University of Groningen

Connectionist lexical processing

Stoianov, I

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2001

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Stoianov, I. (2001). *Connectionist lexical processing*. s.n.

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license. More information can be found on the University of Groningen website: https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment.

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 11-10-2022

Part III

HOLISTIC LANGUAGE MODELLING

Chapter 6

RECURRENT AUTOASSOCIATIVE NETWORKS

6.1 Introduction

Chapters 4 and 5 presented a work on two lexical problems, where the words were represented as sequences built out of letters or phonemes. Fortunately, there are only 26 Latin letters and some 44 phonemes used to represent the sounds of the Dutch language, and those could nicely be encoded either with a properly selected feature set, or simply by assigning one neuron to each of them. Unfortunately for connectionist natural language processing, however, natural languages consist not only of letters/phonemes and words, but also of more complex items such as phrases, sentences, paragraphs, stories, etc. Phrases are built out of words; sentences in turn are built out of phrases, etc. To make the picture even more difficult, the items at each of those levels, including the lexical one, are associated with a variety of other representations (meanings). The idea of sequential association, exploited in Chapter 5 on Grapheme-to-Phoneme Conversion, works to some extent, but it is unlikely that very complex sequences would be dynamically associated with other events spanning a long period (Thornton 1996). It is simply more efficient to develop representations of less complex dynamic objects, which can be included in more complex sequences.

Therefore, natural language modeling, and more generally, cognitive modeling, needs an adequate hierarchical system of representations that can represent a system of such (external) structured dynamic objects –

words, sentences, but also movements, actions, etc., in the extra-linguistic domain. What we distinguish as letters, words, sentences, etc., need to be encoded in a proper and systematic manner, permitting direct, "*holistic*" operations over the resultant abstract representations rather than over their external sequential forms (Chalmers 1990, Blank, Meeden & Marsgall 1991, Hammerton 1998*b*). Operations on complex (symbolic) objects are regarded as holistic if they do not rely on a prior analysis of the external form of the objects, but act upon their internal (holistic) representations. It will be further required that representations be *static*, unique characterisations of the original objects. They should allow holistic transformations and associations to representations of other modalities – visual, effectual, etc.

It is common in connectionist natural language modeling to use localistic and hand-crafted feature-based encoding (Seidenberg & McClelland 1989, Elman 1988, Plaut et al. 1996, Henderson & Lane 1998), which restricts the capacity of the processing system. It would be preferable that representations evolve in the course of experiencing the language (data) in its external sequential form, which would also be in accordance with our capacity to learn any language without any prior knowledge of it, as well as any other system.

A first attempt to build such representations was suggested by Pollack (1990). He extended the static Multilayered Perceptron (Rumelhart, Hinton & Williams 1986) to the Recursive Auto-Associative Memory (RAAM) model, which develops compact distributed representations of static input patterns through autoassociation. The RAAM was further extended to a Sequential RAAM (SRAAM) for sequential processing. However, the implementations of the latter model have had variable success even when applied to trivial data (Chalmers 1990, Blank et al. 1991, Blair 1997, Kwasny & Kalman 1995, Hammerton 1998*b*).

The development of global-memory recurrent neural networks, such as the Jordan Recurrent Networks (Jordan 1986) and the Simple Recurrent Networks (SRN) by Elman (1988) stimulated the development of models that gradually build representations of their sequential input in this global memory. The Sentence Gestalt Model (St. John & McClelland 1990) gradually encodes the input words into a gestalt and questions it further for roles with another static neural network. A similar architecture, known under the name "Movie Description Network", was trained to gather representations of a sequential visual input (a movie) and to describe it with some simple language (Cottrell, Bartell & Haupt 1990). A more recent implementation of the SRAAM also employs SRNs in order to build representations of the sequential input (Kwasny & Kalman 1995) .

In this chapter, I present another connectionist architecture designed to build and process a hierarchical system of static distributed representations of complex sequential data, which I proposed earlier in (Stoianov 1999, Stoianov 2000*b*). It follows the idea of building complex static representations of the input sequence, but has been extended with the ability to reproduce those static representations into their original form by building unique representations for every input sequence. The architecture consists of sequential autoassociative modules – Recurrent Autoassociative Networks (RANs). Each of these modules learns to reproduce a set of input sequences and, as a side effect, develops static distributed representations of those sequences. The complete architecture for processing sequentially represented hierarchical input data consists of a cascade of RANs.

A RAN-cascade encodes structured sequences and decodes their static representations in the following way: During sequence encoding, each RAN gradually gains in its context layer a representation of the incoming tokens until a proper delimiter marker is received. The input data at the lowest-level RAN module may be sequences of percepts from the external world. Every other higher-level RAN module receives as input sequences of static representations that the RAN module from the immediate lower level has produced. To decode a static representation, it is set to the hidden layer of the correspondent encoding RAN. Then, this RAN starts to cycle – producing patterns at the output layer and copying the hidden layer to the context one – until an end-of-sequence token is produced. If the current RAN operates at a higher level, its output could be sent to the hidden layer of the immediate lower level RAN, and sequentially decoded. Otherwise, the output of this RAN is also the output of the cascade. The output of the lowest-level module can also be associated with an effector. Then, given a static representation set to an RAN hidden layer in this cascade, the attached effector would sequentially receive commands during the unpacking process.

As we see, the RAN is a recurrent neural network which conforms to the dynamics of natural languages. Also, RANs both produce representations of sequences and interpret them by unpacking back to their sequential form. The more extended architecture – a cascade of RANs – resembles the hierarchical structure in natural languages.

