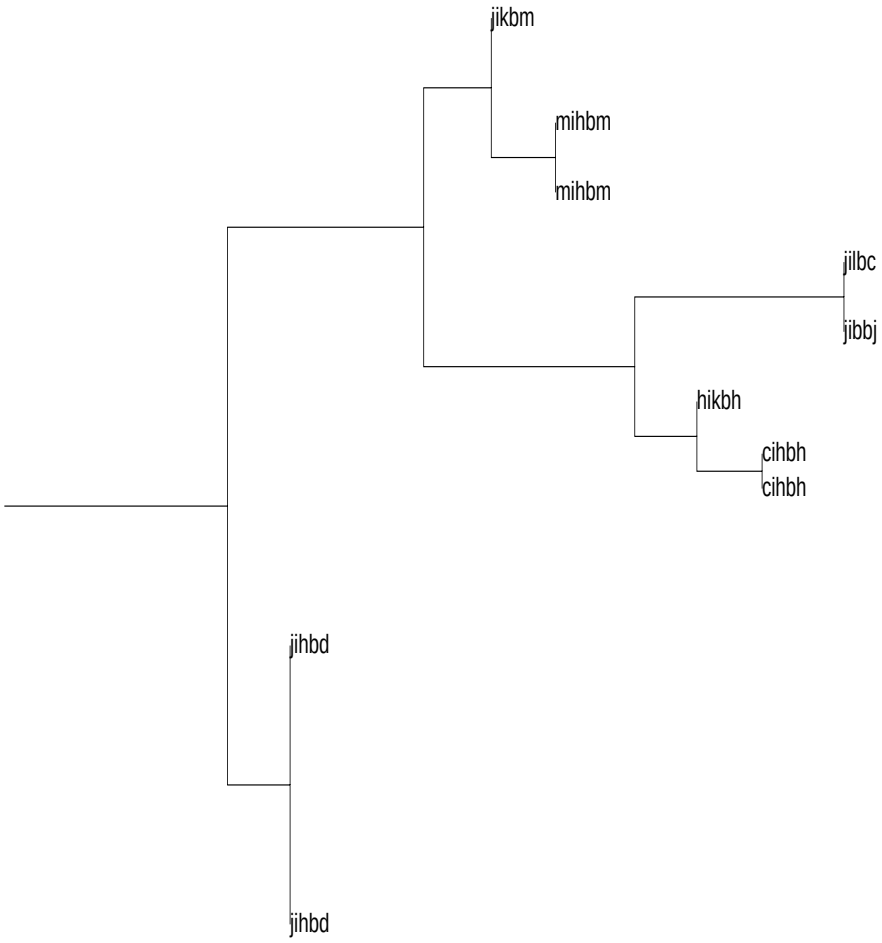

Figure A.57: Top branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 2 unit representation after training to a tolerance of 0.16.

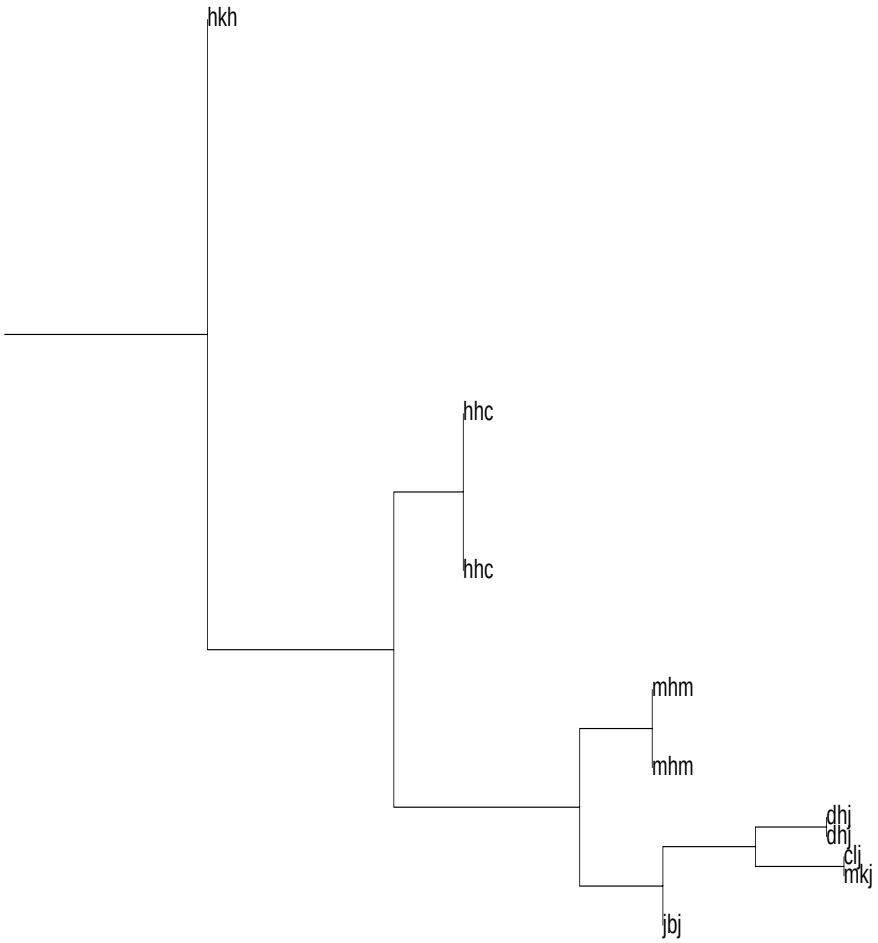

Figure A.58: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 2 unit representation after training to a tolerance of 0.16.

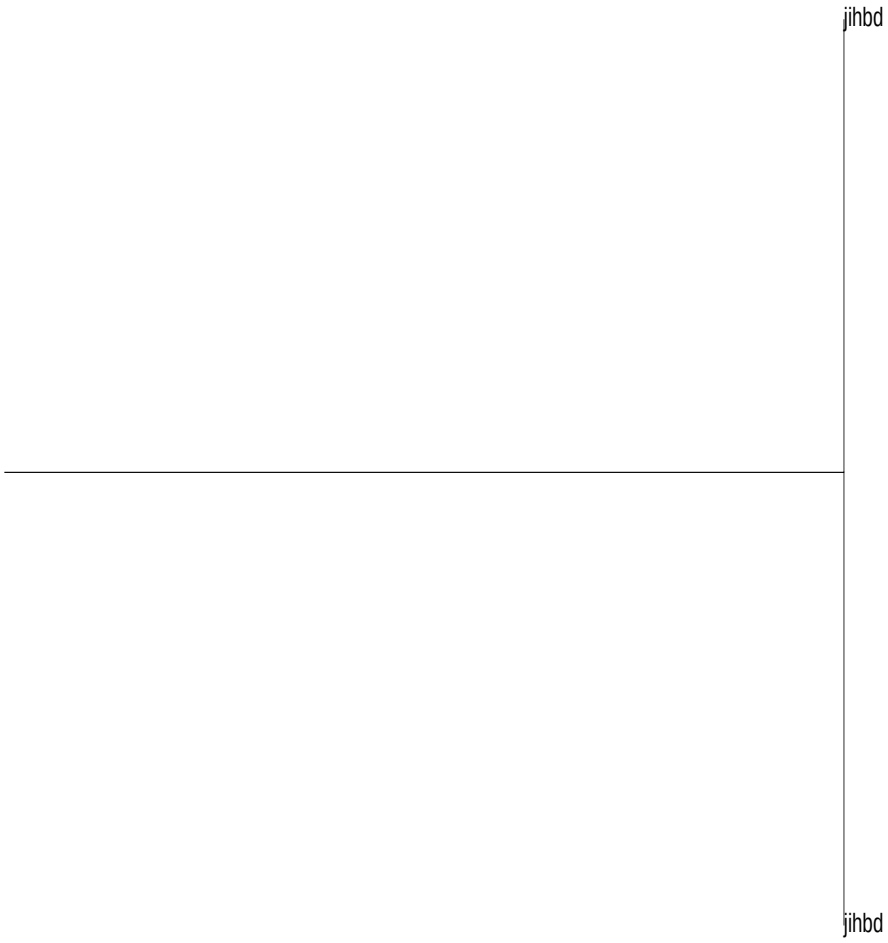

Figure A.59: Top branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 2 unit representation after training to a tolerance of 0.04.

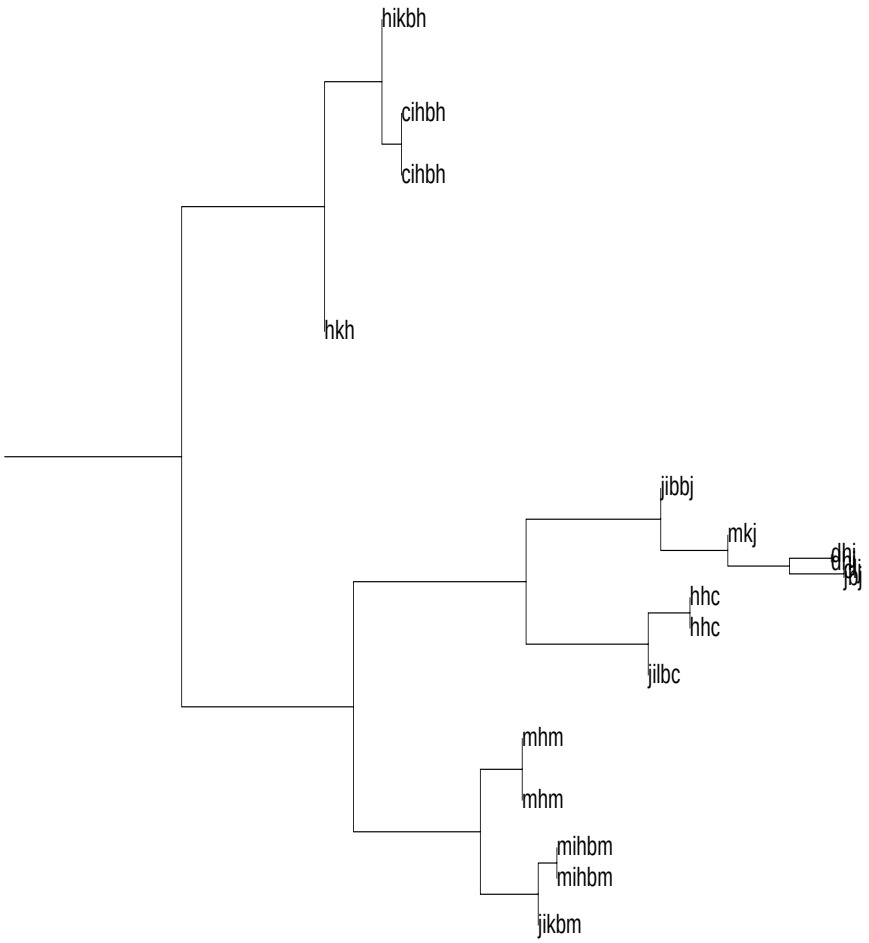

Figure A.60: Bottom branch of the HCA plot for the SRAAM trained on 20 sequences with Kwasny and Kalman's method and the 2 unit representation after training to a tolerance of 0.04.

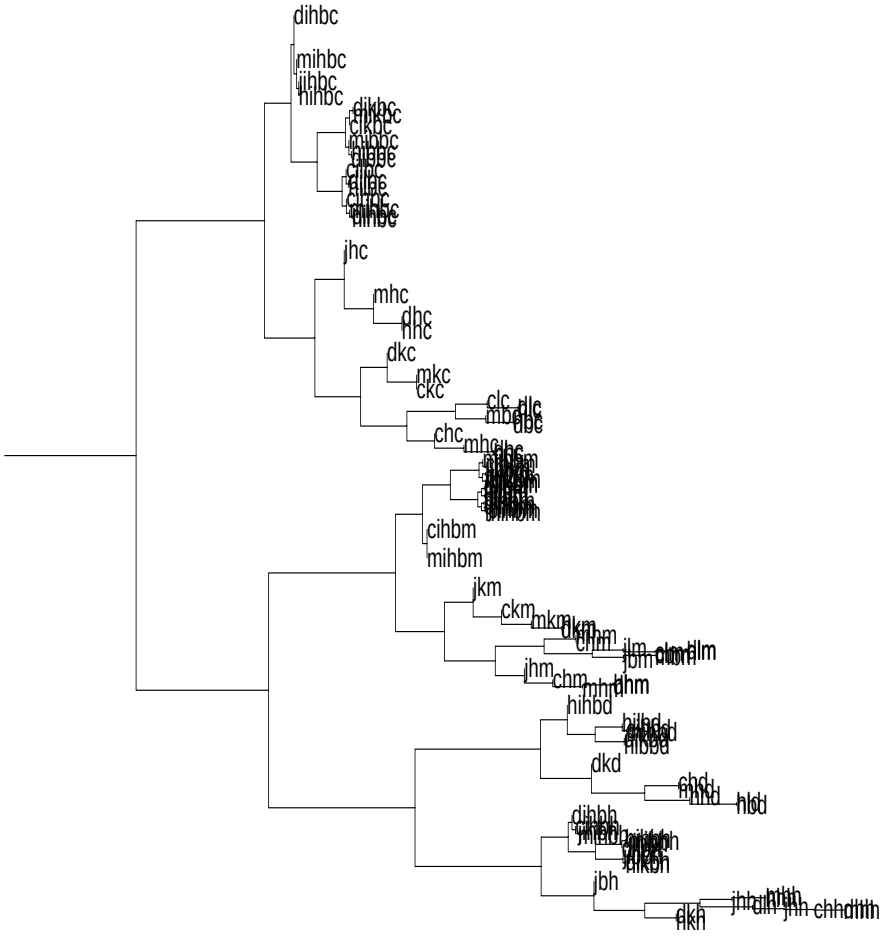

Figure A.61: Top branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 2 unit representation after training to a tolerance of 0.16.

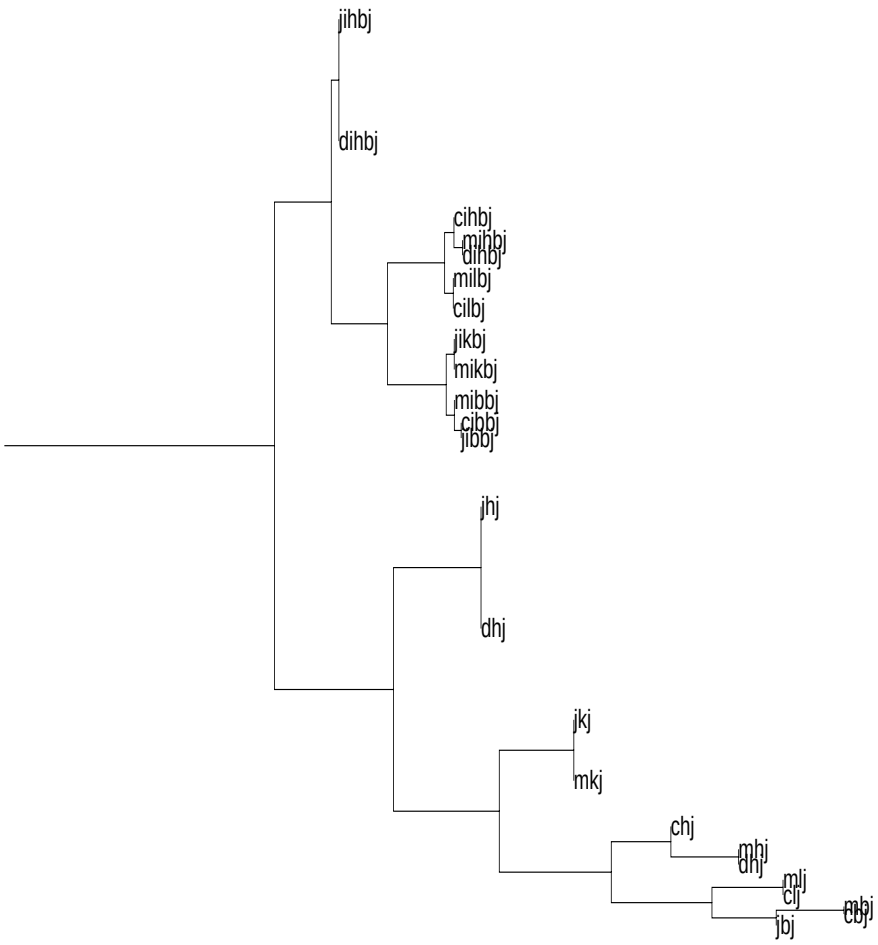

Figure A.62: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 2 unit representation after training to a tolerance of 0.16.

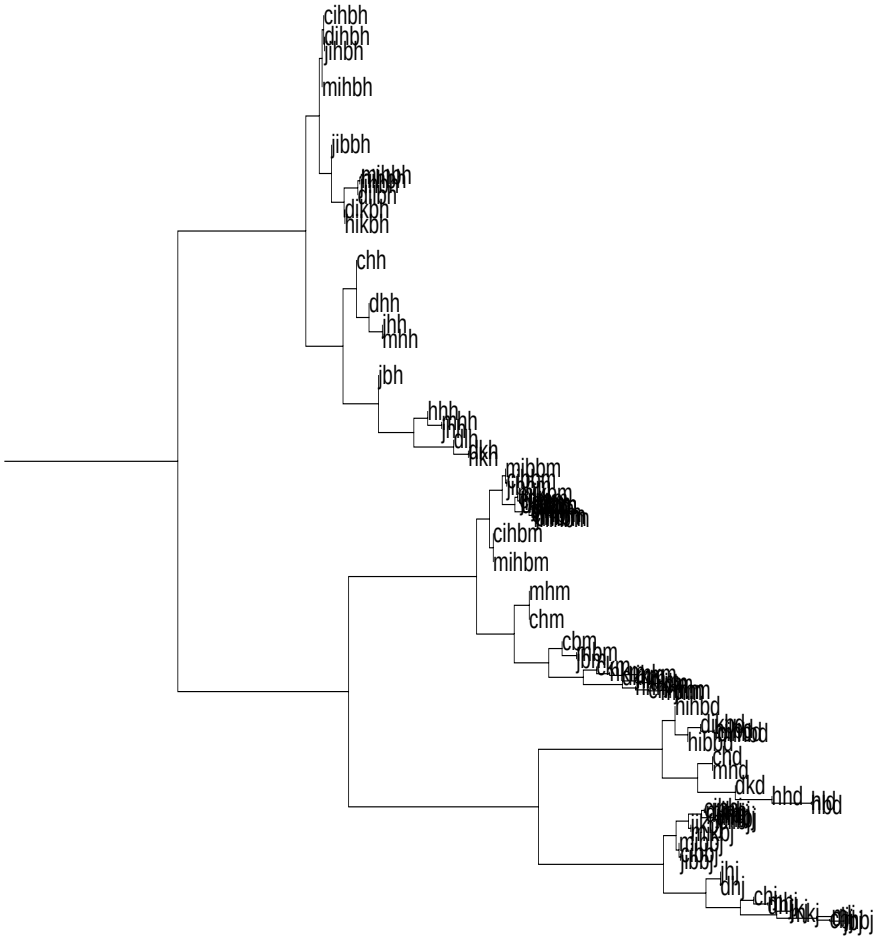

Figure A.63: Top branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 2 unit representation after training to a tolerance of 0.04.

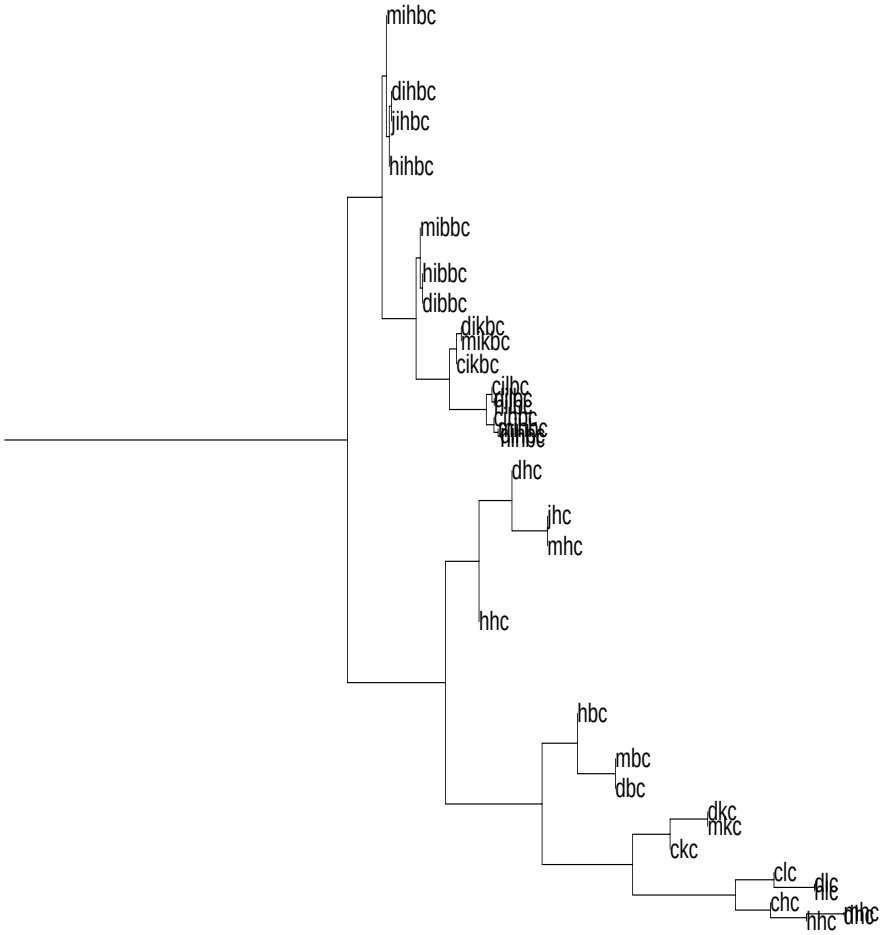

Figure A.64: Bottom branch of the HCA plot for the SRAAM trained on 130 sequences with Kwasny and Kalman's method and the 2 unit representation after training to a tolerance of 0.04.

Appendix B

Representational Lesioning Graphs

B.1 Networks trained by back-propagation

B.1.1 Sigmoidal Units

Figures B.1 to B.16 show the results of lesioning with the SRAAMs trained on the 2 bit representation with sigmoidal units.

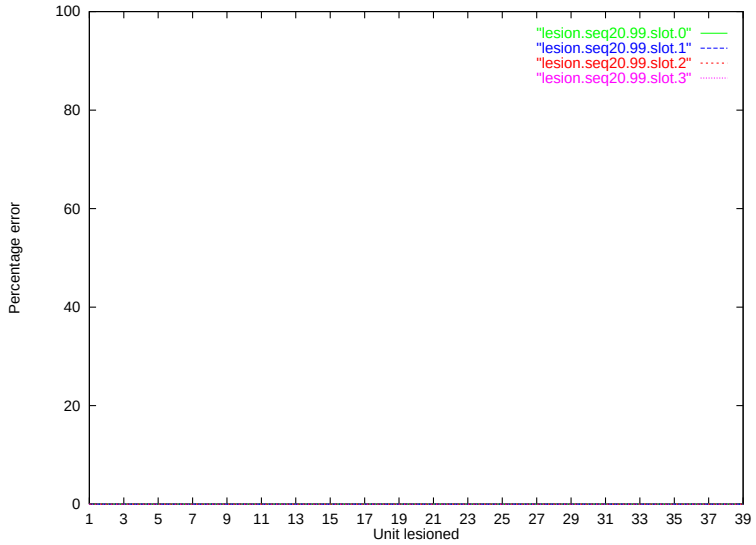

Figure B.1: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 20 sequences and the 2 unit representation, using 1% noise. No errors were produced.

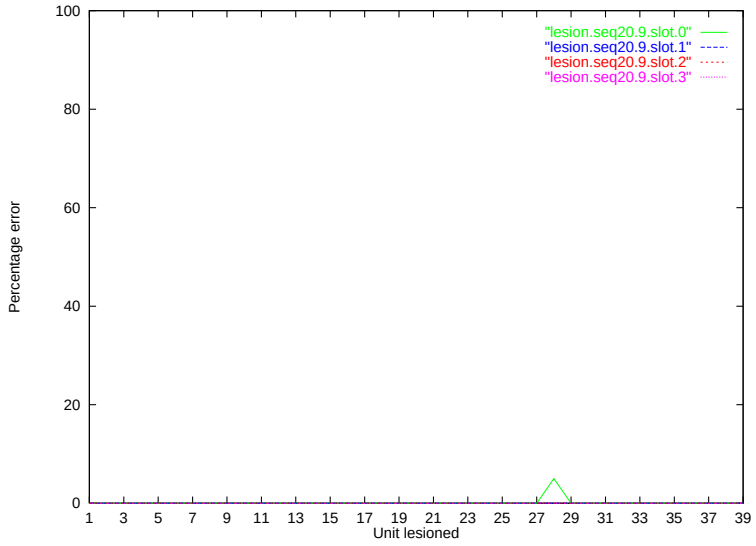

Figure B.2: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 20 sequences and the 2 unit representation, using 10% noise.

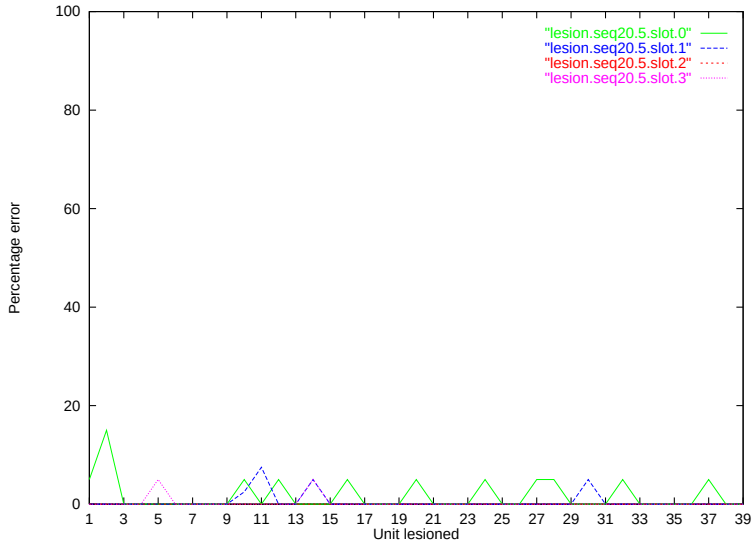

Figure B.3: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 20 sequences and the 2 unit representation, using 50% noise.

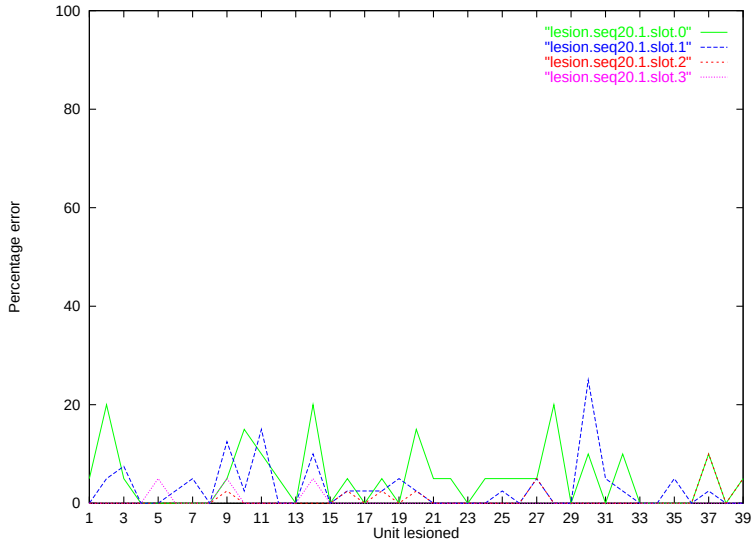

Figure B.4: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 20 sequences and the 2 unit representation, using 90% noise.

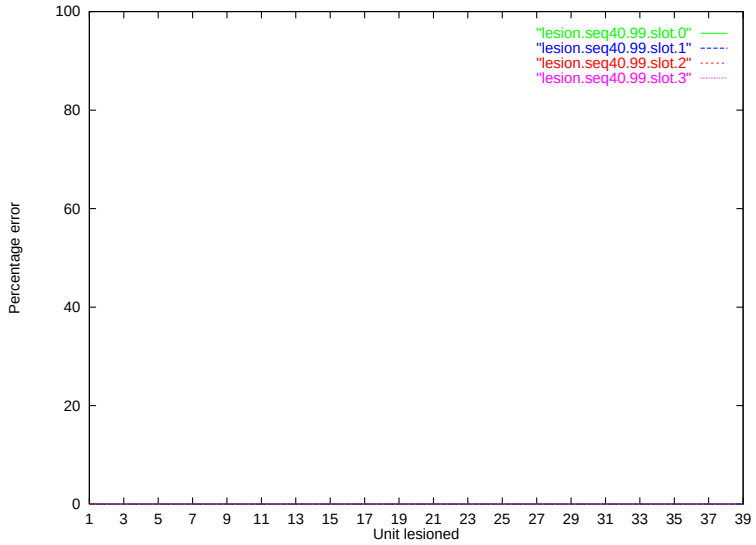

Figure B.5: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 40 sequences and the 2 unit representation, using 1% noise. No errors were produced.

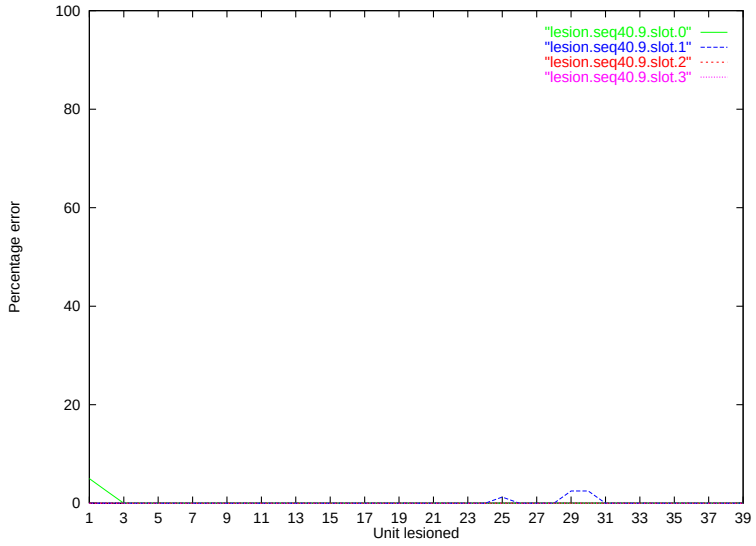

Figure B.6: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 40 sequences and the 2 unit representation, using 10% noise.

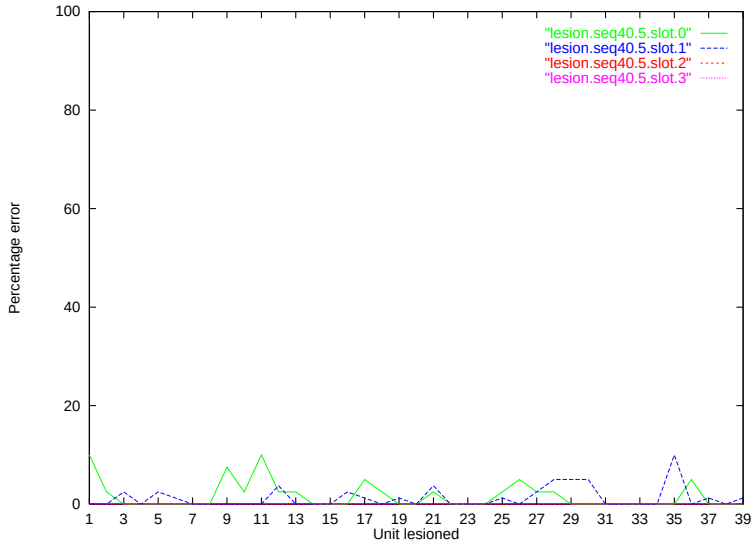

Figure B.7: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 40 sequences and the 2 unit representation, using 50% noise.

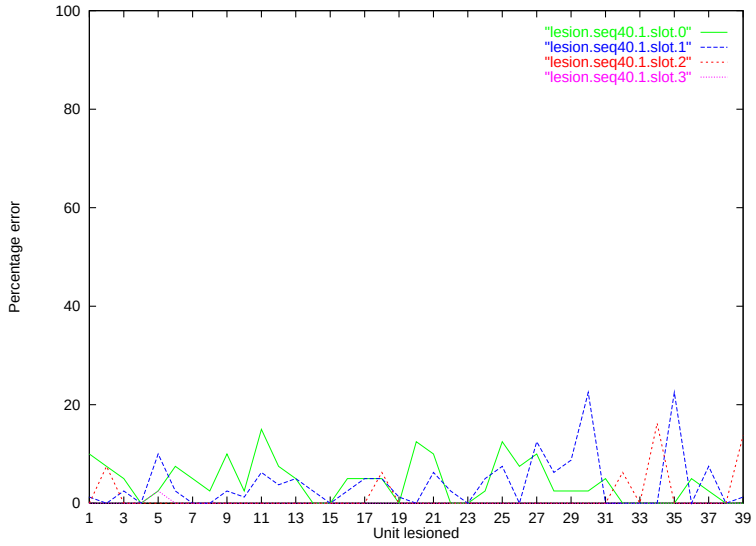

Figure B.8: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 40 sequences and the 2 unit representation, using 90% noise.

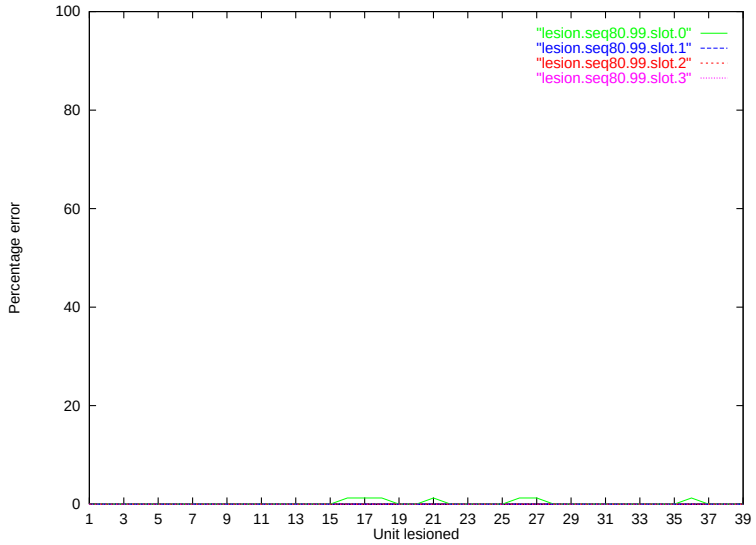

Figure B.9: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 80 sequences and the 2 unit representation, using 1% noise.

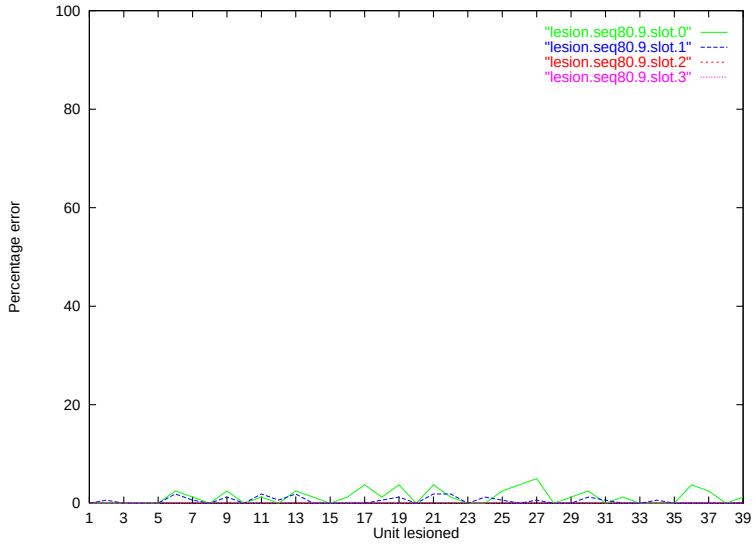

Figure B.10: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 80 sequences and the 2 unit representation, using 10% noise.

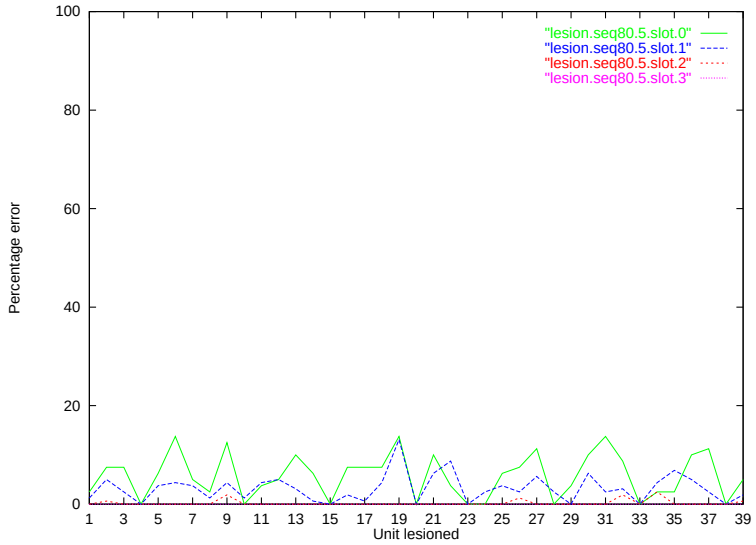

Figure B.11: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 80 sequences and the 2 unit representation, using 50% noise.

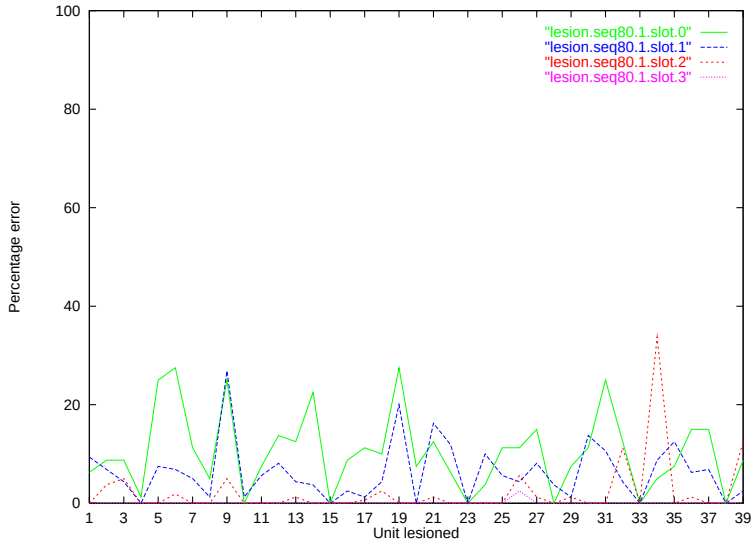

Figure B.12: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 80 sequences and the 2 unit representation, using 90% noise.

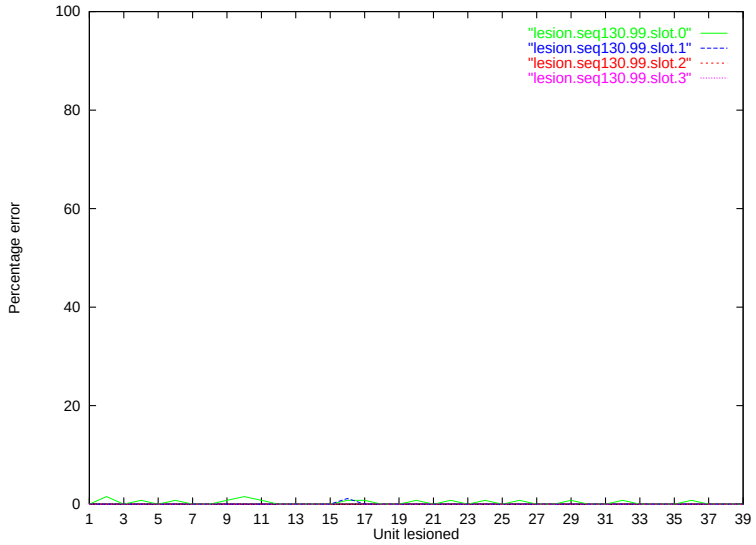

Figure B.13: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 130 sequences and the 2 unit representation, using 1% noise.

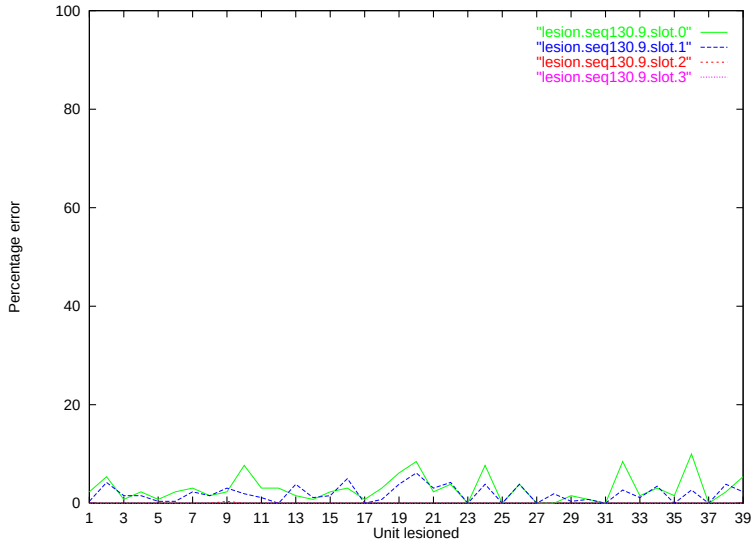

Figure B.14: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 130 sequences and the 2 unit representation, using 10% noise.

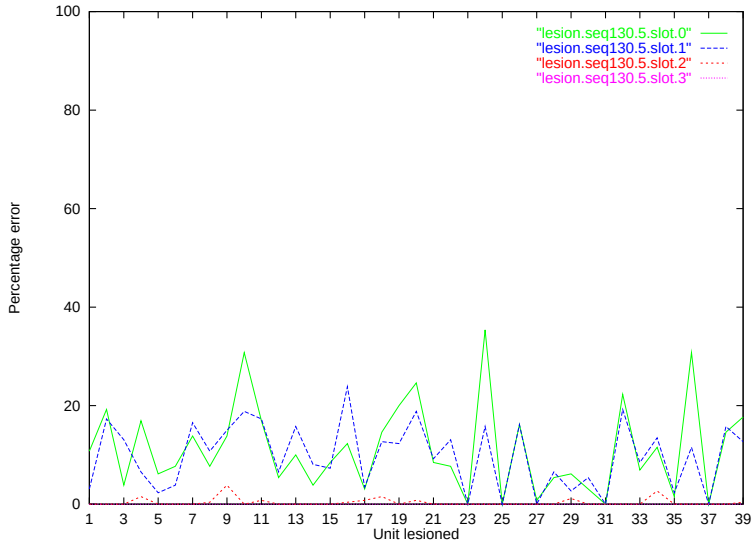

Figure B.15: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 130 sequences and the 2 unit representation, using 50% noise.

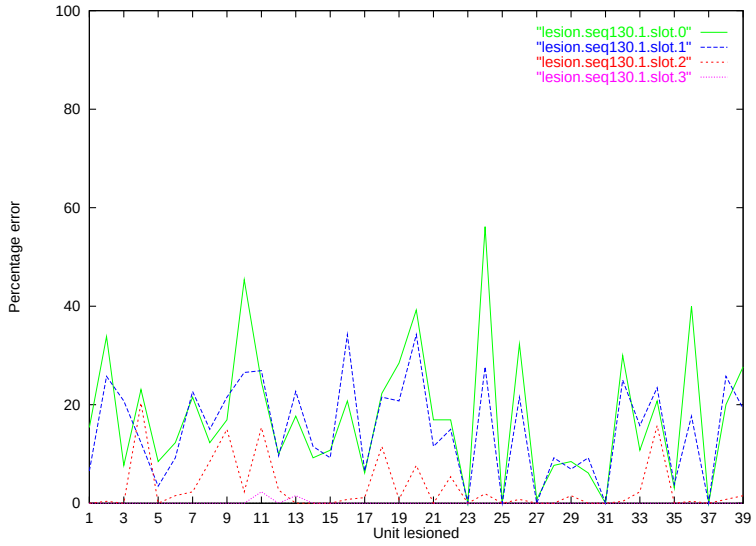

Figure B.16: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and sigmoidal units on 130 sequences and the 2 unit representation, using 90% noise.

B.1.2 Hyperbolic Tangent Units

Figures B.17 to B.21 show the results of the lesioning experiment for the SRAAMs with hyperbolic tangent units trained via back-propagation on the 1 unit representation and 40 and 80 sequences. Figures B.25 to B.40 show the results of lesioning with the SRAAMs trained on the 2 bit representation with hyperbolic tangent units.

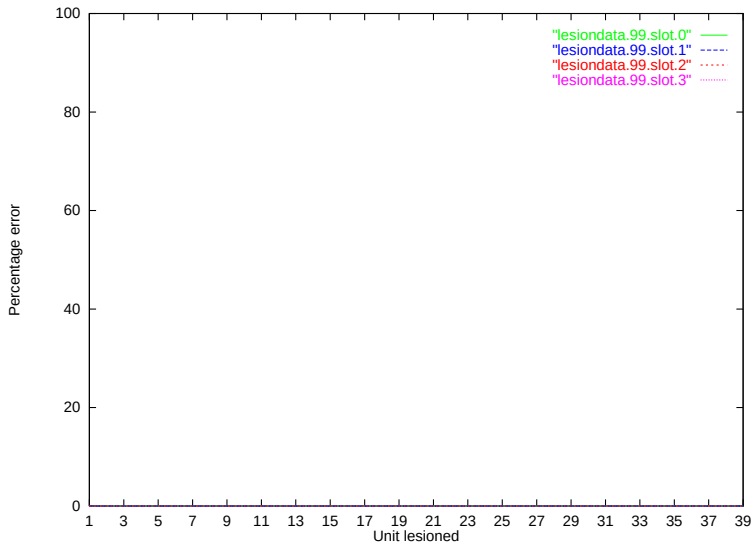

Figure B.17: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 40 sequences and the 1 unit representation, using 1% noise. No errors were produced.

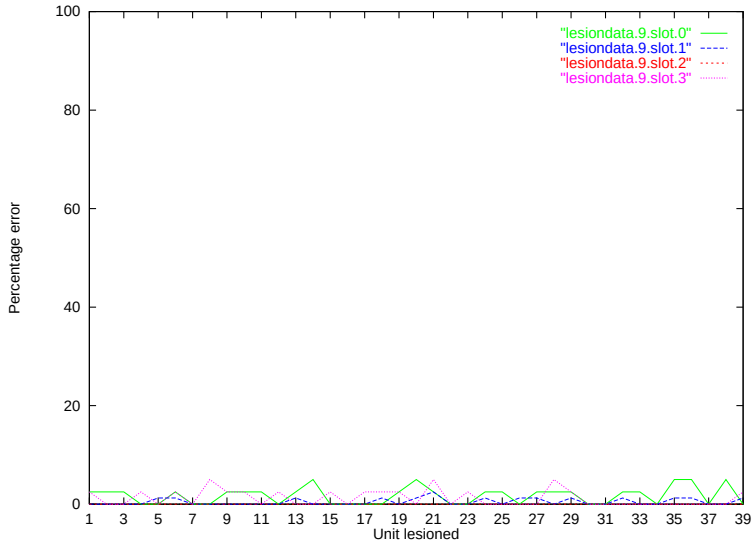

Figure B.18: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 40 sequences and the 1 unit representation, using 10% noise.