Furthermore, given a representative training environment, this architecture has the capacity to develop distributed representations in a *systematic* way. The question whether connectionist models can develop systematic representations has been discussed ever since the challenge put by Fodor & Pylyshyn (1988) that only classical symbolic systems can guarantee systematicity (see Aydede (1997) for review). Connectionist systems claimed to meet this challenge are the (S)RAAM, and the Smolensky's tensor products (Smolensky 1990), among others. Later in this chapter I will argue that the RANs also provide an account of systematicity and will present a number of experiments testing this thesis. Therefore, I believe that the RAN and the RAN cascade can participate in a more explanatory cognitive model, where the distributed representations they produce are extensively transformed and associated. Chapter 7 will elaborate this problem more extensively, presenting experiments on *Holistic Computations*.

This chapter continues with a discussion of the hierarchy in dynamic data in the next section. There follows a review of connectionist sequential processing, after which the RAN model is presented in detail in section 6.4. In the same section, the RAN model is illustrated with two examples: developing representations of syllables and developing representations of numbers. The cascade architecture is given in section 6.5, where a two-level representation of words is presented too. Next, I discuss some cognitive aspects related to the RANs and how this architecture might provide some answers for cognitive modeling. After a discussion of the RAN capacities and the representations it develops in section 6.7, the chapter will finish with a conclusion.

6.2 Sequences, Hierarchies and Representations

Static objects and dynamic processes are mutually interconnected. On one hand, dynamic processes are ultimately composed of sequences of static objects but on the other hand, the same dynamic processes are generated by single objects and might be represented by these objects. This is more explicit in discrete dynamic objects, such as sequences composed of discrete data. The sequences consist of strings of tokens, but these sequences are entities by themselves too and they might build even more complex sequences. Therefore, sequential data might have some underlying structure more complex than linear, that is, there might be some hierarchy within long sequences composed out of basic tokens.

Let us consider, for example, the organisation of the natural languages. As discussed earlier in Chapter 2, there are basic tokens – phonemes or letters; next, there are words consisting of sequences of letters or phonemes, phrases consisting of sequences of words, and so on (6.1). That is, in natural languages, sequential objects are part of other, more complex sequences.

(6.1) ((J o h n) ((l o v e s) (M a - r y)))

Hierarchical objects are better suited than linear objects to represent, process and transmit information, which is apparently shown by evolution: natural languages are hierarchically organised. Firstly, components of the structures can be referred to, which allows natural *compression* of the transmitted information. Next, since the sub-components of a given component of the structure are more frequently used together (e.g. the letters 'J', 'o', 'h', 'n' of the word "John"), that is, there is certain redundancy in the exchanged information, the receiver can predict the latter sub-components (e.g., 'h' and 'n') given certain initial left context ('J' and 'o' for this example), which in turn makes the transmission of the information content of the sequences more *reliable*.

Sequential data that have such composite structure might have very long external representations, that is, representations consisting of rows of lowest level tokens. In natural languages, for example, an average sentence might have some 50 characters, and one paragraph might consist of more than 200 characters. This data is difficult to represent and process in this external form. It has a structure, and we organise and remember the language we experience in accordance with this structure.

Simple evidence in support for this idea comes from the natural language mechanisms for referring to linguistic elements presented earlier in the course of the speech act, e.g., the use of pronouns (*this*, *that*, *he*, *she*, *it*) to refer to phrases. When it comes to referring to already presented structures, we prefer to use links to them instead of repeating their external forms. The definite noun phrases and pronouns are just simple external expressions of the links to the internal representations of earlier structures, and we use them very often. This makes speech communication easier and faster, although there might be some difficulties in resolving such links (so-called anaphora resolution).

In addition, single internal representations are more economical to use when associating linguistic expressions with visual objects, actions, etc. Such associations could be made by using the internal representations of complex objects instead of using their external linguistic representations. Therefore, if we want to model a cognitive system dealing with a large variety of multi-modal data, this data should be properly organised in a hierarchical system of such internal representations.

Similar systems of representations naturally occur in symbolic modeling: external *terminal* tokens are organised in a system of rules with the help of internal *non-terminal* nodes, which in turn are similar to the internal static representations discussed. People still argue that because of this organisation of the symbolic approach and its unlimited representational capacity, cognitive modeling should be based on the classical "Language of Thought" (Fodor & Pylyshyn 1988), which makes use of syntactically structured representations and rules defined over those representations.

Connectionists object to this approach, mainly because of the so-called *symbol grounding problem* – the problem of explaining the relations between symbols and complex percepts representing objects from the environment where cognitive agents exists (Searle 1984, Dorffner 1991). Connectionist architectures are particularly good at associating low level data – percepts or commands to effectors – to higher-level tokens. Therefore, more recent works suggest cognitive modeling with hybrid systems, instead, which combine symbolic systems dealing with high-level symbolic tokens and connectionist models that ground (associate) those tokens to low-level data (Harnad 1990).

Another problem concerning symbolic approaches is related to the "hardness" of the rule-based logical computations they are based on (Smolensky 1991). Models should accommodate the "softness" of cognition as connectionism does, by processing data in a fuzzier, stochastic manner.

Nevertheless, for proper high-level cognitive modeling, connectionism is still looking for an answer to the question of how to organise the information coming from external sensors. The available connectionist solutions are still not satisfactory. Although the implementation of the sequential RAAM proposed by Kwasny & Kalman (1995) promised to provide a better solution than the standard sequential RAAM, Hammerton (1998*b*) found that in practice this model did not learn even trivial data well. Therefore, the door is still open for other solutions and in section 6.4 of this chapter, I propose another model: a cascade of Recurrent Autoassociation Neural Networks to build a system of such representations.