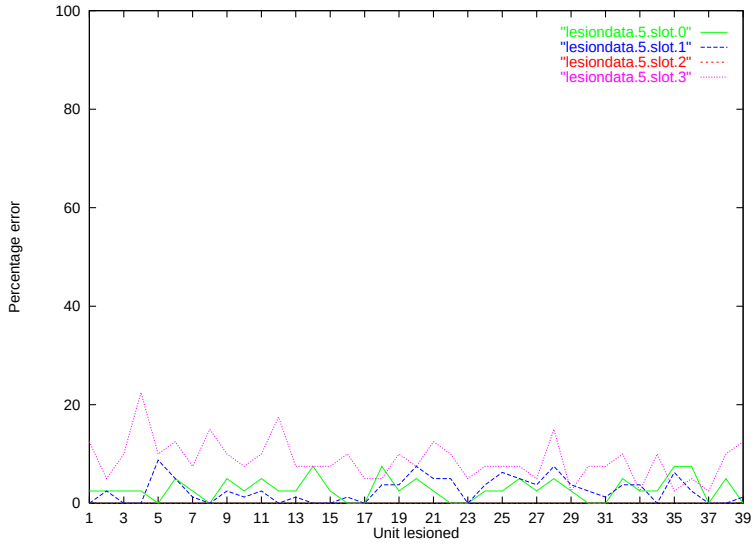

Figure B.19: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 40 sequences and the 1 unit representation, using 50% noise.

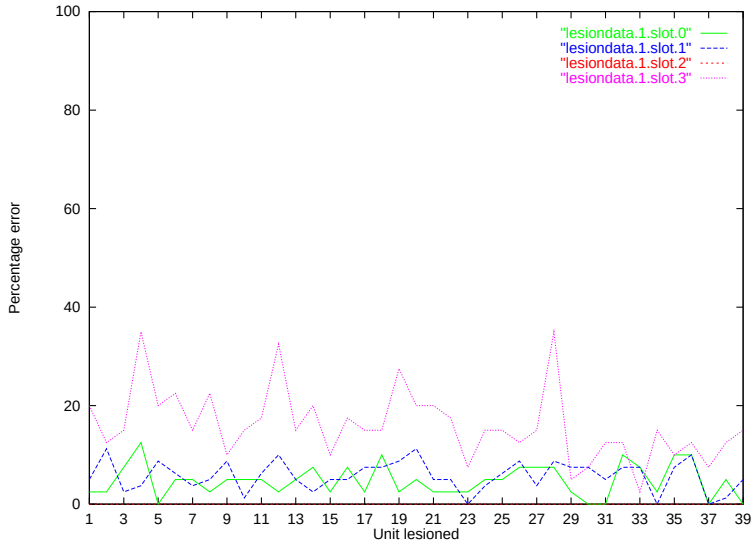

Figure B.20: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 40 sequences and the 1 unit representation, using 90% noise.

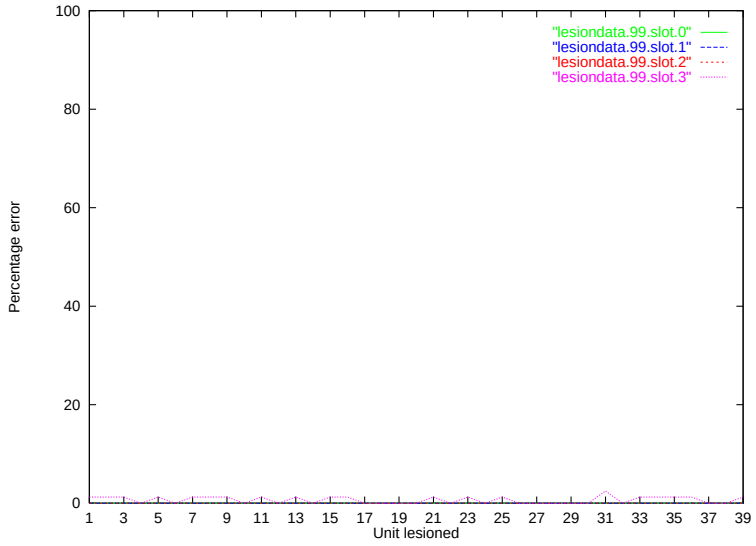

Figure B.21: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 80 sequences and the 1 unit representation, using 1% noise.

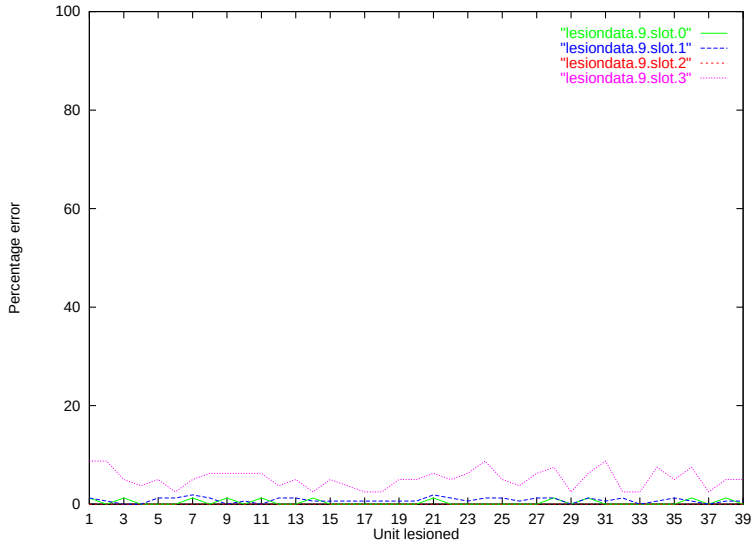

Figure B.22: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 80 sequences and the 1 unit representation, using 10% noise.

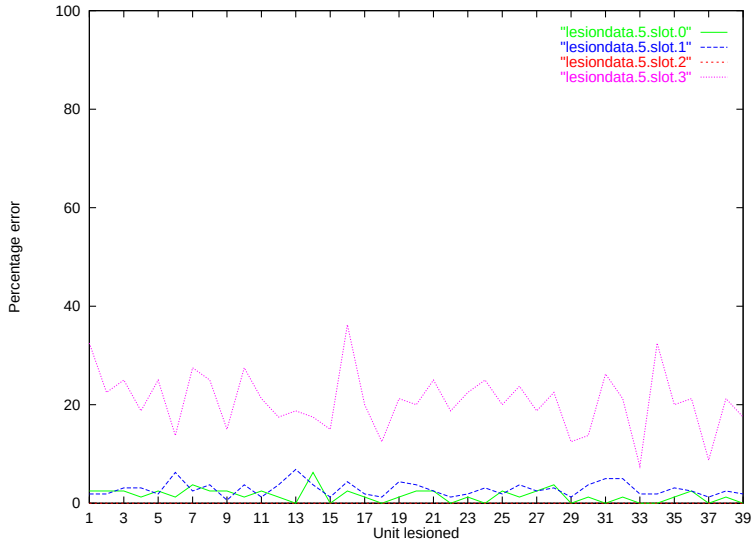

Figure B.23: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 80 sequences and the 1 unit representation, using 50% noise.

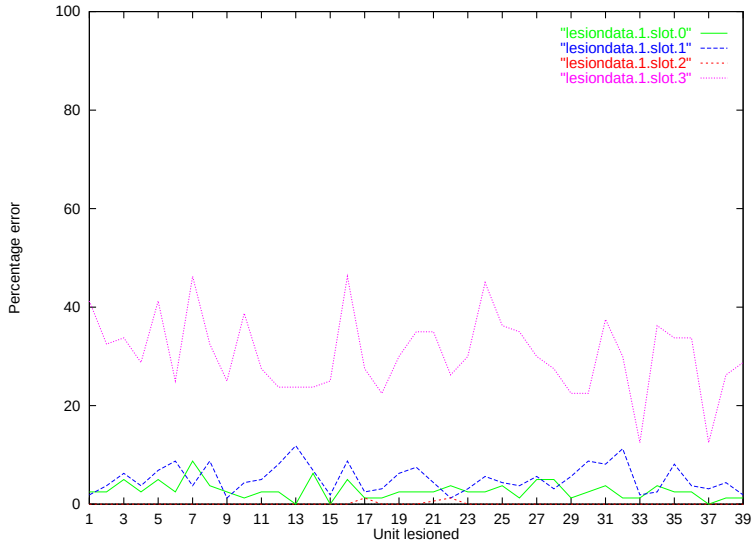

Figure B.24: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 80 sequences and the 1 unit representation, using 90% noise.

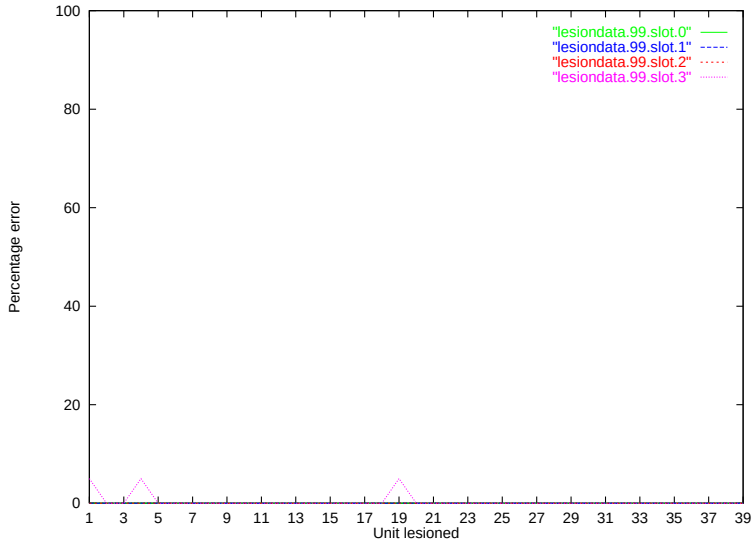

Figure B.25: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 20 sequences and the 2 unit representation, using 1% noise.

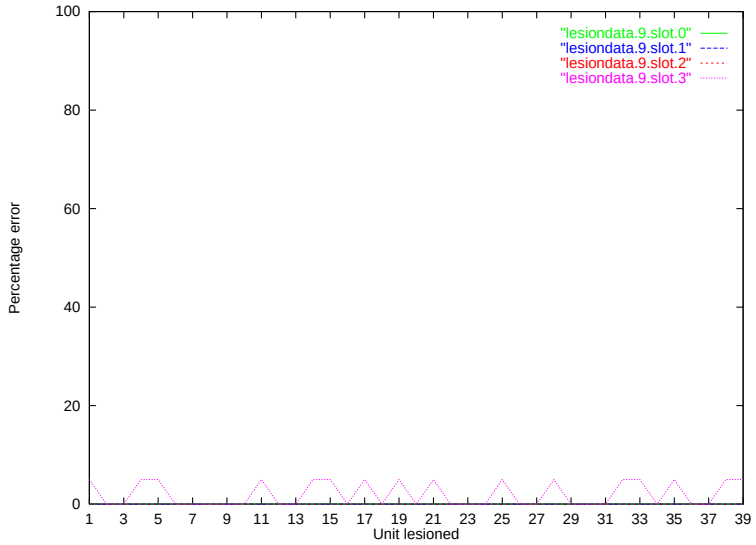

Figure B.26: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 20 sequences and the 2 unit representation, using 10% noise.

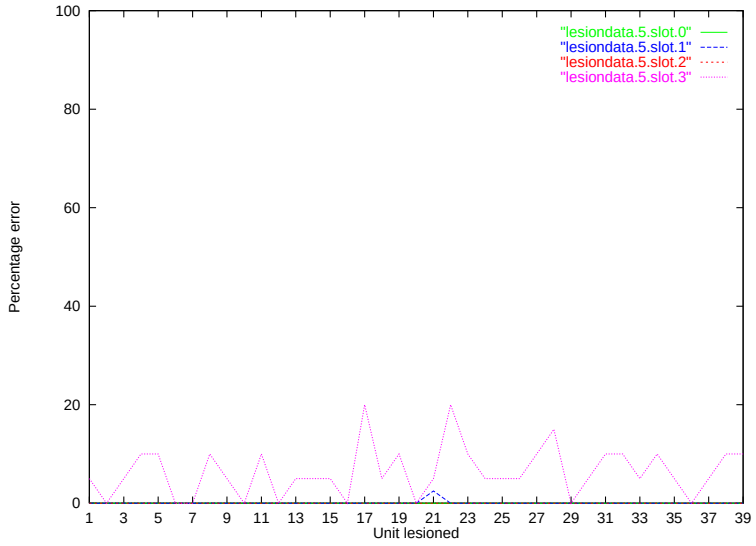

Figure B.27: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 20 sequences and the 2 unit representation, using 50% noise.

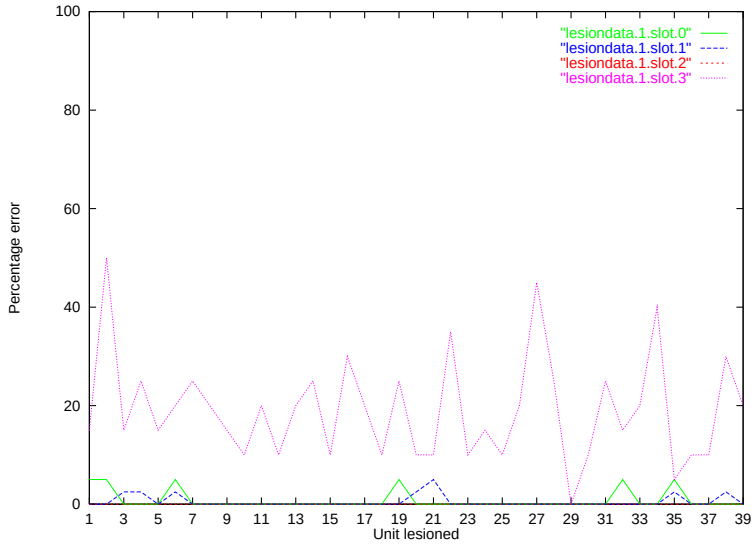

Figure B.28: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 20 sequences and the 2 unit representation, using 90% noise.

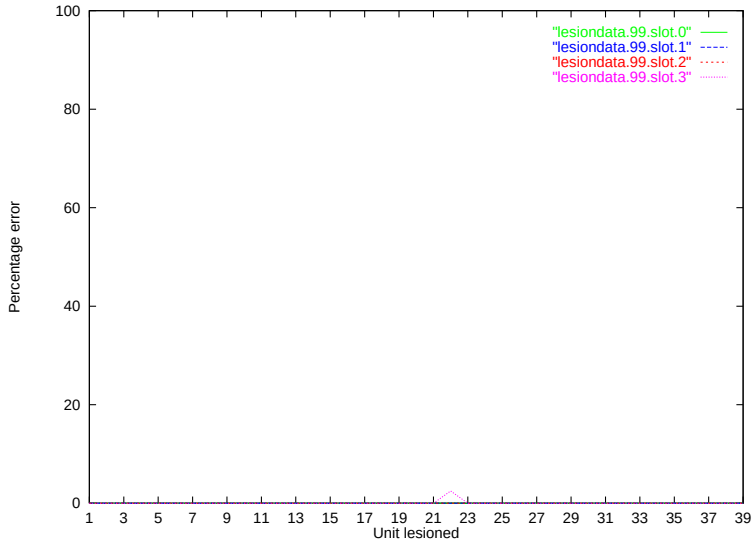

Figure B.29: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 40 sequences and the 2 unit representation, using 1% noise.

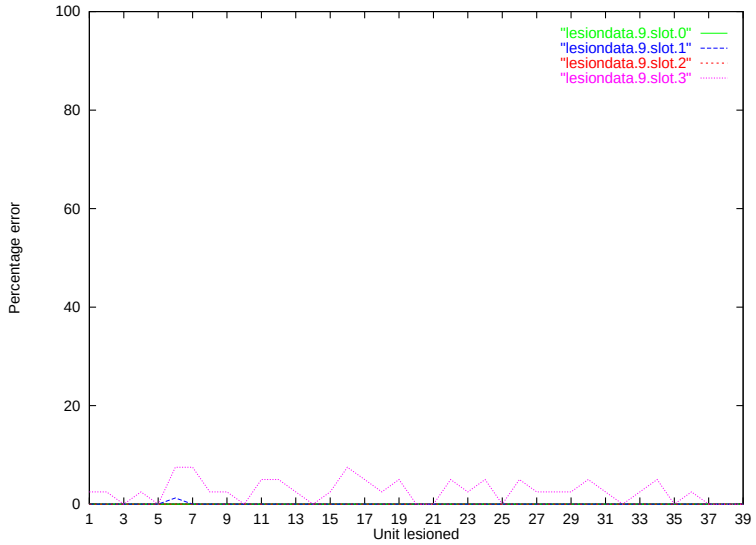

Figure B.30: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 40 sequences and the 2 unit representation, using 10% noise.

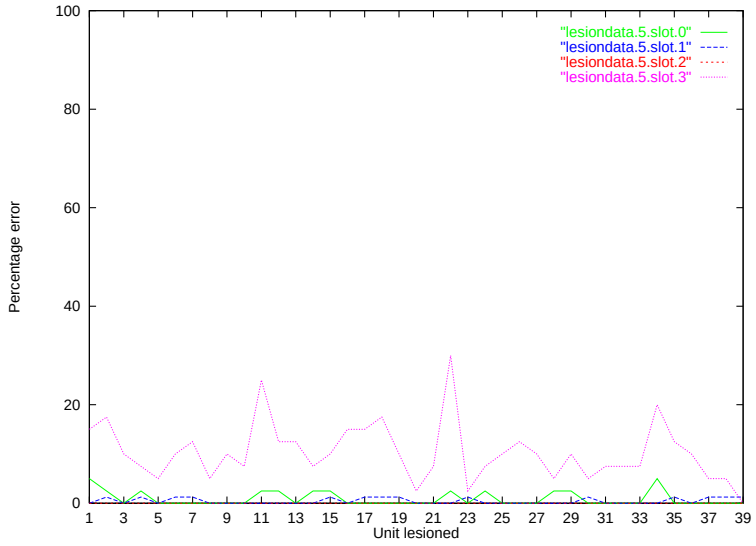

Figure B.31: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 40 sequences and the 2 unit representation, using 50% noise.

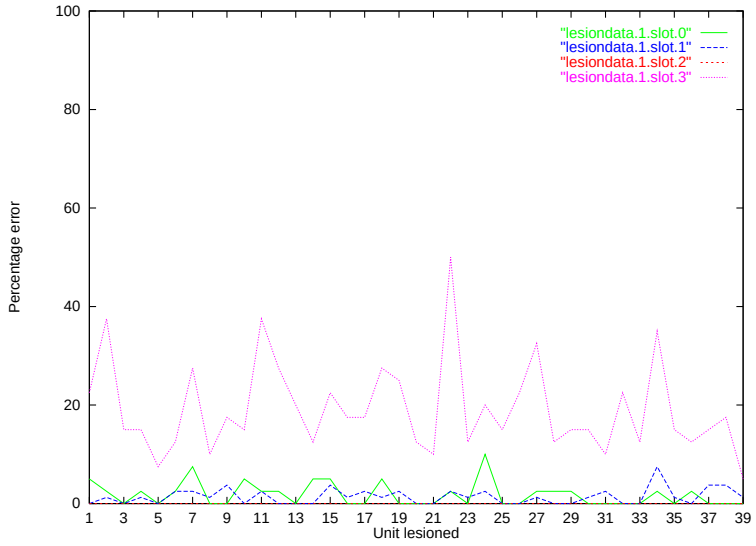

Figure B.32: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 40 sequences and the 2 unit representation, using 90% noise.

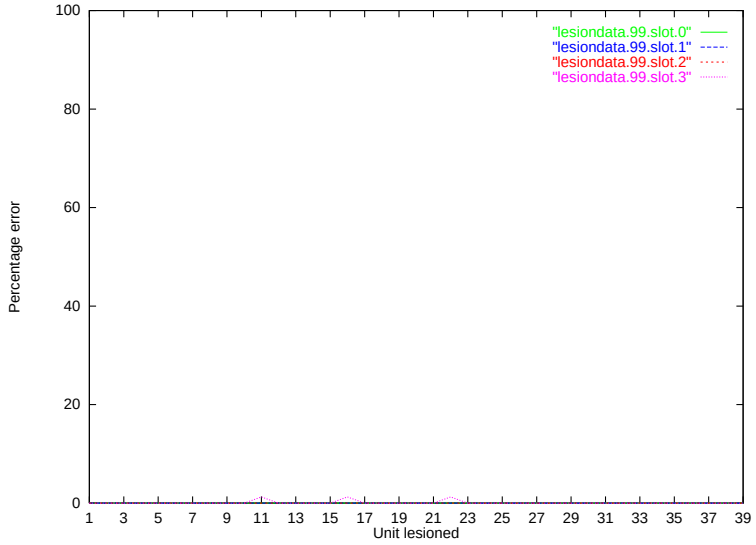

Figure B.33: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 80 sequences and the 2 unit representation, using 1% noise. Some error was produced at units 11,16 and 20.

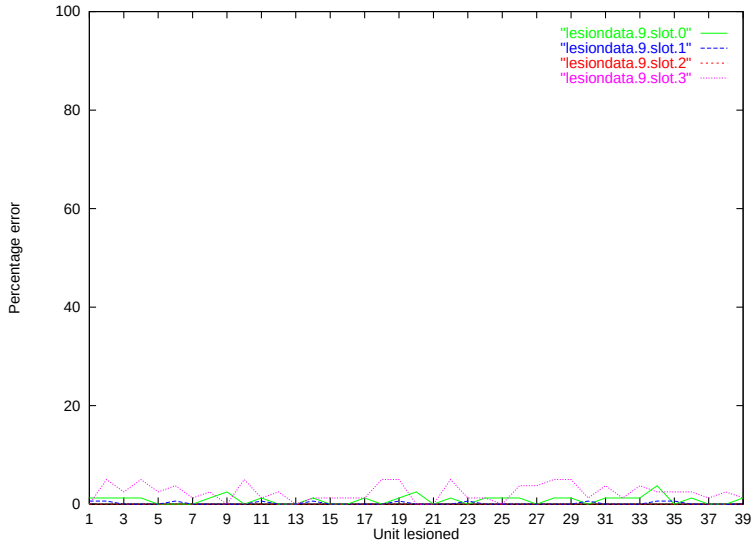

Figure B.34: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 80 sequences and the 2 unit representation, using 10% noise.

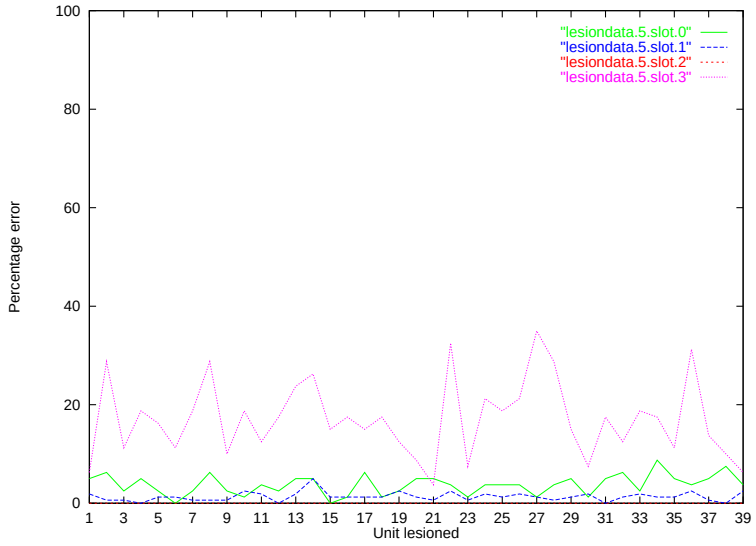

Figure B.35: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 80 sequences and the 2 unit representation, using 50% noise.

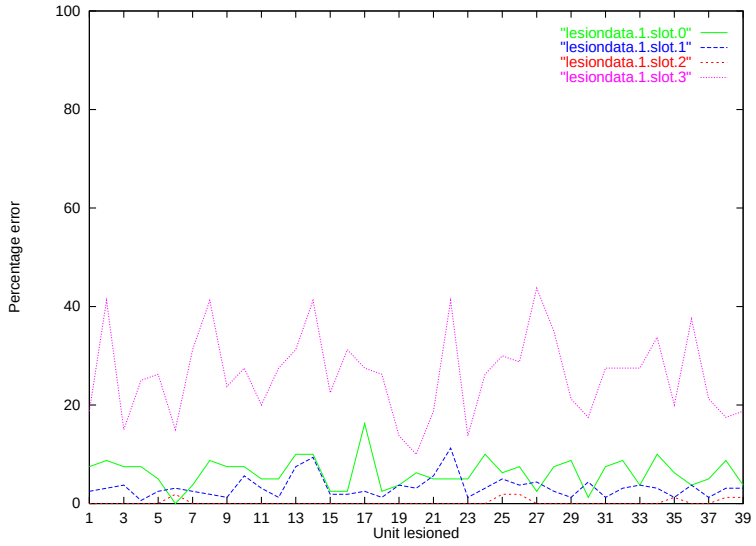

Figure B.36: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 80 sequences and the 2 unit representation, using 90% noise.

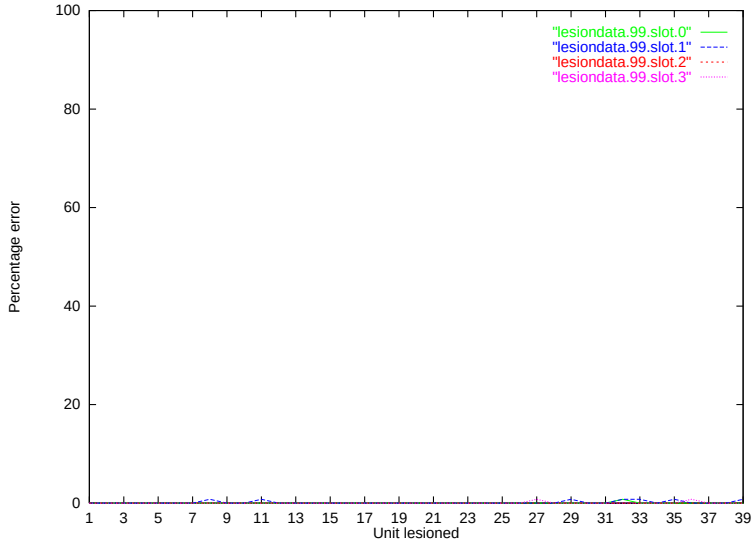

Figure B.37: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 130 sequences and the 2 unit representation, using 1% noise. Some error was produced at units 8, 11, 27, 29, 32, 33 and 35.

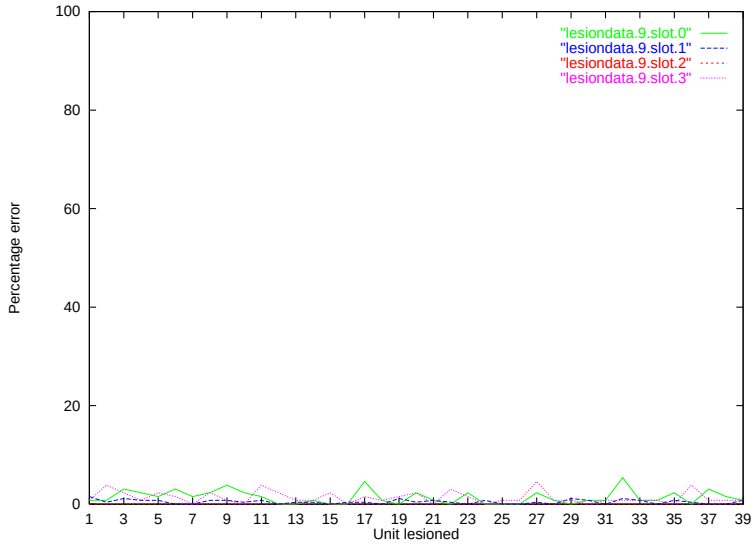

Figure B.38: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 130 sequences and the 2 unit representation, using 10% noise.

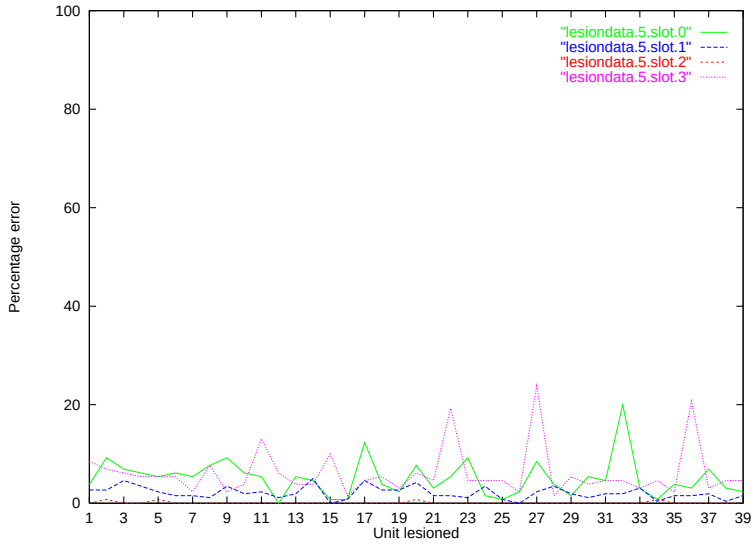

Figure B.39: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 130 sequences and the 2 unit representation, using 50% noise.

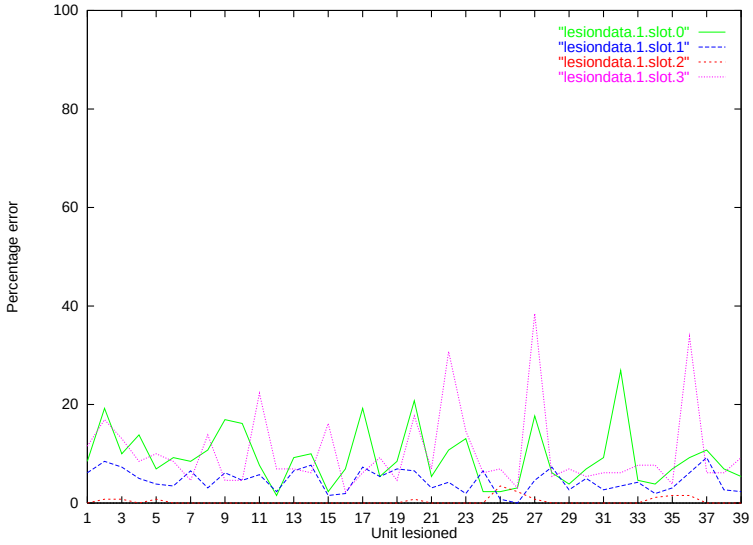

Figure B.40: The results of perturbing each unit in encodings developed by the SRAAM trained via back-propagation and hyperbolic tangent units on 130 sequences and the 2 unit representation, using 90% noise.

B.2 Networks trained by Kwasny and Kalman's method

Figures B.41 to B.48 present the results of the lesioning experiment for networks trained via Kwasny and Kalman's method, on the 1 unit representation and 40 and 80 sequences.

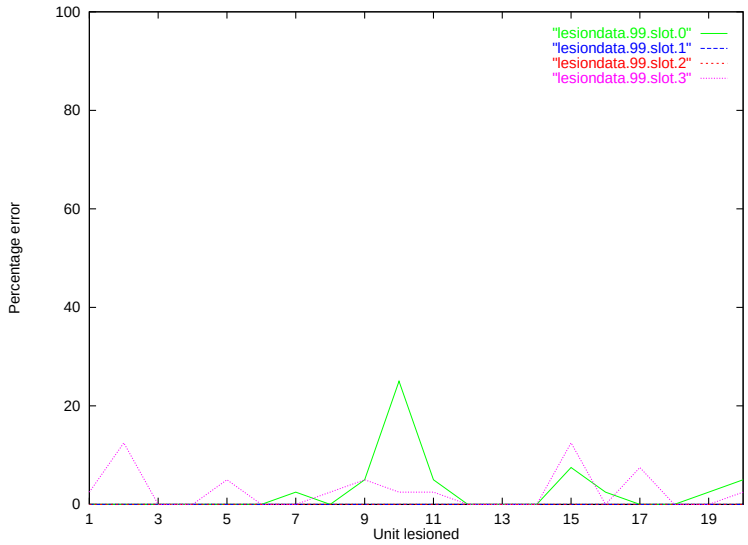

Figure B.41: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 40 sequences and the 1 unit representation, using 1% noise.

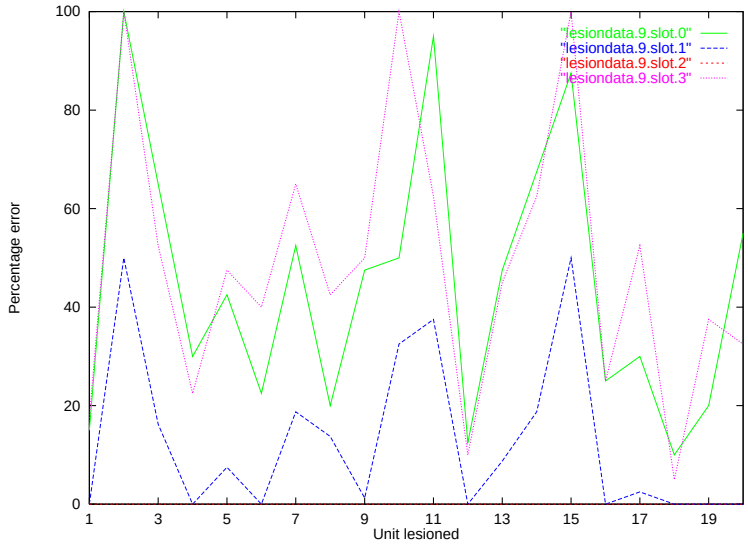

Figure B.42: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 40 sequences and the 1 unit representation, using 10% noise.

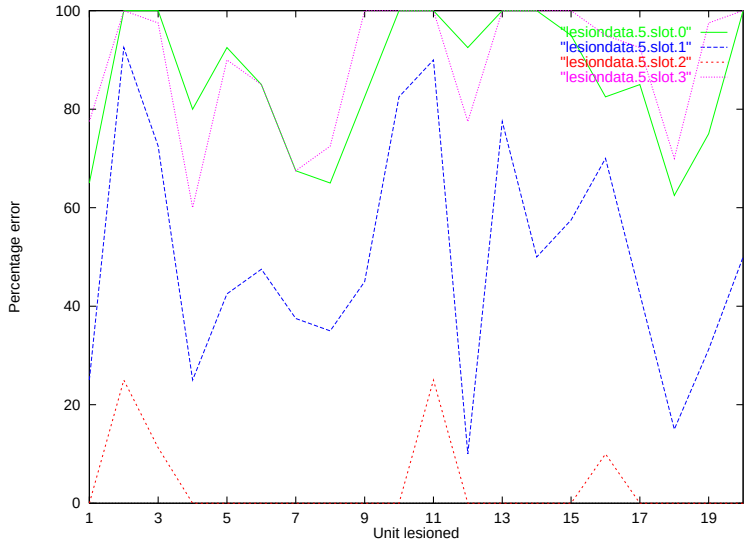

Figure B.43: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 40 sequences and the 1 unit representation, using 50% noise.

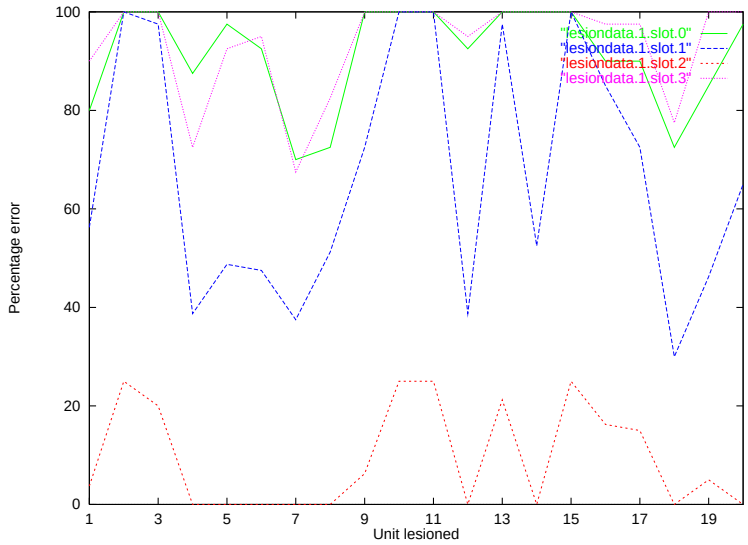

Figure B.44: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 40 sequences and the 1 unit representation, using 90% noise.

Figures B.49 to B.64 show the results of lesioning with the SRAAMs trained on the 2 bit representation with hyperbolic tangent units.

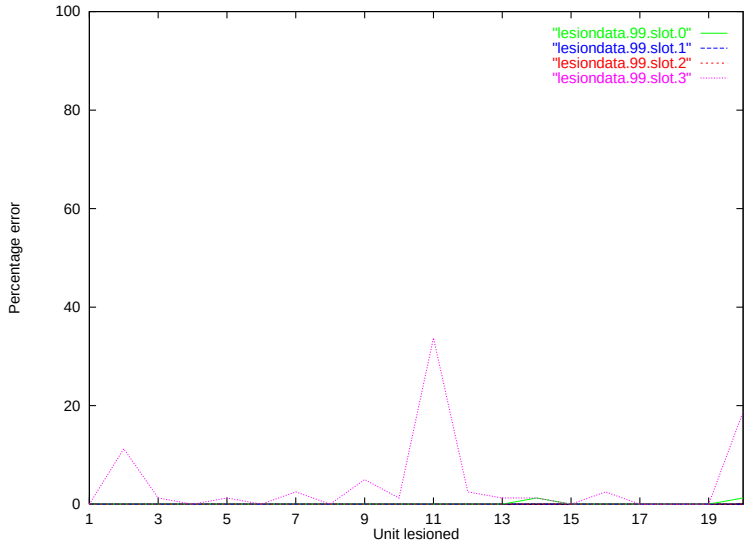

Figure B.45: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 80 sequences and the 1 unit representation, using 1% noise.

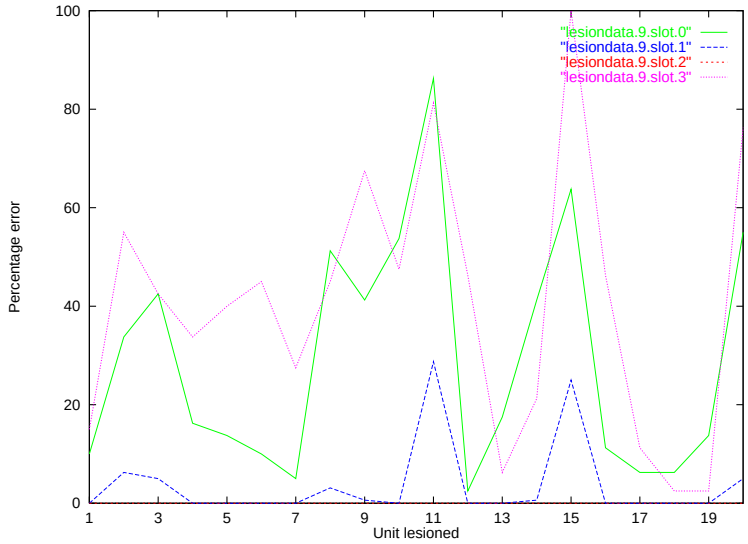

Figure B.46: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 80 sequences and the 1 unit representation, using 10% noise.

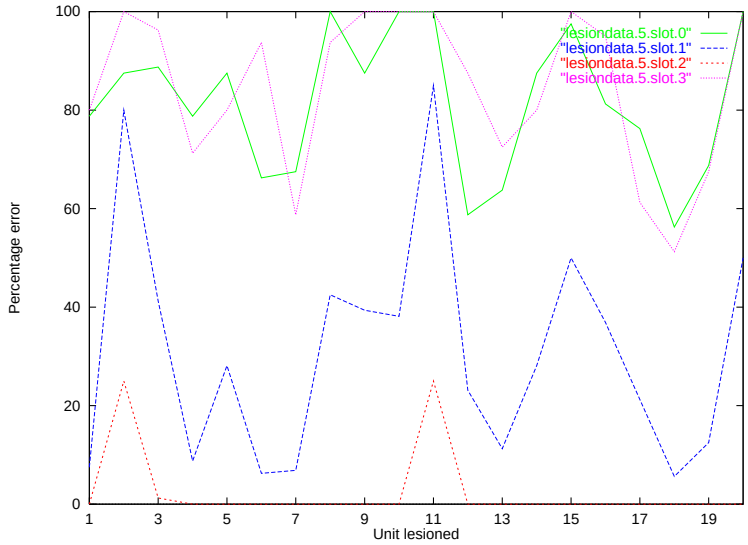

Figure B.47: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 80 sequences and the 1 unit representation, using 50% noise.

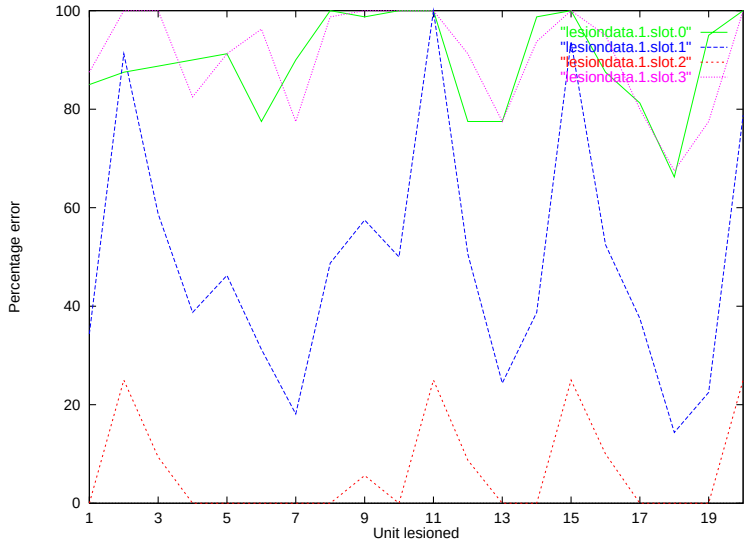

Figure B.48: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 80 sequences and the 1 unit representation, using 90% noise.

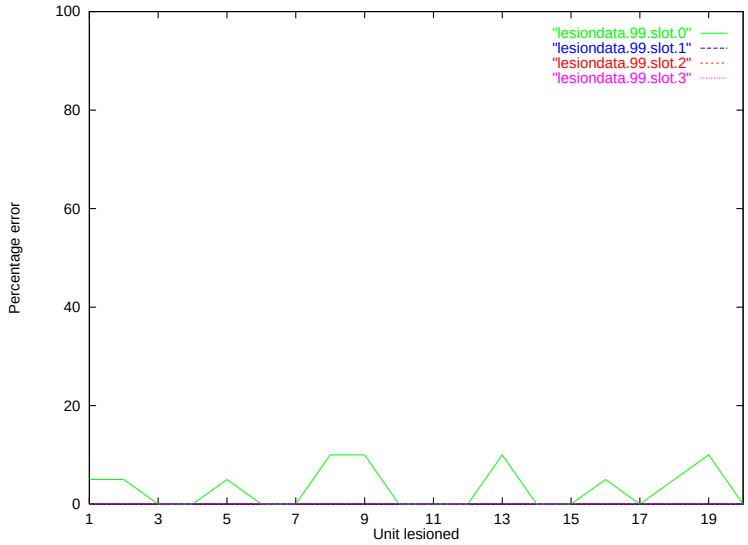

Figure B.49: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 20 sequences and the 2 unit representation, using 1% noise.

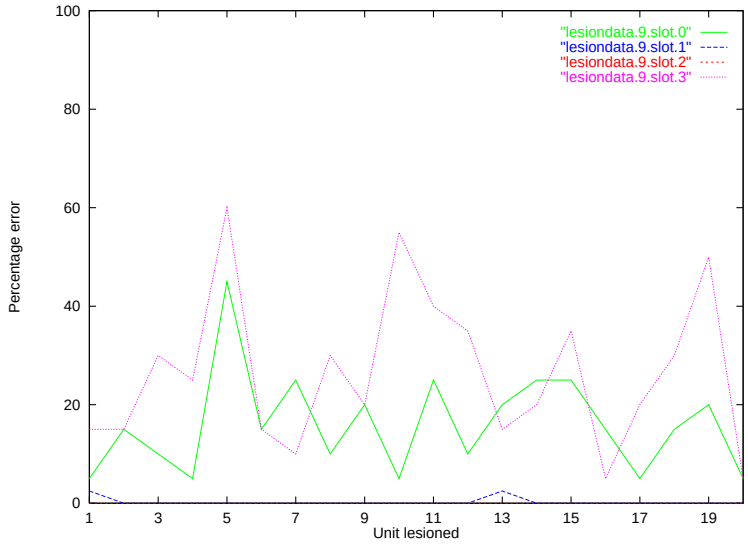

Figure B.50: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 20 sequences and the 2 unit representation, using 10% noise.

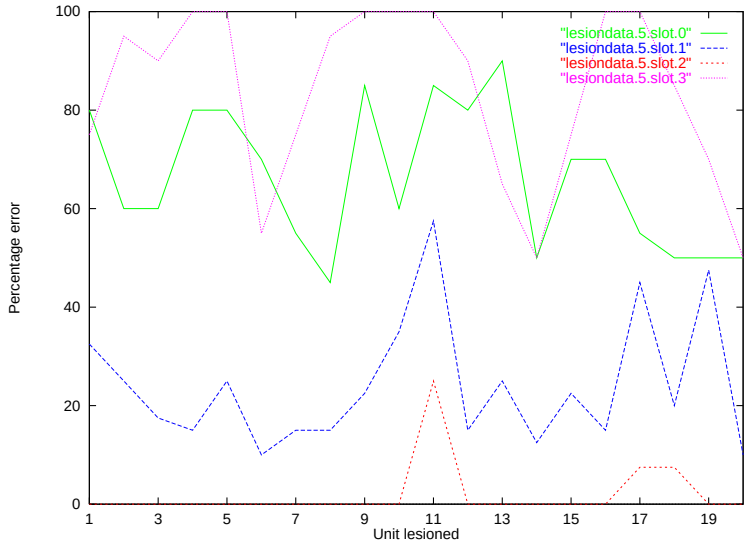

Figure B.51: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 20 sequences and the 2 unit representation, using 50% noise.

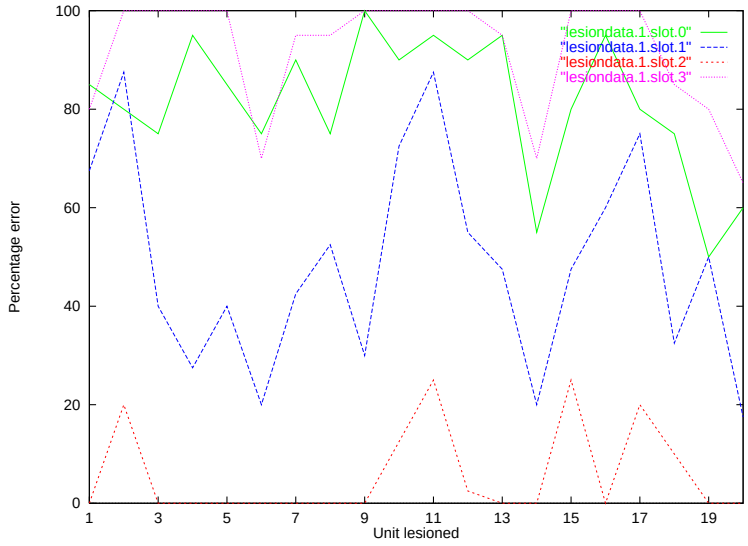

Figure B.52: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 20 sequences and the 2 unit representation, using 90% noise.

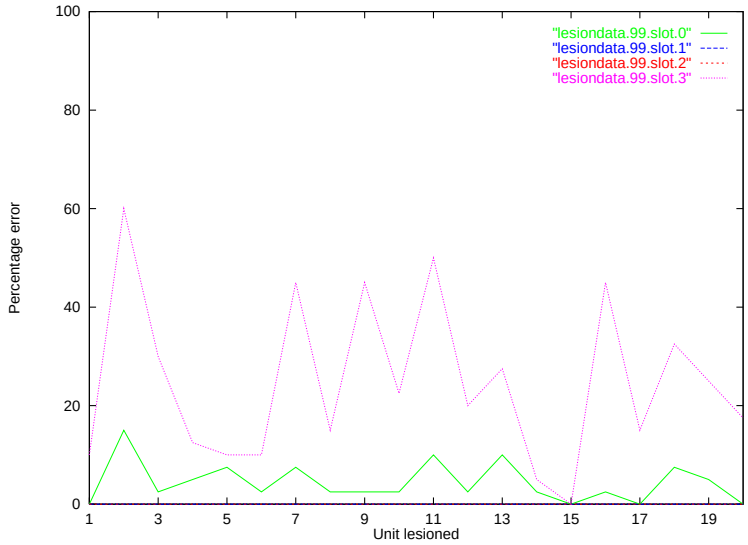

Figure B.53: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 40 sequences and the 2 unit representation, using 1% noise.