In the next section I will present some details about a few connectionist architectures designed to develop representations of sequences, but before that, I will outline more explicitly some features that connectionist systems and distributed representations must meet.

Principles in developing distributed representations

Firstly, the lowest level of representations should simply be *perceptual* in case of dealing with sensors, or *effectual* when producing actions. This is the natural data that we experience and it is the source of any further processing. Therefore, the structure of these representations should follow the structure of the external data as close as possible. Visual processing systems, for example, usually have two-dimensional array of receptors detecting light (the retina), because there would be no physical basis for a putative system inputting data in three spatial dimensions (of course, stereoscopic vision provides some three-dimensional data). Auditory sensors, on the other hand, are strictly speaking of a zero-dimension in space, since this is enough to detect auditory signals that take place over time. Nevertheless, for more efficient processing, two dimensional arrays of sensors are implemented in humans (in the cochlea).

Next, representations should develop *emergently*, in the course of experiencing the external (training) data. That is, higher-level representations should not be externally imposed. Different levels of representations should be develop in a *well-founded* way, one after another, possibly with simultaneous refining of lower-level representations. This gradual representation development, together with consequent associations to representations from other modalities would solve the *symbol grounding problem* (Harnad 1990).

Further, each representation should *uniquely* represent the corresponding sequential data, allowing (approximately) exact decoding. Some of the connectionist models designed to develop representations of sequences do not ensure this, which would lead to potential conflicts later when using those representations (e.g., the Sentence Gestalt models, (St. John & McClelland 1990)). During thinking, we produce and articulate language sequentially (usually silently), by translating representations of structures back into their external temporal forms, and finishing this process by executing motor commands. Sensoric data is both extensively encoded and decoded. It is known from neuro-imaging studies that visual signals flow not only from lower- to higher-level processing regions, but also in the reverse direction, providing top-down expectations about the lower-level patterns.

Similarity across data should also be accommodated properly. Representations shall be developed in such a way that similar data result in similar distributed representations. This is very important for a number of reasons, including system reliability – this way, noise or data degradation would not immediately degrade the performance of the system. Also, this is a way to introduce *generalisation* – good performance on data unseen during training (or early experience).

Finally, Fodor & Pylyshyn (1988) required that cognitive representations should have an explicit structure that allows a combinatorial syntax, as in the classical symbolic systems. I will weaken this by requiring an *implicit structure* in these representations, allowing a combinatorial syntax understandable to other computational models, including connectionist ones. This requirement is necessary for emergence of *systematicity* among these representations and for *holistic computations* (Hammerton 1998*b*, Smolensky 1991). The latter are structural operators acting upon holistic (static) representations of sequences/structures, instead of first decoding these structures into parts, then processing them and consequently recoding into holistic representations. Systematicity might be considered a key to higher-order cognitive processing and will be discussed further in section 6.7.

A failure to accommodate these principles results in models with limited capacities and no lasting implications, which is typical for many of the reported architectures that feature, for example, static language processing and hand-coded data encoding. We encounter language sequentially, by hearing sounds and building or recognising gradually a number of objects or temporal structures, such as, words, phrases and sentences. The number of those structures is enormous and a designer's hardwired encoding does not seem plausible at all. Also, the number of existing languages and our competence to learn any of them imply that a system of such representations should develop during the communication process.

6.3 Developing connectionist representations of linguistic objects

In this section I will shortly present some of the ideas and models used to develop static representations of linguistic and more generally, sequential data.

Firstly, one can approach language modeling either with a static or a dynamic connectionist model. An example of two views on the same problem is connectionist GPC modeling, discussed in Chapter 5. The basic objects there are words, which Seidenberg & McClelland (1989), Plaut et al. (1996), and most of the other GPC models represent as static feature-based hand-crafted encoding and process as static objects, in contrast to the dynamic approach used in Chapter 5. However, sequential models have difficulties in processing long sequences, as discussed earlier. Also, dynamic data do not allow holistic transformations, where static representations are required. Even more importantly, when the sequential data have an internal structure, it would be an advantage to represent this structure somehow.

In connectionist modeling, static representations have been developed by using various approaches. For example, words were represented statically as a set of letter/token triples (e.g., Seidenberg & McClelland 1989) or by using position-based encoding techniques (e.g., Plaut et al. 1996, Zorzi et al. 1998), which impedes generalisation to polysyllabic words, or to other languages.

When modeling syntax, words have been encoded either localistically, or again, using hand-crafted feature-based schemes (e.g., Elman 1988, Miikkulainen & Dyer 1991, Tabor, Juliano & Tanenhaus 1997).

Recognising the problem, various connectionist techniques that produce static representations of sequential data were developed as early as 1990, e.g., the Recurrent Autoassociative Memory (Pollack 1990) or the Tensor Products technique (Smolensky 1990). Nevertheless, I am not aware of a method that develops static representations of sequences which works as reliably and easily as the MLP in static mappings or the SRN in sequence processing. One of the reasons for this is that although those techniques indeed provide a good theoretical basis for developing representations of sequences, in practice they were found difficult to explore for various reasons, such as practical limitations of their representational capacity, a demand for powerful computational resources, or very long training. Therefore, high-level connectionist language modeling has focused only on small illustrative problems, mostly using the 'hand-crafted' technology, learning only simple grammars and a very limited number of words (Elman 1988, Tabor et al. 1997) or to keep things simple, using part-of-speech labels (TAGs) instead of working with words (Henderson & Lane 1998). This could be attributed not only to the insufficient computational resources, but also to the absence of an adequate working concept of how to develop high-level distributed representations, which leaves the 'market' open for such methods.