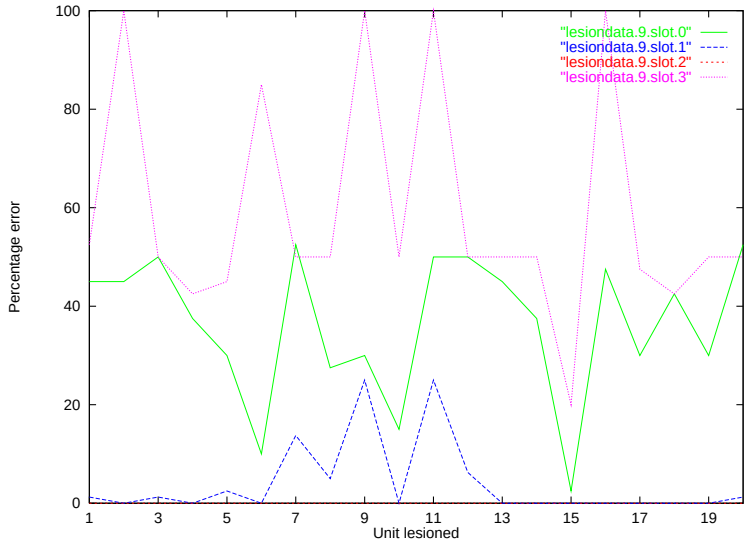

Figure B.54: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 40 sequences and the 2 unit representation, using 10% noise.

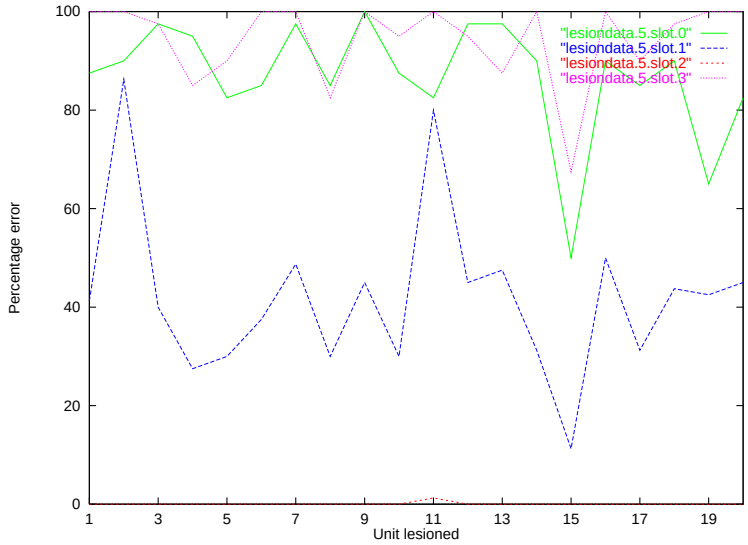

Figure B.55: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 40 sequences and the 2 unit representation, using 50% noise.

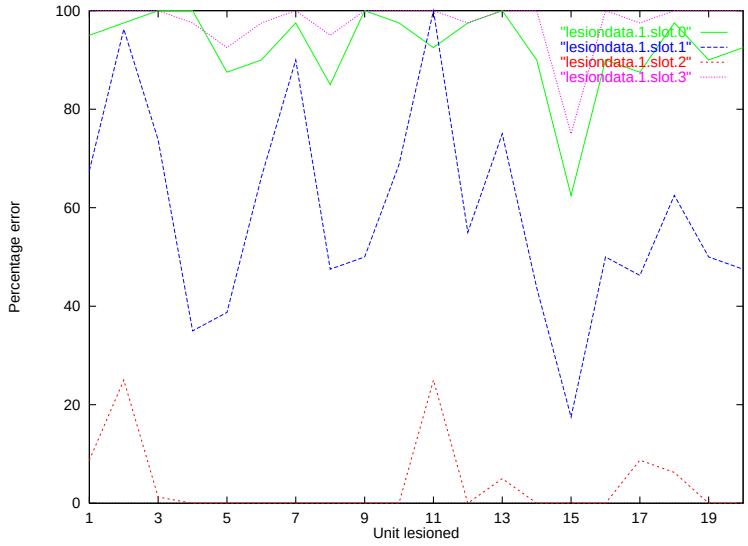

Figure B.56: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 40 sequences and the 2 unit representation, using 90% noise.

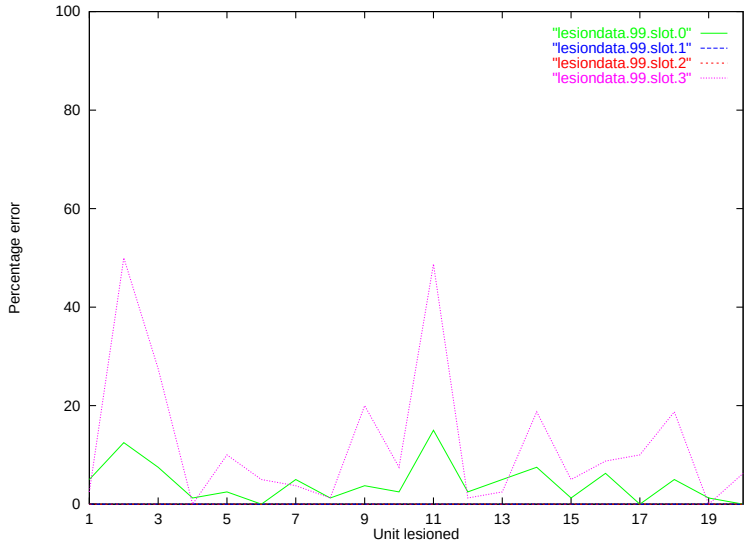

Figure B.57: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 80 sequences and the 2 unit representation, using 1% noise.

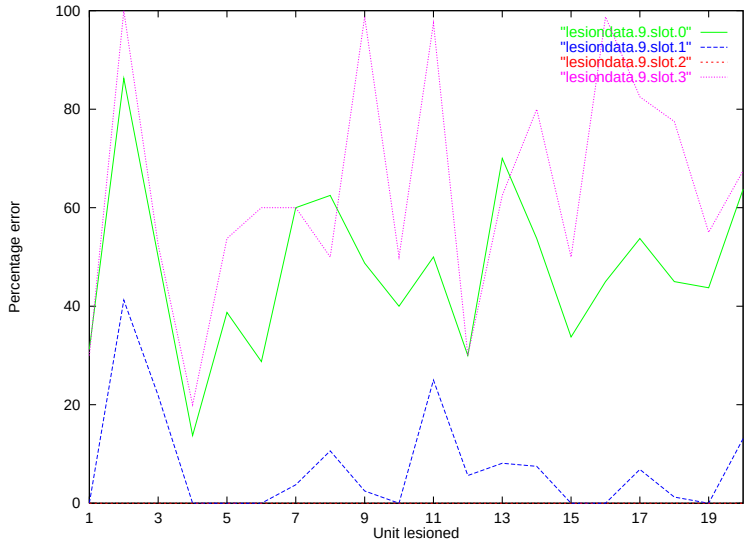

Figure B.58: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 80 sequences and the 2 unit representation, using 10% noise.

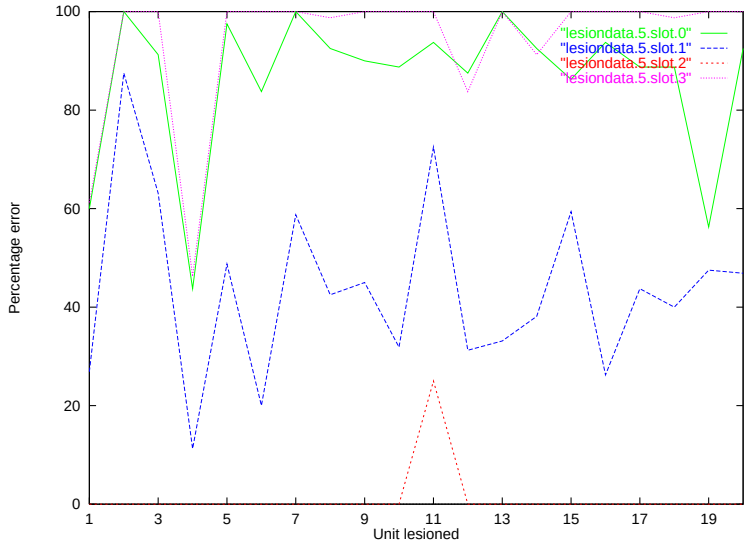

Figure B.59: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 80 sequences and the 2 unit representation, using 50% noise.

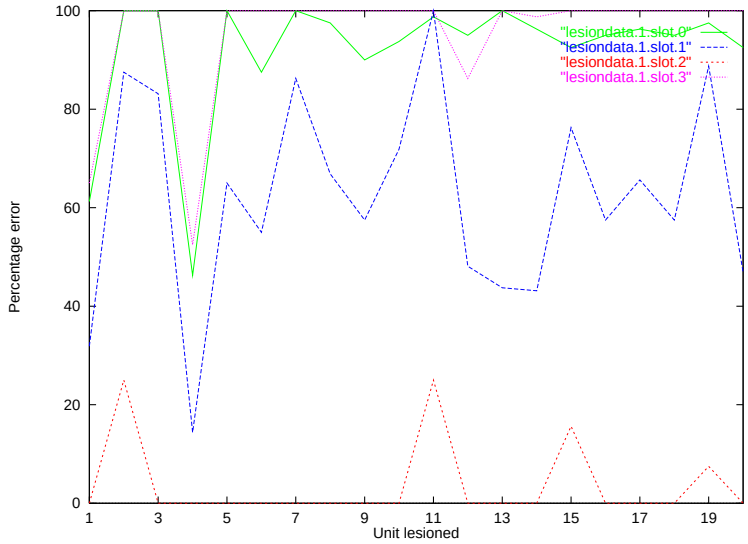

Figure B.60: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 80 sequences and the 2 unit representation, using 90% noise.

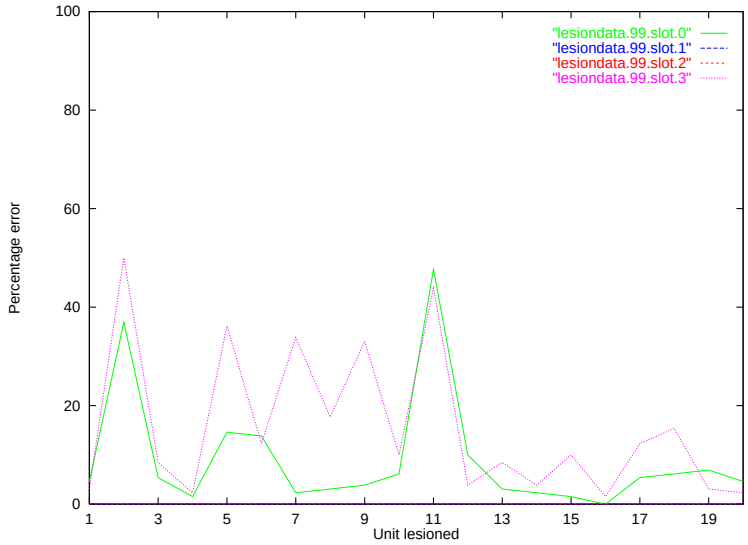

Figure B.61: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 130 sequences and the 2 unit representation, using 1% noise.

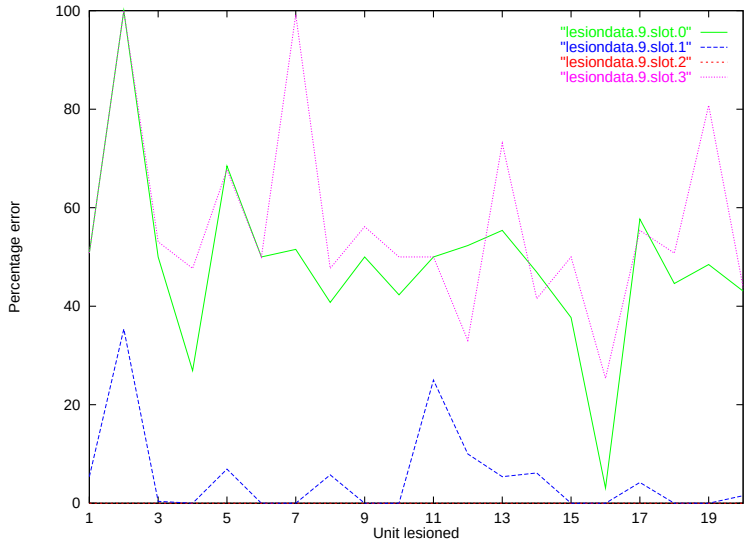

Figure B.62: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 130 sequences and the 2 unit representation, using 10% noise.

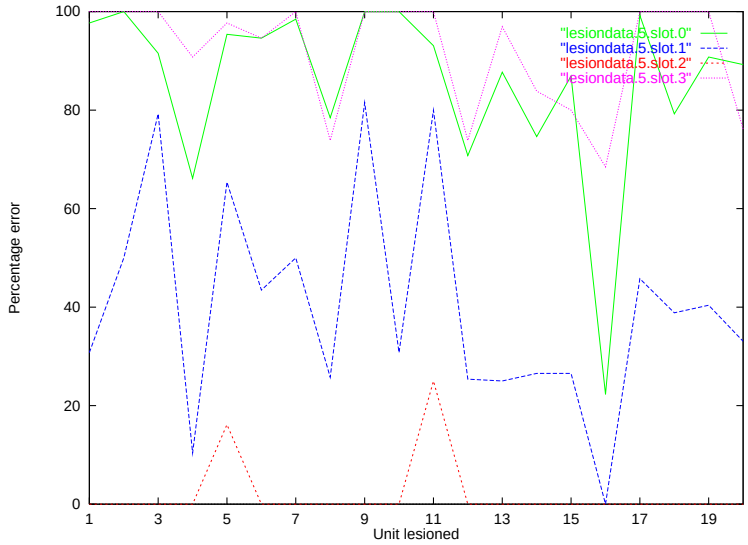

Figure B.63: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 130 sequences and the 2 unit representation, using 50% noise.

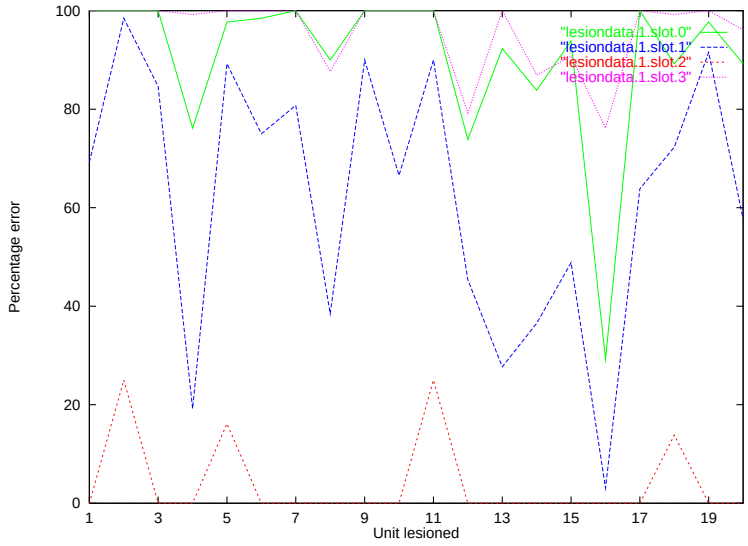

Figure B.64: The results of perturbing each unit in encodings developed by the SRAAM trained via Kwasny and Kalman's method on 130 sequences and the 2 unit representation, using 90% noise.

Appendix C

Plots of encoding and decoding trajectories

C.1 Back-propagation networks

Figures C.1 to C.6 show the encoding and decoding trajectories for the SRAAMs with sigmoidal units trained on the 2 unit representation.

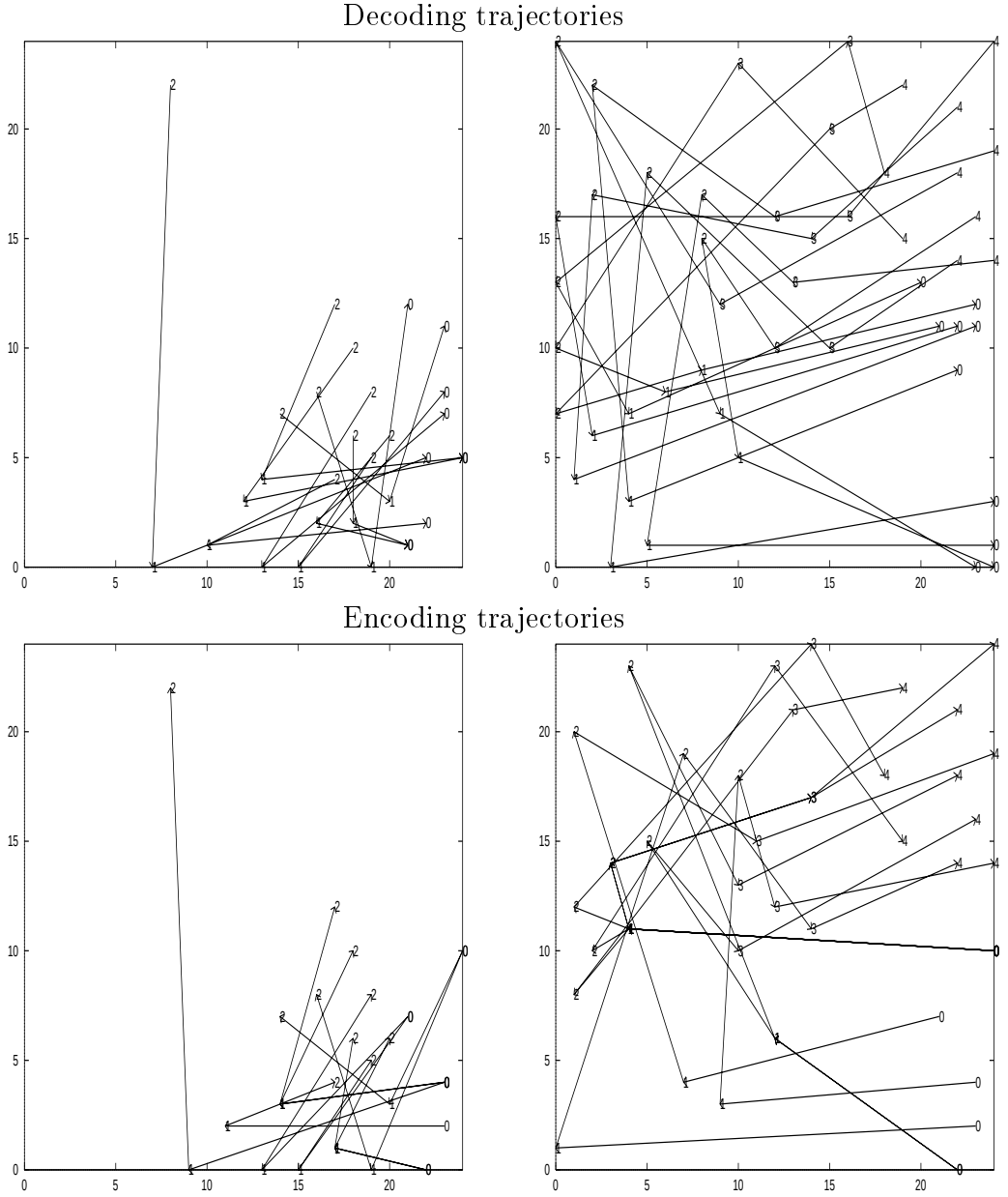

Figure C.1: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation, back-propagation and sigmoidal units after 1000 iterations. Active sentences on the left, passive on the right.

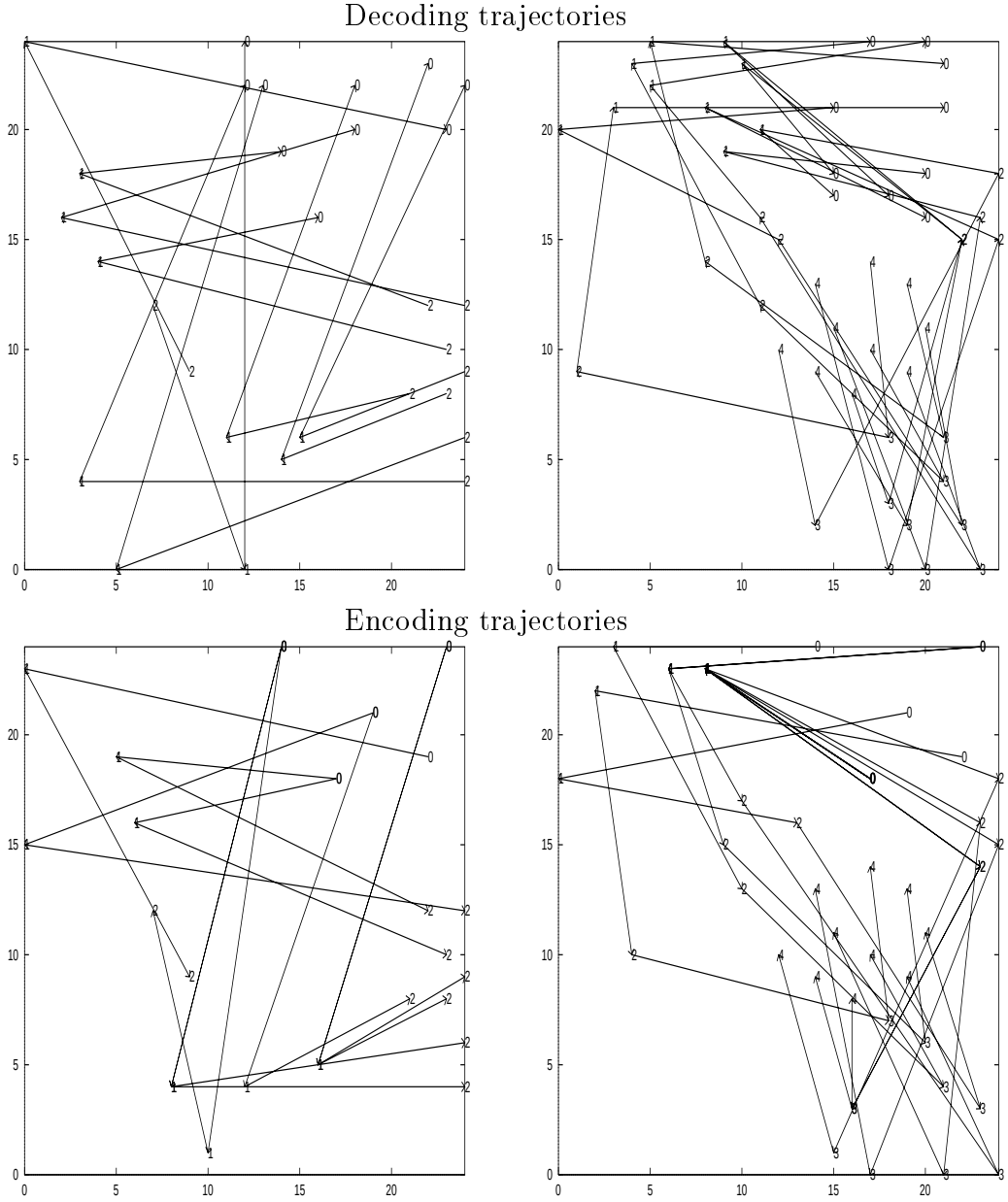

Figure C.2: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation, back-propagation and sigmoidal units after 2000 iterations. Active sentences on the left, passive on the right.

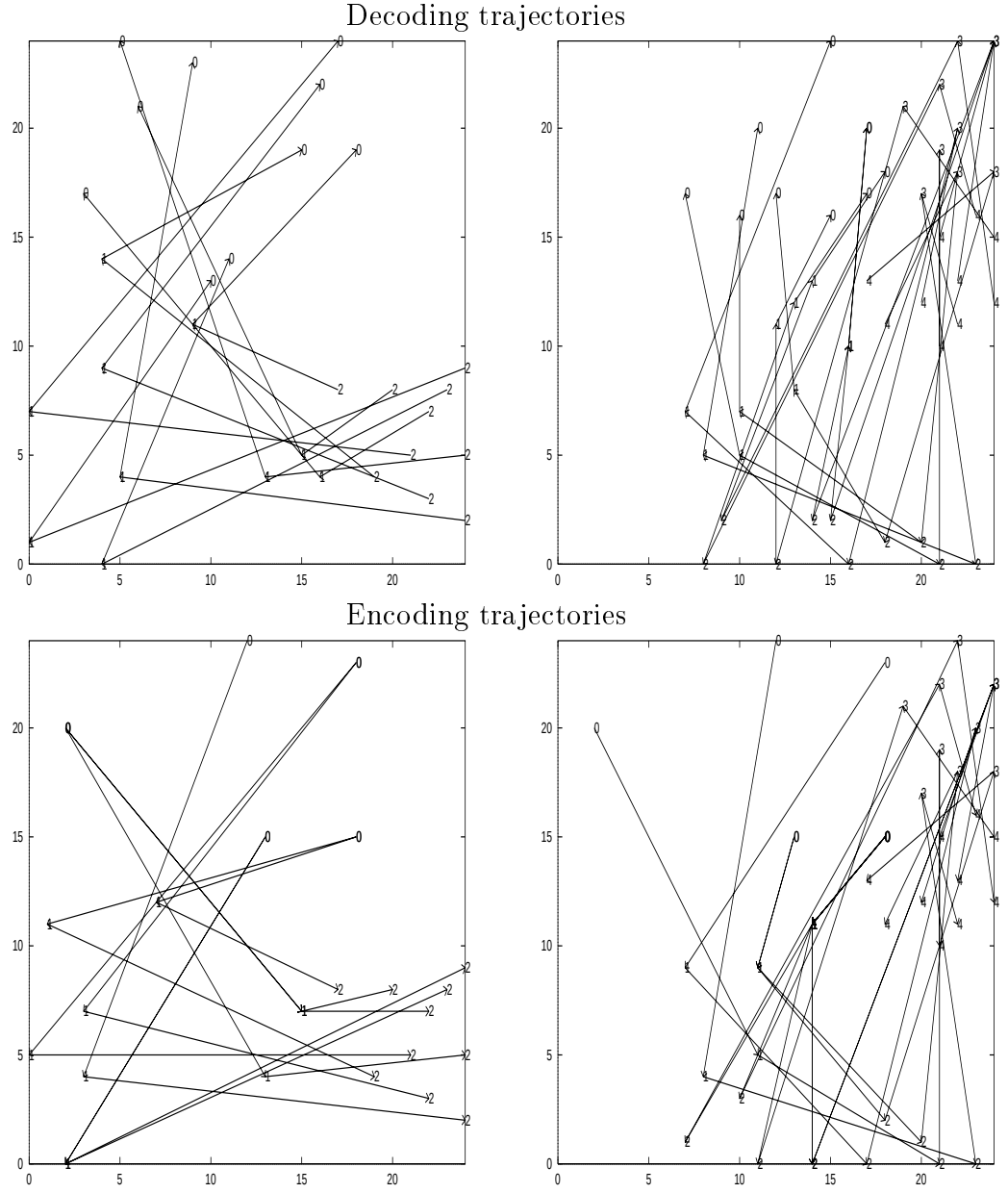

Figure C.3: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation, back-propagation and sigmoidal units after 3200 iterations (training complete). Active sentences on the left, passive on the right.

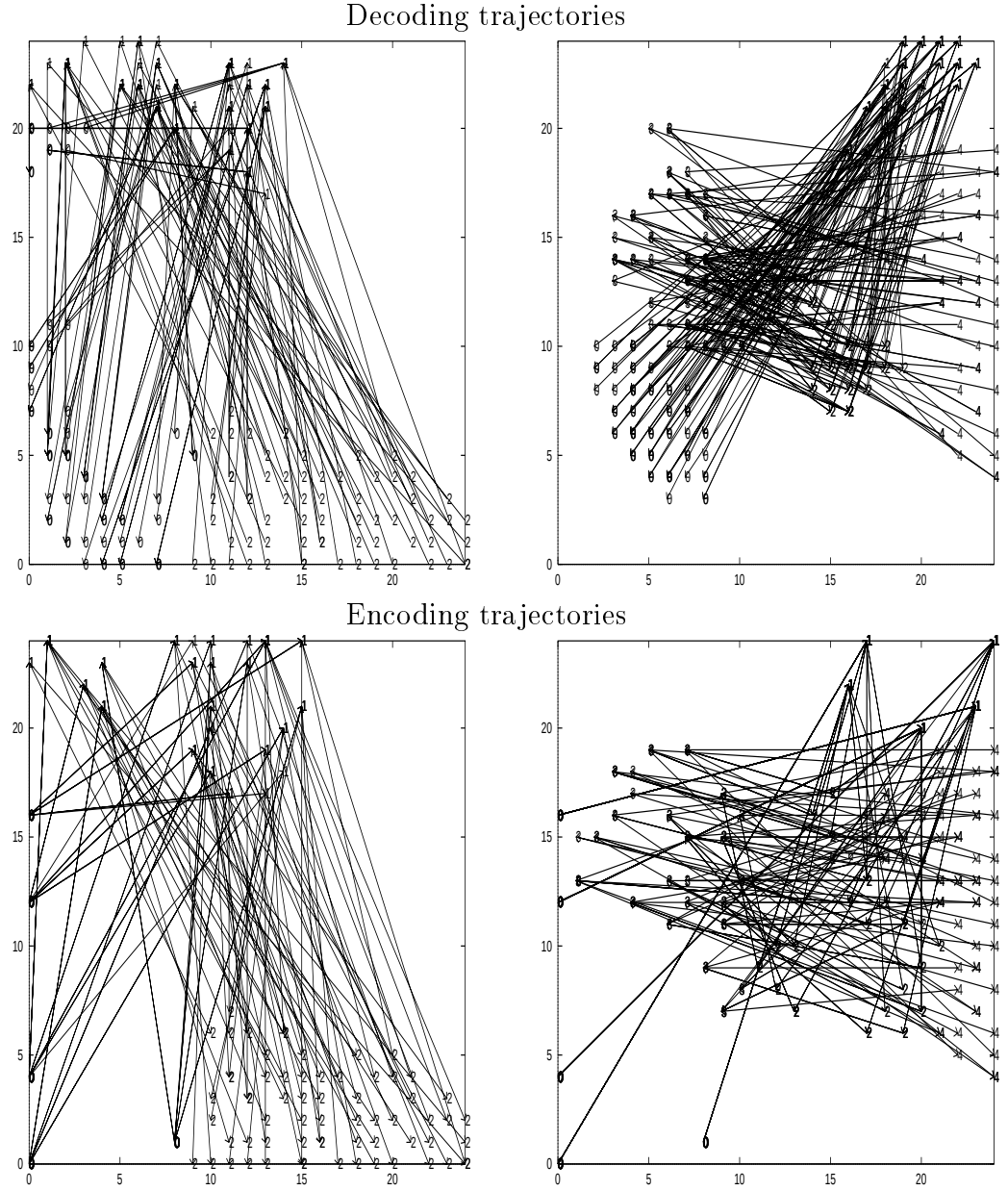

Figure C.4: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation, back-propagation and sigmoidal units after 500 iterations. Active sentences on the left, passive on the right.

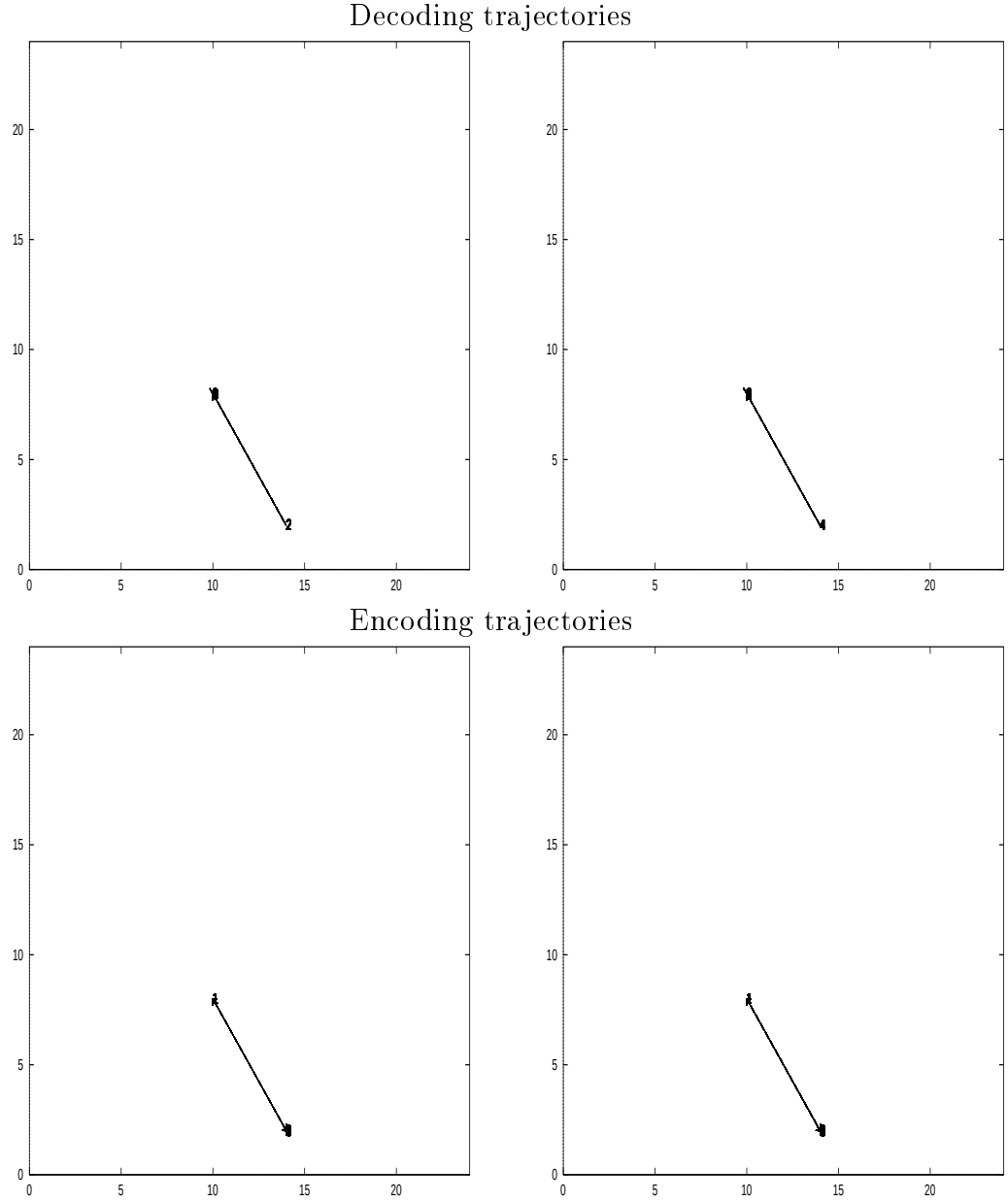

Figure C.5: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation, back-propagation and sigmoidal units after 1000 iterations. Active sentences on the left, passive on the right.

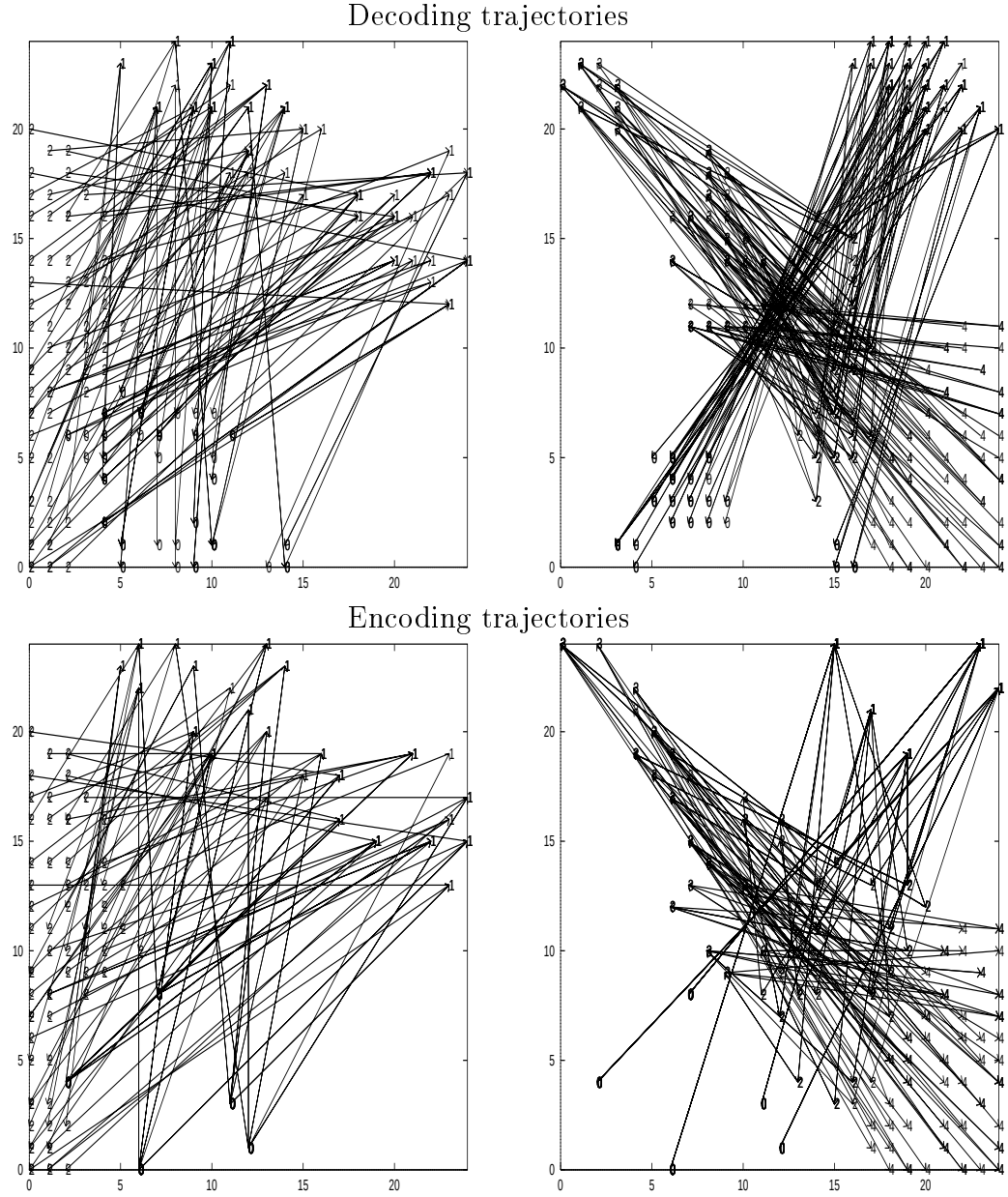

Figure C.6: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation, back-propagation and sigmoidal units after 1514 iterations. Active sentences on the left, passive on the right.

C.2 Networks trained via Kwasny and Kalman's method

Figures C.7 to C.12 show the encoding and decoding trajectories for the SRAAMs trained with Kwasny and Kalman's method on the 2 unit representation.

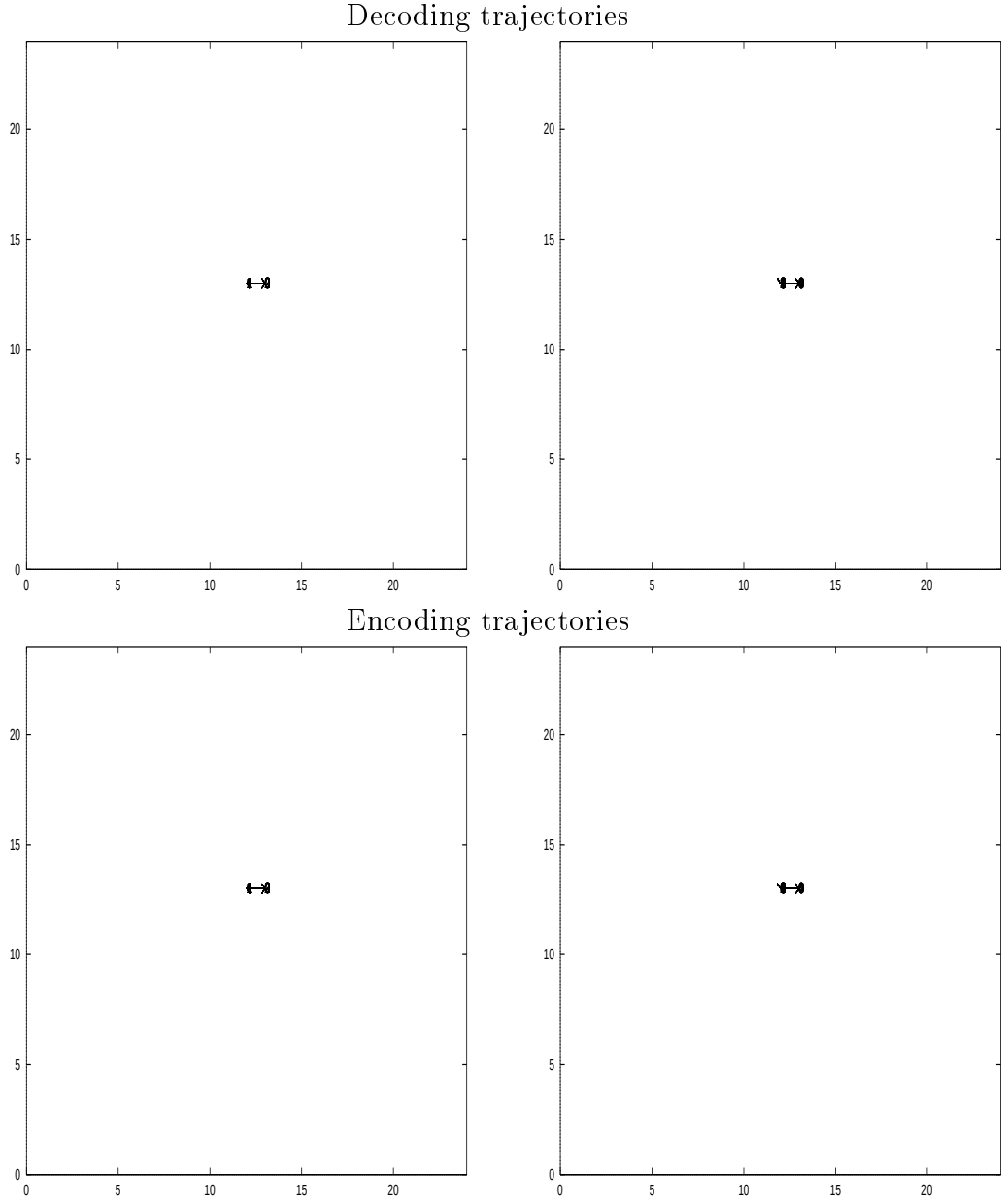

Figure C.7: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation and Kwasny and Kalman's training after training to a tolerance of 0.16. Active sentences on the left, passive on the right.

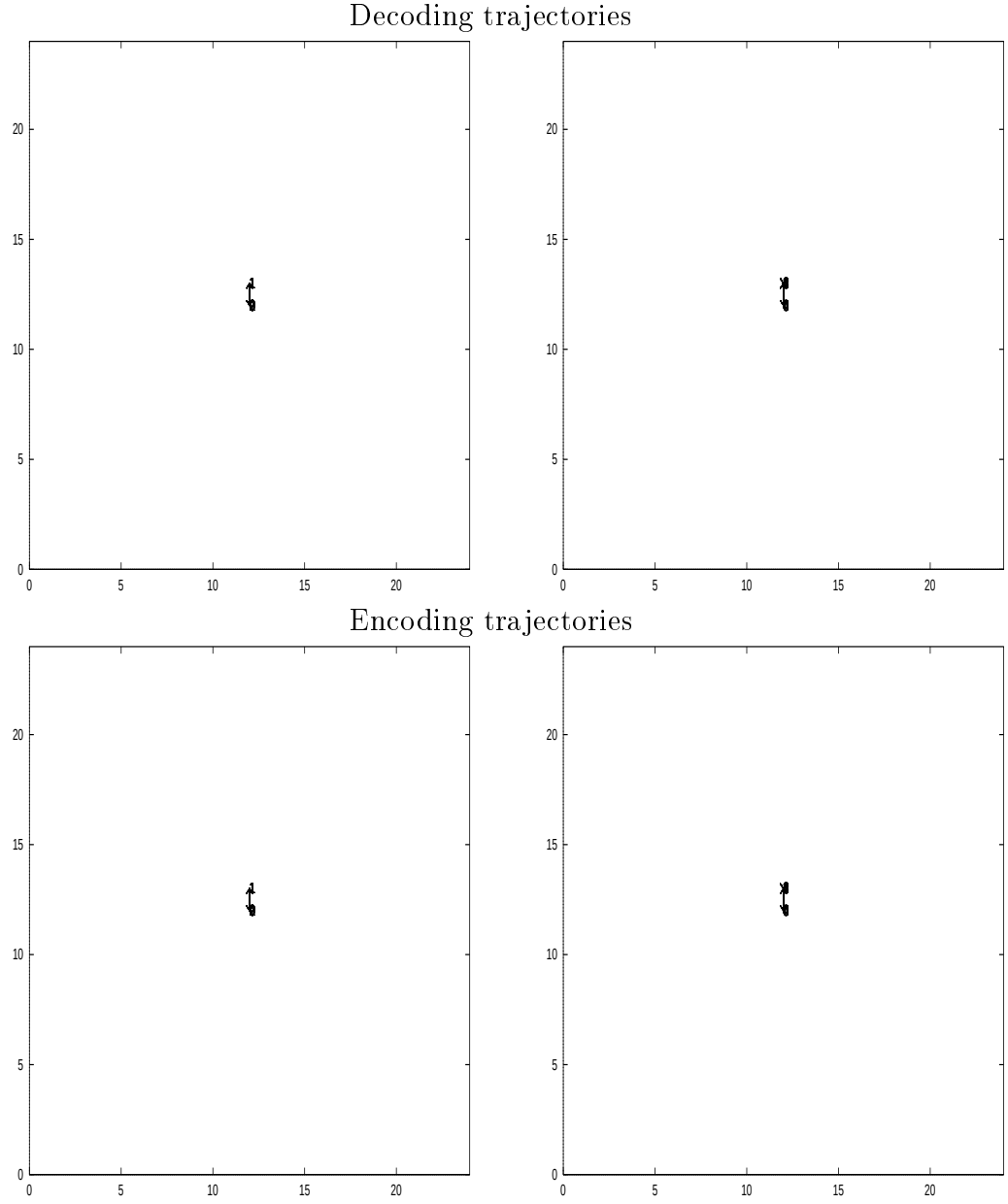

Figure C.8: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation and Kwasny and Kalman's training after training to a tolerance of 0.04. Active sentences on the left, passive on the right.