Tensor Products

One of the pioneer connectionist works on developing representations of structures was presented by Smolensky (1990), which is based on the tensor product algebra. The tensor products $z = x * y$ of two n dimensional vectors x and y is simply their outer product xy^T and the result is a n by n matrix. Higher order tensor products are built in a similar way by outer multiplication. The resulting matrix of a k-th order tensor product is a k-dimensional matrix with n^k elements, that is, the size of the representations grows exponentially with the dimensionality of the structure to be represented. This is clearly problematic. Yet, one of the very useful properties of this approach is that given that the vectors are orthogonal each other, then any of the vectors participating in this product could be extracted by applying the so-called inner product with the rest of the elements, followed by a proper normalisation. Non-orthogonal vectors could be extracted only approximately, similarly to the single token errors which the connectionist model presented latter in this chapter produces for some of the vectors during the decoding phase.

Convolution methods

Even earlier than Smolensky's work, another even more interesting approach based on the mathematical operations *convolution* and *correlation* was presented as an approach at modeling distributed associative memory. The convolution operation combines two n-dimensional vectors x and y into a vector z of size $2n-1$. Convolution could also be applied recursively. The general k-way convolution results in a $k(n-1)+1$-dimensional vector. Similarly to tensor algebra, there is an approximate inverse operator of convolution – correlation – which can approximately decode the original vectors if those vectors are normalised so that their mean is zero, and variance is $1/n$.

An even more compact, convolution-based method implemented into a connectionist system was presented by Plate (1994), called "Holographic Reduced Representations". This method is based on the more-specific circular convolution, which results in vectors with the same dimension. This way, a k-way circular convolution multiplication of n-dimensional vectors results in a n-dimensional vector. Similarly to the standard convolution, the circular convolution has also an approximate decoding operator – circular correlation, which needs the same data distribution as in the standard convolution, in order approximately to be able to extract the original items.

However, this operator is commutative, that is, $x * y = y * x$, which makes it impossible to preserve order. Nevertheless, there is one interesting property of this operation, namely that it can superimpose vectors and then approximately extract them. This, in turn, allows the using of a lower dimensional scheme for representing structures into a memory trace z by superimposing terms, where each term is a convolution of a positional token p_i and the token c_i placed at that position. Then, the decoding operation of a sequence consists of questioning (correlating) the memory trace z with each position p_i.

As far as the capacity of this scheme is concerned, the author gives an estimation that, 4-item sequences composed out of 1000 different items could be encoded in a 512 items vector with 1% chance for error in decoding. Yet, Weber (1992) when reporting an alternative implementation of this approach noted a worse reliability of the circular convolution than what was originally claimed.

The connectionist implementation of the circular convolution – the "Holographic Recurrent Network" – aims at decoding the holographic representations developed by circular convolution. The network assumes localistic input and output vectors indexing the tokens, which are then transformed into proper basic vectors in the input part of the network. The output of this decoding network is interpreted as a vector of probabilities for each token that it has been convolved with the current positional item. The network has an architecture similar to the Simple Recurrent Networks, but with different computations, which perform convolutions and other operations to decode the trace being decoded. The training procedure aims at more precise decoder tuning, the basic functions being roughly initially encoded into the connectivity and operations themselves. There are also some other modules, for example one that generates the position vectors with which the basic vectors are convolved.

Apparently this is a very interesting approach that uses a mathematically elaborate background theory, implemented in a connectionist architecture. Yet, there are questions about the capacity of the model. Since at certain moments it requires localistic token encoding, there is apparent limitation on the number of tokens possibly being encoded. Even alternative connectionist implementations that avoid the requirement for localistic interface vector encoding would have to deal with the special type of vectors the approach needs – normally distributed, centred around zero, with variance $1/n$. Another problem with its connectionist implementation is its structural and operational complexity: it comprises some processing modules designed to perform operations specific to the model, which questions its cognitive plausibility.

In contrast, the method that I will present later involves very simple principles, widely used in nature, with no restrictions on the patterns to be used. It simply off-loads the task of designing a very specific architecture to the learning algorithm, which finds its own way to develop perfectly decodable meaningful representations.

Synchronous Oscillation

Another very interesting approach that has been exploited for variety of linguistic tasks is based on the synchronous oscillation of roles (variables) and fillers (data), because of which this model is called Temporal Synchrony Variable Binding (Shastry & Ajjandagadde 1993).

The basic idea of this model is very simple: There is a number of units that oscillate with the same frequency, but in different phases. Those units that share common phase oscillate in synchrony and would be temporally bound. On the other hand, units oscillating out of phase would not be bound in time. This way, if there are units that represent roles and other, that represent fillers, then the synchronous oscillation of different roles and fillers would at every given moment represent different configurations – a set of current relationships.

Henderson (1996) showed that since it is an internal property of the model to generalise across entities, systematicity is also an inherited feature of this model. Therefore, the model is plausible for modeling high-level reasoning. More recently, Henderson & Lane (1998) proposed a connectionist implementation of this model for language modeling – the Simple Synchrony Networks. This network is effectively a two-level Simple Recurrent Network, where the first level oscillates across the entities and the second one represents the overall context. In particularly, this network was used for syntactic parsing in language – a linguistic problem that still lacks proper connectionist implementations.

Although an excellent idea for representing various relationships, the synchronous oscillations-based models have one important limitation – that there is a very limited number of entities that can be represented at a given time, which is related to the number of different phases which could exist within a given cycle. Also, as far as holistic computations are concerned, this is not a holistic model, since the relationships are represented both in time and space – it does not develop holistic representations of relationships, but it only temporally represents relationships. Yet, it has a very promising future for short-term memory modeling, for example, for hyppocampus modeling.

RAAM and SRAAM

A significant attempt toward a more systematic way of representing structures and sequences was the development of the Recursive Auto-Associative Memory (RAAM) by Jordan Pollack (Pollack 1990). This architecture is another simple extension of the MLP. RAAMs auto-associate the input data – concatenations of two patterns – and use the activations of the hidden layer neurons to represent these concatenations (Fig. 6.1, left). This is equivalent to learning simple symbolic rules. When applied recursively, that is, when using the representations developed as an input for another compression, the RAAM can learn a grammar and develop representations for the non-terminal symbols in this grammar. This makes RAAM a connectionist implementation of a symbolic processor. However, theoretical problems arise from the connectionist point of view, due to the need for an external symbolic mechanism to store representations. Also, the training process is immensely difficult due to the recursive reuse of the ever-changing representations in the course of the training (Kwasny & Kalman 1995).

If RAAM models a stack, then it can learn and represent sequences – Sequential RAAM (SRAAM). However, this model needs an external stack during the training, which is a step back from connectionism as commented earlier. Then, it decodes sequences inversely, which necessitates another mechanism to invert the decoded sequences. Also, it is difficult for the SRAAM to learn even trivial structures and sequences, which makes it an impractical model.

Also, the RAAM produces representations at every time step, to be reused as inputs, while dynamic objects with uniform structure need single static representations. In that respect, producing single representations of a whole object is more economical – representations at a certain level should be produced only if there is a necessity for them. In natural language, producing static representations of items such as syllables, words and sentences is more useful than producing representations of arbitrary combinations of letters, words, etc. Syllables are involved in morphological transformations. Words are associated with visual patterns and actions; sentences have more concrete semantic meanings. We know that those linguistic objects are distinguished because they have certain functions, and we make use of them. On the contrary, producing representations of arbitrary combinations of mixed items is not so useful.

Another implementation of the sequential RAAM was presented by Kwasny & Kalman (1995). Their SRAAM combines the architecture of the Simple Recurrent Networks and the RAAM idea for autoassociation (Fig. 6.1, right). The stack that the RAAM requires during the learning is encoded in the contextual memory of the SRN. This makes the training faster and easier. Further, Kwasny and Kalman suggested a variation of the mean square error function that boosts small differences between the targeted and the resulted neuron activations. When combined with a modified conjugate gradient training algorithm, this reportedly improved the learning. Still another important contribution in this work was a method for representing recursive structures – by means of symbolic transformation of any tree structure into a binary tree, which can easily be transformed to a sequence. Those two operations are reversible, which allows reconstructing the original structures from their sequential representations.

Exploring holistic computations, Hammerton (1998*b*) attempted to replicate Chalmers's (1990) experiments using the Kwasny and Kalman's SRAAM, which was reported to learn faster and more reliably. For this purpose, he used the corpus from Chalmers (1990) – a small dataset containing 250 sequences built out of 13 distinct items. The standard backpropagation

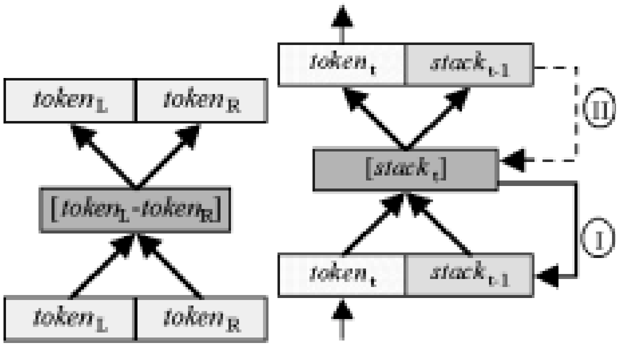


Figure 6.1: (left) RAAM: left and right input tokens are autoassociated, which results in a single compact representation of the input tokens at the hidden layer. (right) SRAAM: based on RAAM and SRN. (I) Compression (or stack pushing): tokens apply one at a time to the input and they are autoassociated, together with the previous state of the context. This results in an even more compact representation of the input sequence at the hidden layer. (II) Decompression (or stack popping): a compact representation applies to the hidden layer and produces the last token from the encoded sequence and the previous state of the stack, which in turn is applied again to the hidden layer.

learning algorithm and two variations of the Kwasny and Kalman's training algorithm were utilised. Hammerton reported that with the best learning algorithm (the modified error function noted earlier and the conjugate gradient training) the network encoded and decoded up to 85% of the 130 training and 87.5% of the remaining 120 unseen testing sequences (page 43), which departs from the reported perfect learning by Kwasny and Kalman (with SRAAM on a more complex task), and Chalmers (with RAAM on the same task). Therefore, Hammerton concluded that "the SRAAM is not as effective a vehicle for holistic symbol processing as it initially appeared".

Also, there are two other problems when these models are used to develop sequential representations. First, as I said earlier, they produce static representations at every time step, which is more useful for representing recursive structures than sequential data. Next, due to the stack-based memory organisation, the input sequences are reproduced inversely. Therefore, one would need another external mechanism to reverse those sequences. The

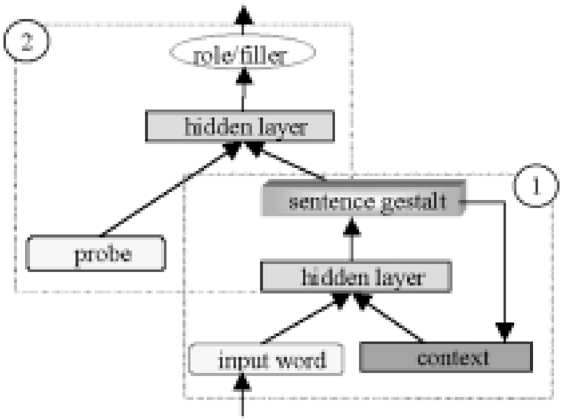


Figure 6.2: Sentence Gestalt Model by St. John. In this architecture, a Jordan recurrent network (1) gradually produces representations of the input sequence. Next, an MLP (2) extracts from these representations some of the constituents of the sentence.

solution I propose in the next section is based on autoassociation and SRNs, too, but it implements a *queue* rather than a *stack* mechanism, which leads to reproducing the sequences in the right order.