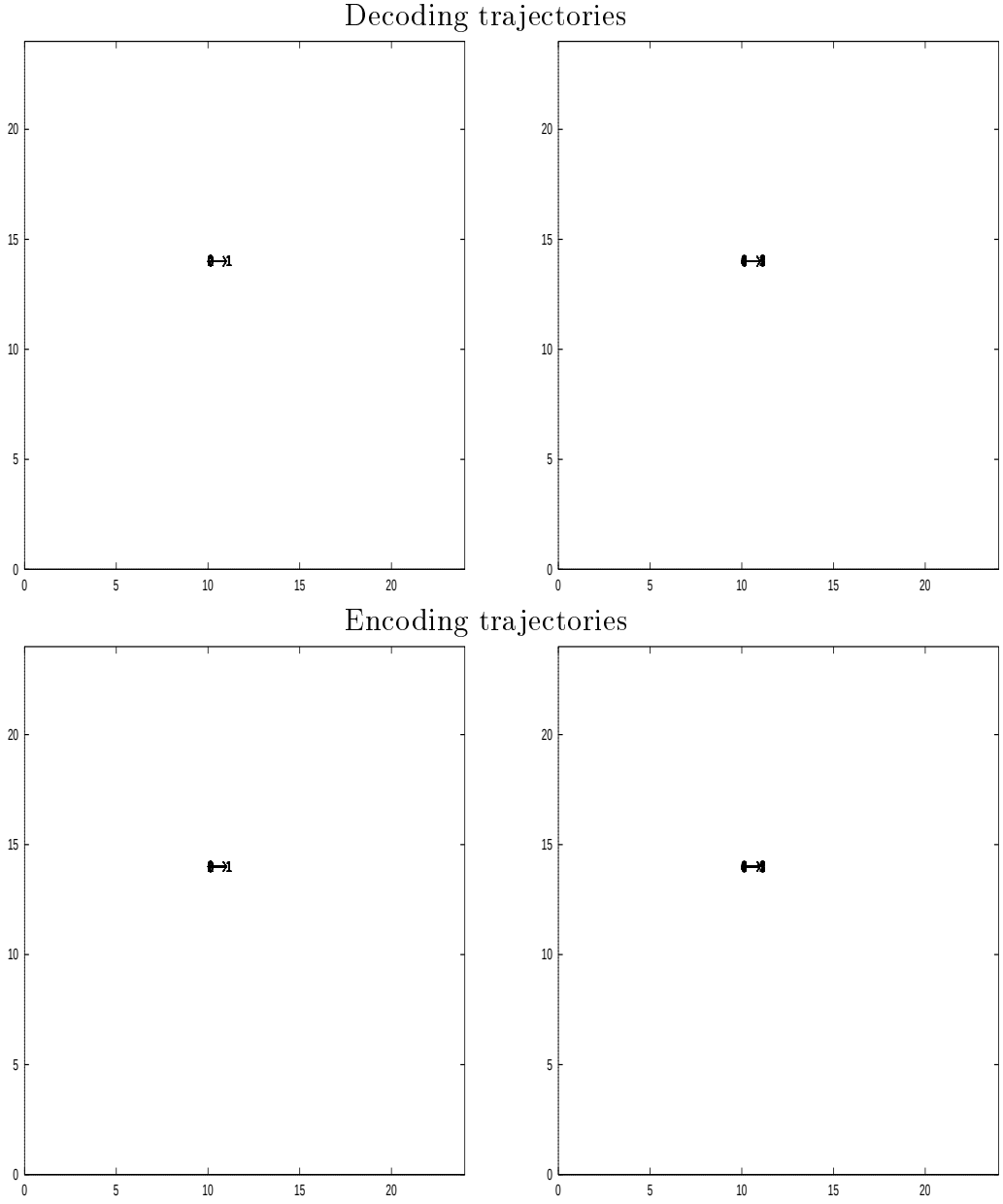

Figure C.9: Encoding and decoding trajectories for the SRAAM trained on 20 sequences with the 2 unit representation and Kwasny and Kalman's training after training has completed. Active sentences on the left, passive on the right.

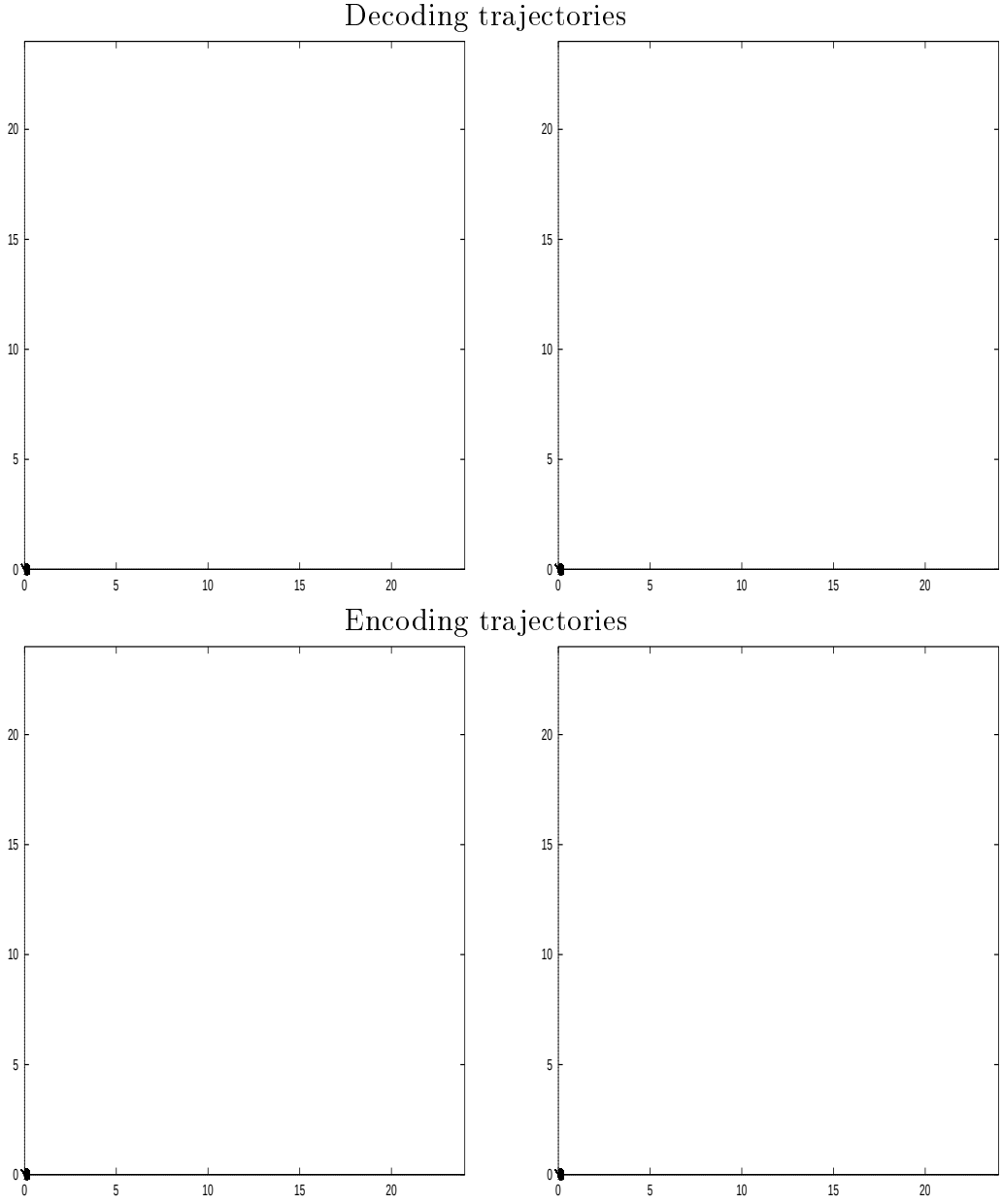

Figure C.10: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation and Kwasny and Kalman's training after training to a tolerance of 0.16. Active sentences on the left, passive on the right.

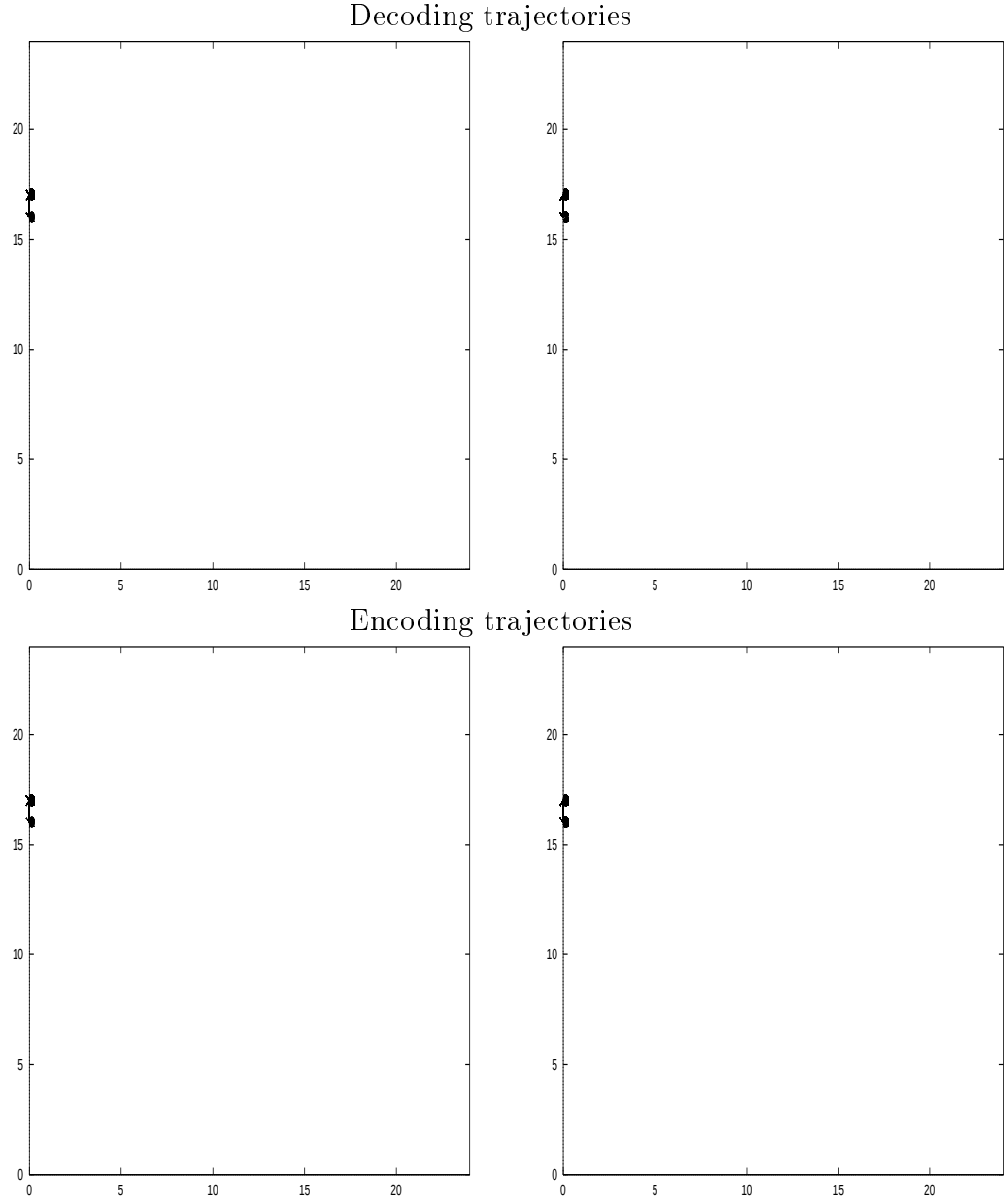

Figure C.11: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation and Kwasny and Kalman's training after training to a tolerance of 0.04. Active sentences on the left, passive on the right.

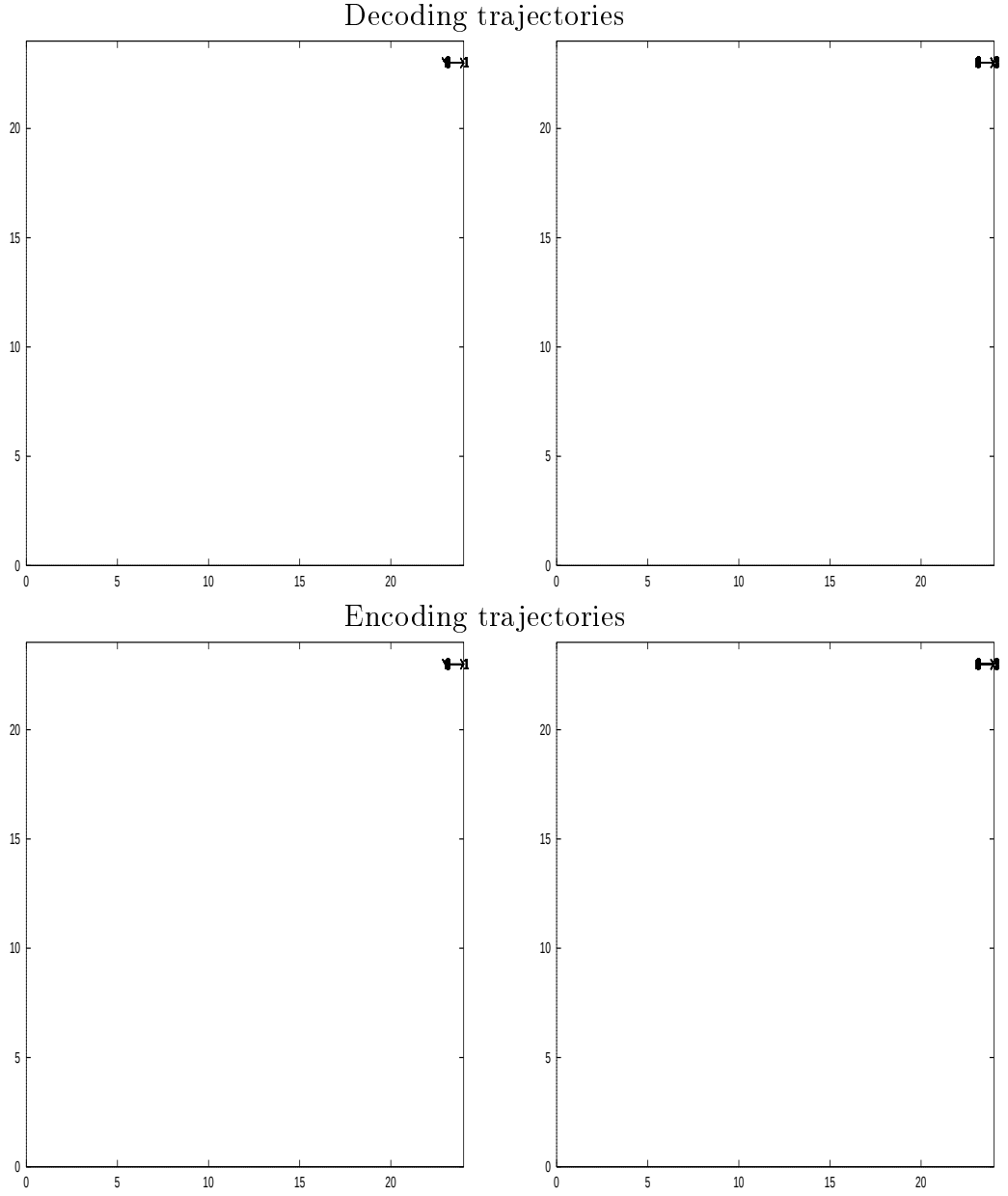

Figure C.12: Encoding and decoding trajectories for the SRAAM trained on 130 sequences with the 2 unit representation and Kwasny and Kalman's training after training has completed. Active sentences on the left, passive on the right.

Appendix D

Hyperspace Utilisation Plots

D.1 Back-propagation networks

Figures D.1 to D.2 show the range of activations generated during encoding and decoding for networks trained with sigmoidal units on the 2 unit representation. Figures D.3 to D.4 show the range of activations for the networks trained with hyperbolic tangent units and the 2 unit representation.

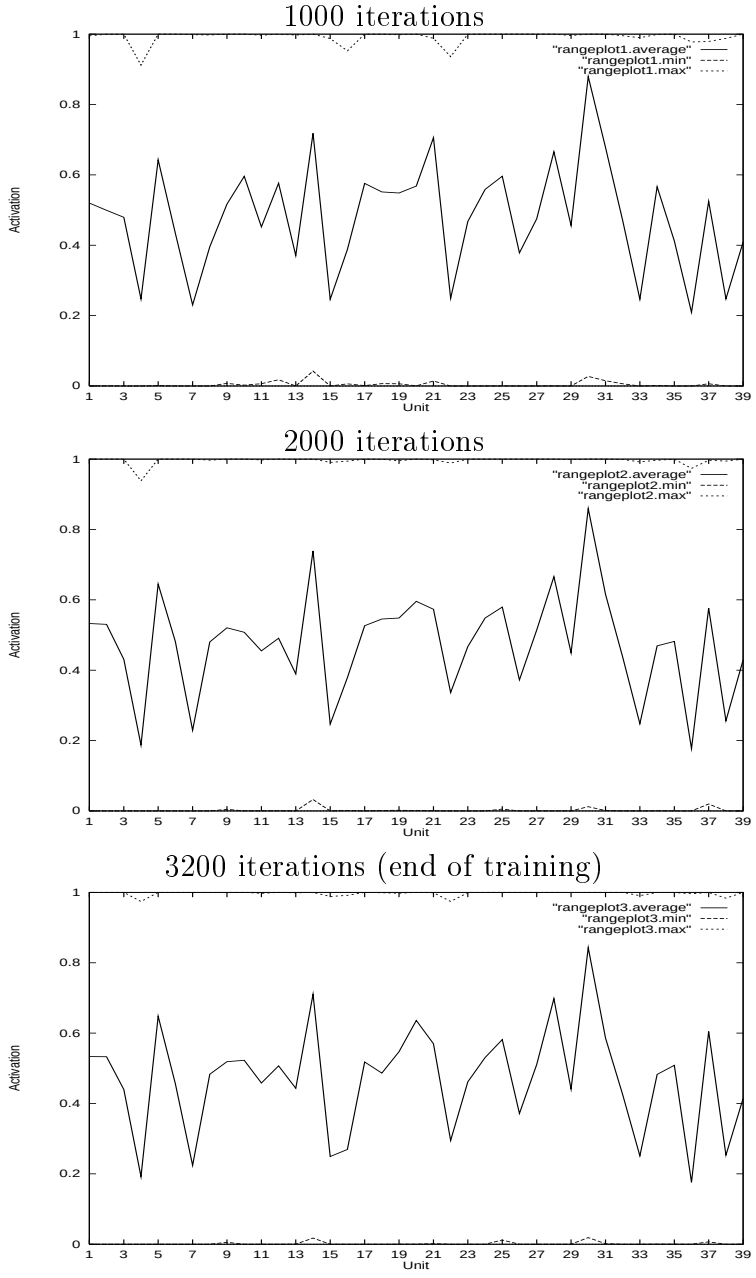

Figure D.1: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 20 sequences with sigmoidal units and the 2 unit representation using back-propagation.

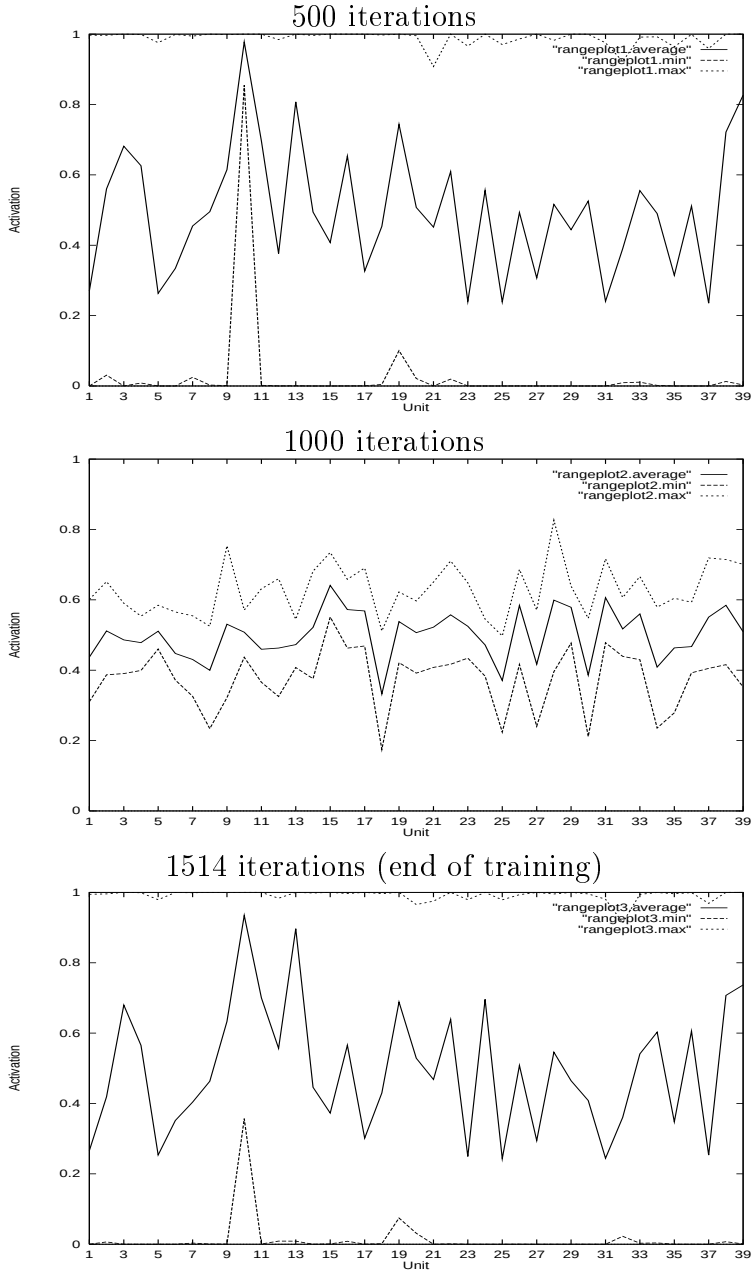

Figure D.2: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 130 sequences with sigmoidal units and the 2 unit representation using back-propagation.

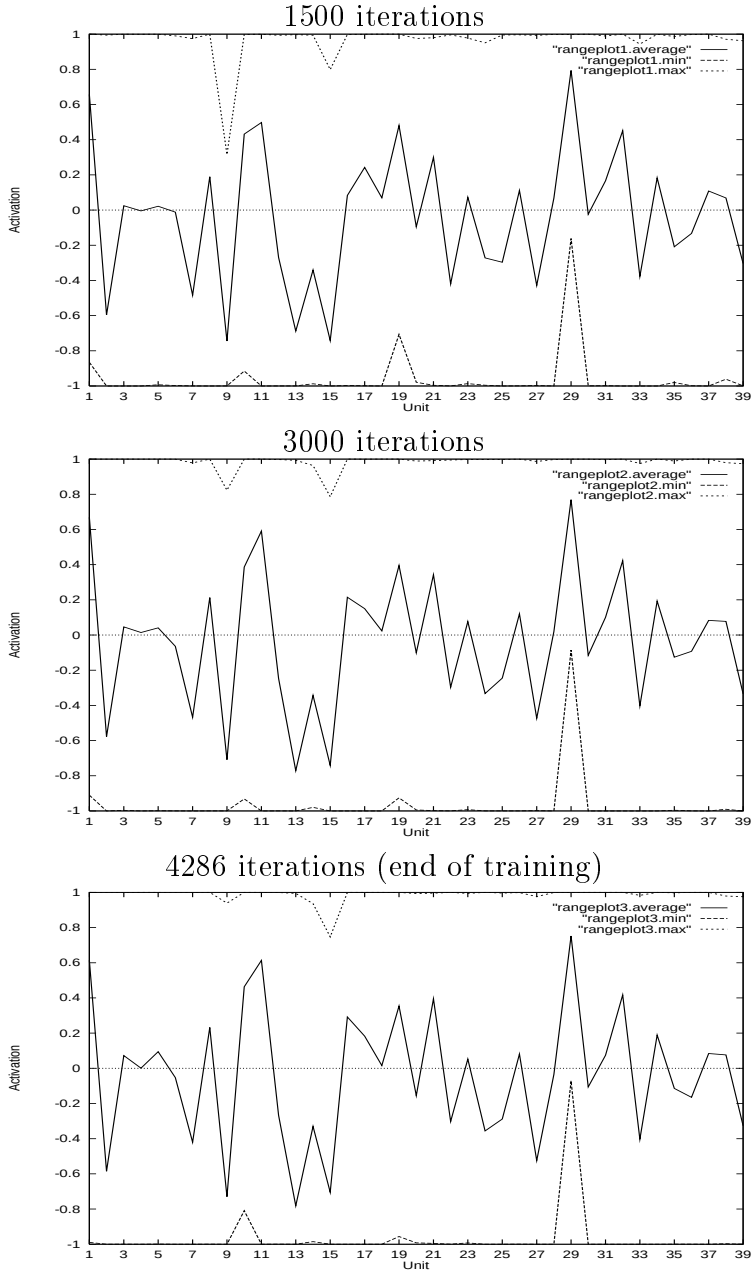

Figure D.3: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 20 sequences with hyperbolic tangent units and the 2 unit representation using back-propagation.