Gestalt Models

The first attempt to employ recurrent architectures for producing a 'gestalt' or a single static representation of a sequence of words (statically represented) was made by St. John & McClelland (1990) – the Story Gestalt Model. This model comprises two networks – a Jordan Recurrent Network that gradually processes the input sentence and uses the activations of the output layer to represent the sequence presented as input thus far. This compact representation was called a *gestalt*. The second NN is a static MLP, trained to extract some information of interest for a sequence represented with its gestalt as input to the MLP (Fig. 6.2).

Another similar model – the Movie Description Network by Cottrell et al. (1990) – uses Simple Recurrent Networks to develop representations of simple movies presented as input to the SRN. A second SRN produces a verbal description of the input movie. These are specific, rather than universal, models; they produce representations which are not necessarily unique and can not be involved in a hierarchical system of representations of composite data.

Syllable modeling

The capacity of SRNs to process sequential data was also exploited in another approach, aiming at obtaining static representations of syllables (Gasser 1992). In this model, one recurrent network was trained on a sequential mapping – an input train of phones to an output train of patterns, which are concatenations of the same phones and a static lexical representation of the word the phones belong to. The recurrent layer activations at the end of each syllable were recorded and used by a second network that was trained to unpack these representations to their original sequential form. Assuming the task of word recognition, this scheme requires that both phones and lexical representations be offered during training. Static syllable representations result as a side effect.

This approach takes a direction that is opposite to the gradual building of language representations. Instead of building word representations, it does the opposite – it produces syllable representations in the course of word recognition. In contrast, the approach presented in this chapter takes another direction, consistent with the principle of well-founded structural learning, by processing and developing language items of increasing complexity. Also, one problem with the solution presented by Gasser is that because the packing and unpacking processes are split, this method requires the training sequences to be available during both learning tasks, which is less plausible and increases the learning time. The sequential autoassociative task in the approach suggested in the following section requires only a short-term memory to keep the sequence to be presented for a second time at the output layer and the system must only perceive the input the environment provides throughout the learning (which may last indefinitely: we can keep the learning going non-stop).

6.4 Recurrent Autoassociative Networks

In this section, I will present an architecture designed to develop and make use of static, implicitly structured, interpretable representations of sequences. The proposed model is an extension of the Simple Recurrent Network (Elman 1988), presented in detail earlier in section 3.7.

Recall that the SRN is a recurrent neural network based on the feed-forward Multi Layered Perceptron with a global context memory that stores the most recent activation of the SRN hidden layer[1]. Theoretically, the context layer has the capacity to encode all the information for the input sequence provided to the network since its beginning, that is, since resetting the context. Hence, if the context layer is reset and a sequence is applied, then at every time moment, the hidden and the context layers will contain static distributed representations of this sequence. Yet, as we will see later, the context layer might, but does not necessarily contain *distinct* representations of the input sequences. This rather depends on the learning task.

There are different possibilities to obtain static representations of input sequences. One of them is to train a SRN on the token prediction task (as in the phonotactics learning presented in Chapter 4) and to use the context layer activation after the whole sequence has been processed – similarly to the gestalt models (St. John & McClelland 1990, Cottrell et al. 1990). The networks in those models were trained to predict the next input token, which forced the networks to learn information specific to this particular task, but which is insufficient for developing distinct representations of the input sequences. We need an organisation of the learning task that guarantees that after the training, the distributed representations developed by the network (1) contain all information about the training sequences, that is, (2) they are unique for each sequence. Given (2), we would also like this architecture not only to develop representations of sequences, but also to be able (3) to decode them back into their original sequential form, that is, to reproduce the original sequences.

Representations satisfying the above requirements would evolve naturally if we train the network on an autoassociation task, that is, if we train the network to reproduce the input sequences. However, there is another problem – the timing, when to present the input and the output patterns. One can try to reproduce the current input pattern immediately, which will produce no useful hidden layer representations – there will be no need of a context in this task. A delay of one step, or two, or some other fixed number of steps would train the network to develop information specific to the delayed prediction task (similarly to the GPC learning in Chapter 5), but the representations would still not necessarily satisfy the above condition for uniqueness. In order to produce such specific representations, the network has to be trained on an autoassociative task in which the input sequence is

[1] I will remind the reader that the term "layer activation" denotes a numerical vector with the activations of all neurons in that layer.

reproduced after the whole input sequence has been presented to the input, followed by a unique pattern not present in the original input sequence (a *trigger* '#') that indicates the end of the sequence. The static representation of the input sequences will be just the hidden layer activations at the moment when the trigger '#' has been applied and processed by the network (Fig. 6.3).

Lemma 1
Let there be an RAN trained to autoassociate a set of sequences of patterns as defined above. Then, given a successful training, this RAN will:
(I) reproduce *the original sequences correctly;*
(II) produce distinct *Distributed Representations for each sequence.*

Proof

(I): Appending a "trigger" input pattern guarantees (a) a unique target (the same input sequence) in a unique context to the prediction task, and therefore (b) that the network will reproduce the input sequence after successful training. If the training set contains a sequence S (e.g., *chickens*) that in turn contains other sequence(s) in the training corpus as initial subsequence(s) $S = S_1S_2$ (for the above example: *chicken*, *chick*, and *chic*), then the network will reproduce both the longer sequence S and its subsequence(s) S_1, because the target patterns at the end of the new sequences $[S\#]$ and $[S_1\#]$ will only be the symbols/patterns of these new sequences, in contrast to the original case when the network would have two targets: (a) the symbol/pattern of S after subsequence S_1, and (b) the first symbol/pattern of the sequence to be autoassociated. That is, appending all sequences with '#' makes them unique targets in unique left contexts to the prediction task, or, the empirical conditional probability $P([S\#]/[S\#]) = 1$ for every string S in the training data. Then, given a successful training, the network will reproduce the strings correctly.

(II): Given a task of reproducing the input sequences and successful training, the context layer must necessarily have distinct representations of these sequences after observing them completely: Let there be two different sequences S_1 and S_2, which are part of the RAN training corpus, and let DR_{S_1} and DR_{S_2} be their DRs, produced by the RAN. If $DR_{S_1} = DR_{S_2}$, then the reproduced strings (that are the same as the original strings as just proved) will be the same: $S_1 = S_2$, which contradicts the assumption that $S_1 \neq S_2$. Therefore, $DR_{S_1} \neq DR_{S_2}$.

[end]

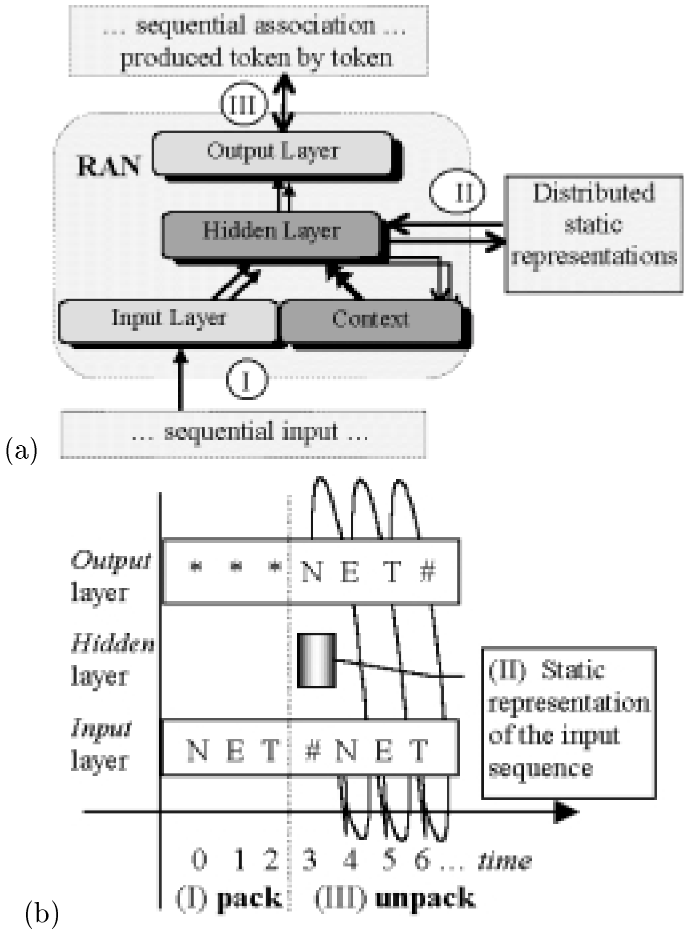


Figure 6.3: Recurrent Autoassociative Network: (a) architecture: based on SRNs and (b) functional processing in time. Operations: (I) encoding an input sequence (time steps 0–3); (II) obtaining or setting a static representation (time step 3), and (III) decoding a static representation to its sequential form (time steps 3-6).

There are some other details to be specified, namely: the input patterns after the trigger has been applied to the input layer, the target output patterns before that and the target output pattern after the sequence has been reproduced. The target output pattern after the reproduction of the whole input sequence is another special pattern, labelled *end-of-sequence*, which signals that the whole sequence has been processed (decoded). The input patterns after the trigger might be either the same trigger, repeatedly provided until the network produces the end-of-sequence pattern (e.g., *chick######*), or what I found more helpful to the network in learning this difficult task – at each time step in that phase to provide to the input layer the last output pattern (e.g., *chick#chick*). The latter approach provides an additional guiding information to the network about how far it has progressed in reproducing the current sequence and experimental work has demonstrated that, indeed, it is easier for the network to learn the autoassociative task with this approach.

With regard to the output target patterns at the time when the sequence is still being entered to the input, the learning algorithm used – Back Propagation Through Time – does not necessarily need target patterns at the earlier moments, provided that an error signal is being propagated back through time. Indeed, the sequential autoassociative task provides such an error that originates in the second phase of the autoassociation (Fig. 6.3, step III). It is also possible to combine the autoassociation task with the standard prediction task, that is, to train the network first to produce at the output layer its anticipations about the sequentially entered input patterns, and next, to reproduce the whole input sequence, whose representation has already been encoded into the context layer.

The latter approach is more plausible, considering the fact that each of us has observed when listening speech in a noisy environment, namely that we use anticipations in interpreting that speech; the same effect occurs in many other cognitive tasks. In such a noisy environment, having got some initial left context, the network could produce: first, the most probable continuants of the context to date and next, the static representation of this sequence (after the triggering pattern has been predicted).

Both approaches were tested, but in the latter case the performance was worse, which I attribute to the higher computational complexity of this joint task – the network had to learn two tasks, which made the learning harder. In contrast, when using the former approach, the network learned the autoassociative task faster and with fewer errors. Since this scheme satisfies completely the above outlined requirements about the representations produced, it will be used for the RAN processing algorithm.