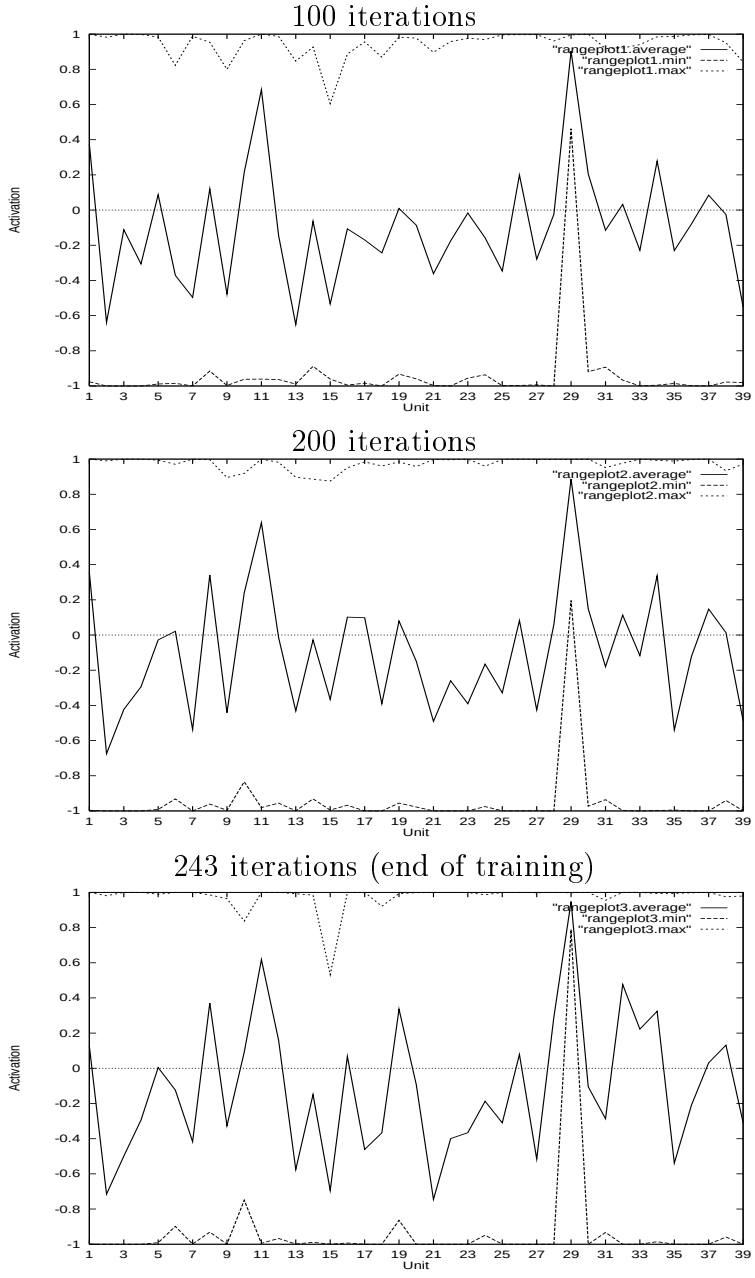

Figure D.4: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 130 sequences with hyperbolic tangent units and the 2 unit representation using back-propagation.

D.2 Kwasny-Kalman networks

Figures D.5 to D.6 present the range of activations for the networks trained via Kwasny and Kalman's method on the 2 unit representation.

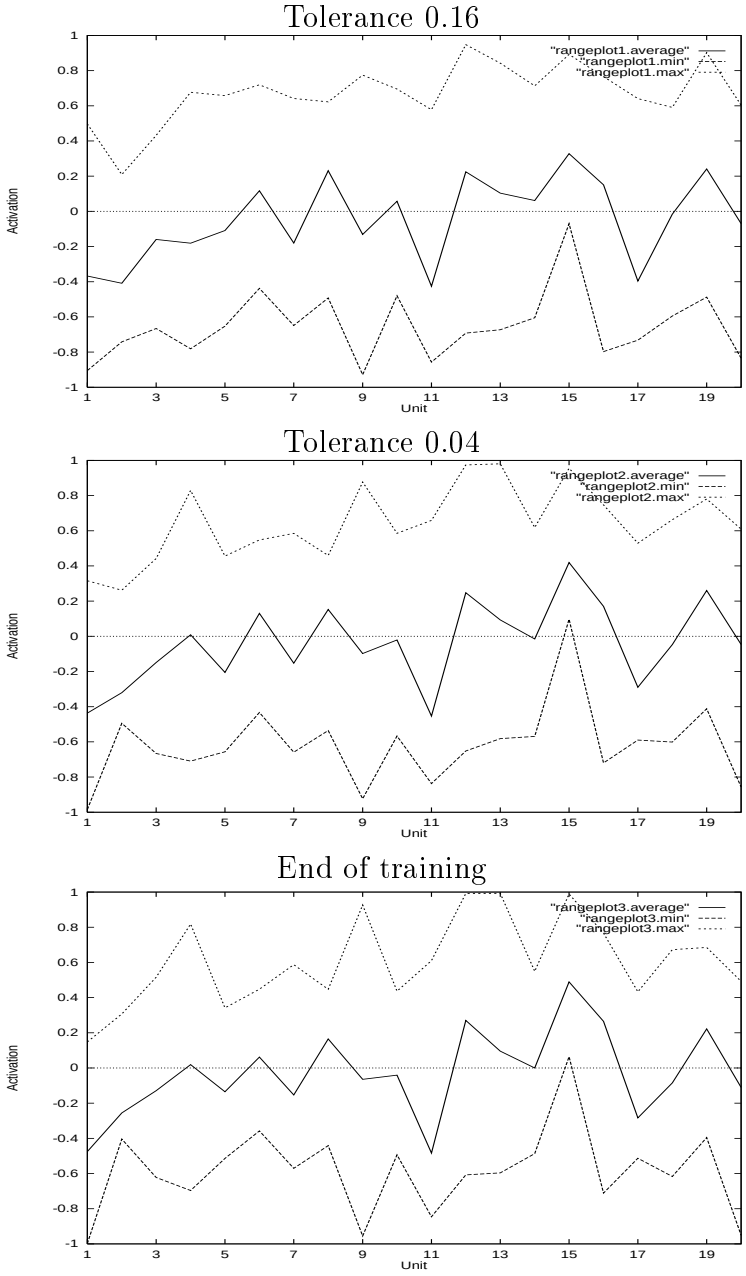

Figure D.5: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 20 sequences with the 2 unit representation using Kwasny and Kalman's training method.

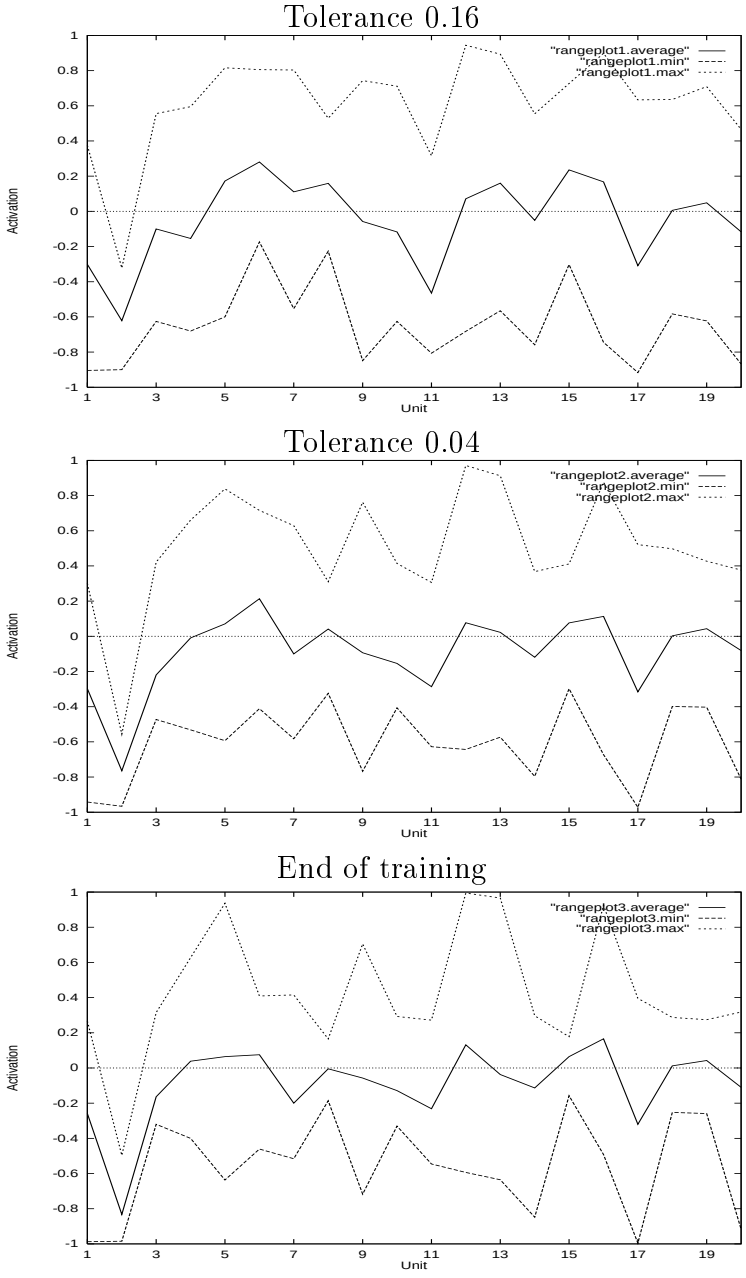

Figure D.6: The range of hidden-layer activations generated during encoding and decoding, for the SRAAM trained on 130 sequences with the 2 unit representation using Kwasny and Kalman's training method.

Appendix E

Training and testing sentences

E.1 Training sentences

```
michael hug michael
michael is hug by michael
helen hit chris
helen is hit by chris
john betray john
john is betray by john
michael kill john
michael is kill by john
chris love john
chris is love by john
helen kill helen
helen is kill by helen
michael hit michael
michael is hit by michael
diane hit john
diane is hit by john
helen hug chris
helen is hug by chris
diane hug john
diane is hug by john
helen betray diane
helen is betray by diane
```

```
diane kill michael
diane is kill by michael
chris hit john
chris is hit by john
michael hit diane
michael is hit by diane
diane kill helen
diane is kill by helen
helen kill michael
helen is kill by michael
chris hit chris
chris is hit by chris
john betray michael
john is betray by michael
chris love michael
chris is love by michael
michael hug helen
michael is hug by helen
diane hit chris
diane is hit by chris
diane betray chris
diane is betray by chris
diane hit michael
diane is hit by michael
michael hug chris
michael is hug by chris
john hug helen
john is hug by helen
helen love chris
helen is love by chris
chris kill michael
chris is kill by michael
chris betray john
chris is betray by john
chris love chris
chris is love by chris
michael hit chris
michael is hit by chris
```

```
chris hug helen
chris is hug by helen
michael betray michael
michael is betray by michael
michael hit john
michael is hit by john
john love michael
john is love by michael
helen love michael
helen is love by michael
helen hit michael
helen is hit by michael
diane love helen
diane is love by helen
john hug chris
john is hug by chris
helen hug diane
helen is hug by diane
chris betray michael
chris is betray by michael
chris kill chris
chris is kill by chris
diane kill diane
diane is kill by diane
diane love michael
diane is love by michael
diane love chris
diane is love by chris
john kill michael
john is kill by michael
chris hit michael
chris is hit by michael
michael kill michael
michael is kill by michael
john hit helen
john is hit by helen
diane hug chris
diane is hug by chris
```

```
michael kill chris
michael is kill by chris
chris hit diane
chris is hit by diane
john hug john
john is hug by john
helen hit helen
helen is hit by helen
diane hug helen
diane is hug by helen
michael love john
michael is love by john
michael hit helen
michael is hit by helen
michael betray chris
michael is betray by chris
john hit michael
john is hit by michael
chris hug michael
chris is hug by michael
helen betray chris
helen is betray by chris
michael betray john
michael is betray by john
john kill john
john is kill by john
diane kill chris
diane is kill by chris
helen love diane
helen is love by diane
john betray helen
john is betray by helen
```

E.2 Testing sentences

```
chris kill john
chris is kill by john
```

diane love john
diane is love by john
michael love helen
michael is love by helen
diane betray helen
diane is betray by helen
john hit chris
john is hit by chris
helen love helen
helen is love by helen
michael kill diane
michael is kill by diane
helen hug helen
helen is hug by helen
michael betray helen
michael is betray by helen
chris betray diane
chris is betray by diane
chris love helen
chris is love by helen
john hit john
john is hit by john
john love chris
john is love by chris
diane hit helen
diane is hit by helen
john hit diane
john is hit by diane
helen betray john
helen is betray by john
john kill helen
john is kill by helen
helen kill john
helen is kill by john
john hug michael
john is hug by michael
diane kill john
diane is kill by john

```
chris hug john
chris is hug by john
john betray diane
john is betray by diane
helen betray michael
helen is betray by michael
chris betray helen
chris is betray by helen
diane hug diane
diane is hug by diane
chris betray chris
chris is betray by chris
john hug diane
john is hug by diane
michael love michael
michael is love by michael
chris hit helen
chris is hit by helen
john kill diane
john is kill by diane
chris hug diane
chris is hug by diane
chris kill diane
chris is kill by diane
john betray chris
john is betray by chris
chris kill helen
chris is kill by helen
diane betray michael
diane is betray by michael
helen hit diane
helen is hit by diane
chris love diane
chris is love by diane
diane hug michael
diane is hug by michael
helen kill diane
helen is kill by diane
```

helen betray helen
helen is betray by helen
michael kill helen
michael is kill by helen
michael betray diane
michael is betray by diane
diane betray john
diane is betray by john
john love helen
john is love by helen
helen hug michael
helen is hug by michael
diane betray diane
diane is betray by diane
michael hug diane
michael is hug by diane
diane love diane
diane is love by diane
helen kill chris
helen is kill by chris
michael love chris
michael is love by chris
chris hug chris
chris is hug by chris
michael love diane
michael is love by diane
john love john
john is love by john
helen hug john
helen is hug by john
helen hit john
helen is hit by john
diane hit diane
diane is hit by diane
john kill chris
john is kill by chris
helen love john
helen is love by john

```
john love diane
john is love by diane
michael hug john
michael is hug by john
```

References

Balogh, I. L. (1994). *An analysis of a connectionist internal representation: Do RAAM networks produce truly distributed representations?* PhD thesis, New Mexico State University.

Berg, G. (1992). Connectionist parser with recursive sentence structure and lexical disambiguation. In *Proceedings of the Tenth National Conference on Artificial Intelligence - AAAI-92, San Jose, CA, July 1992*, pages 32–37, Menlo Park, CA. AAAI.

Blank, D. S., Meeden, L. A., and Marshall, J. B. (1991). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. Technical Report TR 332, Department of Computer Science, Indiana University, Bloomington, Indiana.

Blank, D. S., Meeden, L. A., and Marshall, J. B. (1992). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In Dinsmore, J. (Ed.), *Symbolic and Connectionist Paradigms: Closing the Gap*, pages 113–148. Hillsdale, NJ: Lawrence Erlbaum Associates.

Callan, R. and Palmer-Brown, D. (1997). (S)RAAM: An analytical technique for fast and reliable derivation of connectionist symbol structure representations. *Connection Science*, 9(2):139–159.

Chalmers, D. J. (1990a). Syntactic transformations on distributed representations. *Connection Science*, 2(1–2):53–62. Reprinted in (Sharkey, 1992), pages 46–55.

Chalmers, D. J. (1990b). Why Fodor and Pylyshyn were wrong: The simplest refutation. In *Proceedings of The Twelfth Annual Conference of the Cognitive Science Society, Cambridge, MA, July 1990*, pages 340–347, Hillsdale, NJ. Lawrence Erlbaum Associates.

Chandola, A. and Mahalanobis, A. (1994). Ordered rules for full sentence translation: a neural network realization and a case study for Hindi and English. *Pattern Recognition*, 27(4):515–521.

Chrisman, L. (1991). Learning recursive distributed representations for holistic computation. *Connection Science*, 3(4):345–366.

Cottrell, G. W. (1985). *A Connectionist Approach to Word Sense Disambiguation*. PhD thesis, Computer Science Department, University of Rochester. Available as TR-154.

Feldman, J. and Ballard, D. (1982). Connectionist models and their properties. *Cognitive Science*, 6:205–254.

Fodor, J. A. and Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: a critical analysis. *Cognition*, 28(1–2):3–71.

French, R. M. (1994). Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, Atlanta, GA, August 1994*, pages 335–340, Hillsdale, NJ. Lawrence Erlbaum Associates.

Hadley, R. (1992). Compositionality and systematicity in connectionist language learning. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 659–664, Hillsdale, NJ. Lawrence Erlbaum Associates.

Hammerton, J. A. (1998). Holistic computation: Reconstructing a muddled concept. *Connection Science*, 10(1):3–19.

Henderson, J. B. (1994). *Description Based Parsing in a Connectionist Network*. PhD thesis, University of Pennysylvania.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In *Parallel Distributed Processing: Explorations in the microstructure of cognition*, pages 77–109. Cambridge, MA: MIT Press.

Ho, E. K. S. and Chan, L. W. (1995). Linearization + confluent inference = efficient connectionist representations of parse trees for grammatical inference. In *Proceedings of International Symposium on Artificial Neural Networks, Hsinchu, Taiwan, Dec 18–20 1995*, pages IS–24–IS–31.

Ho, E. K. S. and Chan, L. W. (1996). Confluent preorder parser as a finite state automata. In *Proceedings of International Conference on Artificial Neural Networks ICANN'96, Bochum, Germany, July 16–19 1996*, pages 899–904, Berlin. Springer-Verlag.

Kalman, B. L. and Kwasny, S. C. (1994). High performance training of feedforward and simple recurrent networks. Technical Report WUCS-94-29, St. Louis: Department of Computer Science, Washington University.

Kohonen, T. (1990). The self-organising map. *Proceedings of the IEEE*, 78(9):1464–1480.

Kwasny, S. C. and Kalman, B. L. (1995). Tail-recursive distributed representations and simple recurrent networks. *Connection Science*, 7(1):61–80.

Levy, J., Baraktairis, D., Bullinaria, J., and Cairn, P. (Eds.). (1995). *Connectionist Models of Memory and Language*. London: UCL Press.

MacDonald, C. and MacDonald, G. (Eds.). (1995). *Connectionism: Debates in Psychological Explanation*. Oxford: Blackwell.

Maskara, A. K. (1993). *Recurrent Neural Networks and Grammatical Inference*. PhD thesis, Polytechnic University, Brooklyn, NY.

Miikkulainen, R. (1993). Subsymbolic case-role analysis of sentences with embedded clauses. Technical Report AI 93-202, University of Texas at Austin, Austin, TX 78712.

Miikkulainen, R. (1994). Integrated connectionist models: building AI systems on subsymbolic foundations. In *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence, New Orleans, LA, November 1994*, pages 231–232, Los Alamitos, CA. IEEE Computer Society Press.

Minsky, M. L. and Papert, S. (1969). *Perceptrons*. Cambridge MA: MIT Press.

Niklasson, L. and Bodèn, M. B. (Eds.). (1994). *Connectionism in a broad perspective – selected papers from the Swedish Conference (University of Skovde, Sweden, 9–10 September 1992)*. Chichester: Ellis Horwood.

Niklasson, L. and Sharkey, N. E. (1997). Systematicity and generalization in compositional connectionist representations. In Dorffner, G. (Ed.), *Neural Networks and a New Artificial Intelligence*, pages 217–232. London: Thomson Computer Press.

Niklasson, L. and van Gelder, T. (1994). Can connectionist models exhibit non-classical structure sensitivity? In Ram, A. (Ed.), *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, Atlanta, GA, August 1994*, Hillsdale, NJ. Lawrence Erlbaum Associates.

Plate, T. A. (1991). Holographic Reduced Representations: Convolution algebra for compositional distributed representations. In Mylopoulos, J. and Reiter, R. (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney, Australia, August 1991*, pages 30–35, San Mateo, CA. Morgan Kauffman. Reprinted in Mehra P. and Wah B.W. (editors). Artificial Neural Networks: Concepts and Theory, Los Alamitos, CA, IEEE Computer Society Press 1992.

Plate, T. A. (1993). Holographic recurrent networks. In Giles, C. L., Hanson, S. J., and Cowan, J. D. (Eds.), *Advances in Neural Information Processing Systems 5: NIPS * 92, Denver, CO, November 1992*, pages 34–41, San Mateo, CA. Morgan Kaufmann.

Plate, T. A. (1994). *Distributed Representations and Nested Compositional Structure*. PhD thesis, University of Toronto.

Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46(1–2):77–105.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing* (2nd ed.). Cambridge University Press.

Reilly, R. G. and Sharkey, N. E. (Eds.). (1992). *Connectionist approaches to Natural Language Processing*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Rumelhart, D. and McClelland, J. (1986). *Parallel Distributed Processing*, volume 1 & 2. Cambridge, MA: MIT Press.

Selman, B. and Hirst, G. (1985). A rule-based connectionist parsing system. In *Proceedings of the Seventh Annual conference of the Cognitive Science Society*, pages 212–221, Hillsdale, NJ. Lawrence Erlbaum Associates.

Sharkey, N. E. (1991). Connectionist representation techniques. *Artificial Intelligence Review*, 5(3).

Sharkey, N. E. (Ed.). (1992). *Connectionist Natural Language Processing:Readings from Connection Science*. Oxford: Intellect.

Sharkey, N. E. and Reilly, R. G. (1992). Connectionist natural language processing. In *Connectionist approaches to Natural Language Processing*, pages 1–12. Hillsdale, NJ: Lawrence Erlbaum Associates.

Sharkey, N. E. and Sharkey, A. J. C. (1992). A modular design for connectionist parsing. In Marc, F. J. and Drossaers, A. N. (Eds.), *Twente Workshop on Language Technology 3: Connectionism and Natural Language Processing, Enschede, The Netherlands, May 1992*, pages 87–96. Department of Computer Science, University of Twente.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1–2):159–216.

Sperduti, A. (1993a). Labeling RAAM. Technical Report TR-93-029, International Computer Science Institute, Berkeley, California.

Sperduti, A. (1993b). On Some Stability Properties of the LRAAM Model. Technical Report TR-93-031, International Computer Science Institute, Berkeley, California.

Sperduti, A., Starita, A., and Goller, C. (1995). Learning distributed representations for the classfication of terms. In Mellish, C. S. (Ed.), *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada, 20–25 August 1995*, pages 509–515.

Stolcke, A. and Wu, D. (1992). Tree matching with recursive distributed representations. Technical Report TR-92-025, International Computer Science Institute, Berkeley, California.

Sun, R. (1992). On variable binding in connectionist networks. *Connection Science*, 4(2):93–124.

Sun, R. and Bookman, L. (Eds.). (1995). *Computational Architectures Integrating Neural and Symbolic Processes*. Kluwer Academic.

Touretzky, D. S. (1986). BoltzCONS: Reconciling connectionism with the recursive nature of stacks and trees. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society, Amherst, MA, August 1986*, pages 522–530. [N.P.].

van Gelder, T. (1990). Compositionality: A connectionist variation on a classical theme. *Cognitive Science*, 14:335–364.

van Gelder, T. (1991). What is the "D" in PDP? A survey of the concept of distribution. In Ramsey, W., Stich, S., and Rumelhart, D. E. (Eds.), *Philosophy and Connectionist Theory*, pages 33–60. Hillsdale, NJ: Lawrence Erlbaum Associates.

Waltz, D. L. and Pollack, J. B. (1985). Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, 9:51–74. Note: A Hybrid Connectionist/Symbolic parser.

Ward, N. (1994). *A Connectionist Natural Language Generator*. Norwood, N.J.: Ablex. Revised and extended version of *A Flexible, Parallel Model of Natural Language Generation*, Ph.D. thesis and Technical Report UCB–CSD 91/629, Computer Science Division, University of California at Berkeley.

Weber, V. (1992). Connectionist unification with a distributed representation. In *Proceedings of the International Joint Conference on Neural Networks – IJCNN '92, Beijing, China*, pages 555–560, Piscataway, NJ. IEEE.

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