In summary, by training a recurrent neural network on an autoassociation task as described above with a training corpus containing a set of sequences, the network learns to produce unique static distributed representations of these sequences. The hidden layer activations at the moment when the triggering pattern has been applied to the input and processed by the network are used for this purpose. The static representations for each input sequence are unique due to the specific setting of the autoassociative task. After successful training, a RAN network has two functions: firstly, to generate the static representation of a given input sequence (Fig. 6.3: steps I and II) and secondly, to reproduce the original sequential form of a certain static representation (Fig. 6.3: steps II, III).

6.4.1 Summary of the Processing Steps

Training

The RAN model is essentially a Simple Recurrent Network, with a special data arrangement. Therefore, it has the same feed-forward processing steps and learning algorithm. However, when SRNs are trained on the prediction task they can also be trained with the standard error Back-Propagation algorithm, while the RANs can only be trained with the Back-Propagation Through Time learning algorithm, already given in details in section 3.7. This is because in the first feed-forward processing phase, there is no error signal originating at the output layer and back-propagated through the network (Fig.6.3b, phase I), but only error signal computed in the second phase and back-propagated through time (Fig.6.3b, phase III). Therefore, any other learning algorithm that back-propagates error signal through time can be used, too. Yet, it is important that the input-output training data is built properly out of the current training sequence S, according to the scheme outlined earlier in this section, as also shown in Fig.6.3b.

Encoding

After training, the process of sequence encoding is the same as phase (I) of the training algorithm (see Fig.6.3b). The input sequence is applied sequentially to the input layer, followed by the *delimiter* pattern '#'. The static distributed representation of the input sequence is the content of the hidden layer at the moment when the delimiter input pattern has been processed forward through the network.

Decoding

On the other hand, the decoding phase matches the second and the third processing phases of the training algorithm (Fig.6.3b). In order to unpack (decode) a static representation, it is firstly applied to the hidden layer and propagated forward; the resulting output pattern at that time is the first element of the decoded sequence. Next, the new hidden layer activation is copied to the context layer and a new output pattern is produced with the help of the last output pattern provided as an input to the network. This process is repeated until a pattern recognised as a *delimiter* is produced at the output layer.

6.4.2 Experimenting with RANs – I: Learning Syllables

The idea of using the RAN to develop static representations of sequences was firstly tested on a small set of natural language data. A set of 140 distinct syllables was collected from a list of 100 polysyllabic Dutch words, taken from the CELEX lexical data base (CELEX 1993). The syllables were represented as sequences of Latin characters (normal Dutch spelling). The mean length of the syllables was $4.1 \pm \sigma = 1.12$. The characters were encoded orthogonally, in a vector of length 27, that is, for every symbol there was a correspondent neuron which was set active any time this symbol was encoded. The 27^{th} position was activated when the special triggering pattern for the input layer and the end-of-sequence pattern for the output layer were presented or targeted. In order to speed up the training, the *non-active* and *active* neuron states were set to 0.1 and 0.9 correspondingly, which set the working regimen of the neuron activation functions within its almost linear range rather than around the extremes zero and one, where the sigmoid derivatives approach zero. The size of the hidden layer was set to 30.

The network was trained with the Back Propagation Through Time learning algorithm, with learning coefficient η set to 0.15 and momentum term $\alpha = 0.7$. The training was organised in epochs, in which all patterns were presented randomly, according to their frequency of occurrence as found in the CELEX database. After each learning epoch, the network error was measured by encoding and decoding all training patterns and counting the percentage of character and sequence misprediction.

During the training process, the network error followed the standard pattern of quick initial error drop and consequent slow decreasing. After approximately 50 epochs, the network error was reduced to 6% sequence error or 1% token error. Further training reduced the error even more, down to 2.3% sequence error and 0.4% token error – 3 syllables incorrectly encoded and decoded, with one wrongly decoded token for each of them. The goal of this first experiment was just to test generally the idea of developing representations of sequences by sequential autoassociation and apparently, it worked.

From an implementational point of view, it was also interesting to test different strategies for representing the trigger and the end-of-sequence patterns, for instance, whether it is possible to use only the neurons used for encoding the standard input and output patterns, or whether an extra neuron is necessary. Tests were conducted with patterns such as: all neurons active, non-active or taking a value of 0.5. In all those three cases, the performance was worse than the approach with an extra, switching neuron. Therefore, the latter approach was used in the following experiments. It has the additional advantage of always allowing the encoding of a distinct switching pattern, even if the above (all neurons zeros or ones) patterns are recruited by the network in order to represent sequences.

This first experiment suggests that the RANs can learn such a task. Now, it is interesting to see what kind of static representations the network has developed during training. A simple observation of those vectors does not say much (Fig. 6.4, top), because the network has organised the representations just to accomplish its task, not to make them readable by humans, which is the case in the high-level symbolic systems. It is more important that the network itself can "read" those representations, that is, it can reproduce the original sequential forms. Yet, it is also useful to understand the nature of those representations; how they are organised.

For this purpose, a Kohonen Map neural network was trained to cluster them (Kohonen 1984). The Kohonen Map is known as very useful for clustering multidimensional data, similarly to Principle Component Analysis. The learning algorithm of this self-organising network tends to associate different input vectors to different neurons in the map in such a way that neurons responding to similar input vectors are located at close map positions, no matter what is the dimensionality of the data. The resultant map after training is given in Figure 6.4, at the bottom. At the same figure, the minimal spanning tree covering the input vectors is shown.

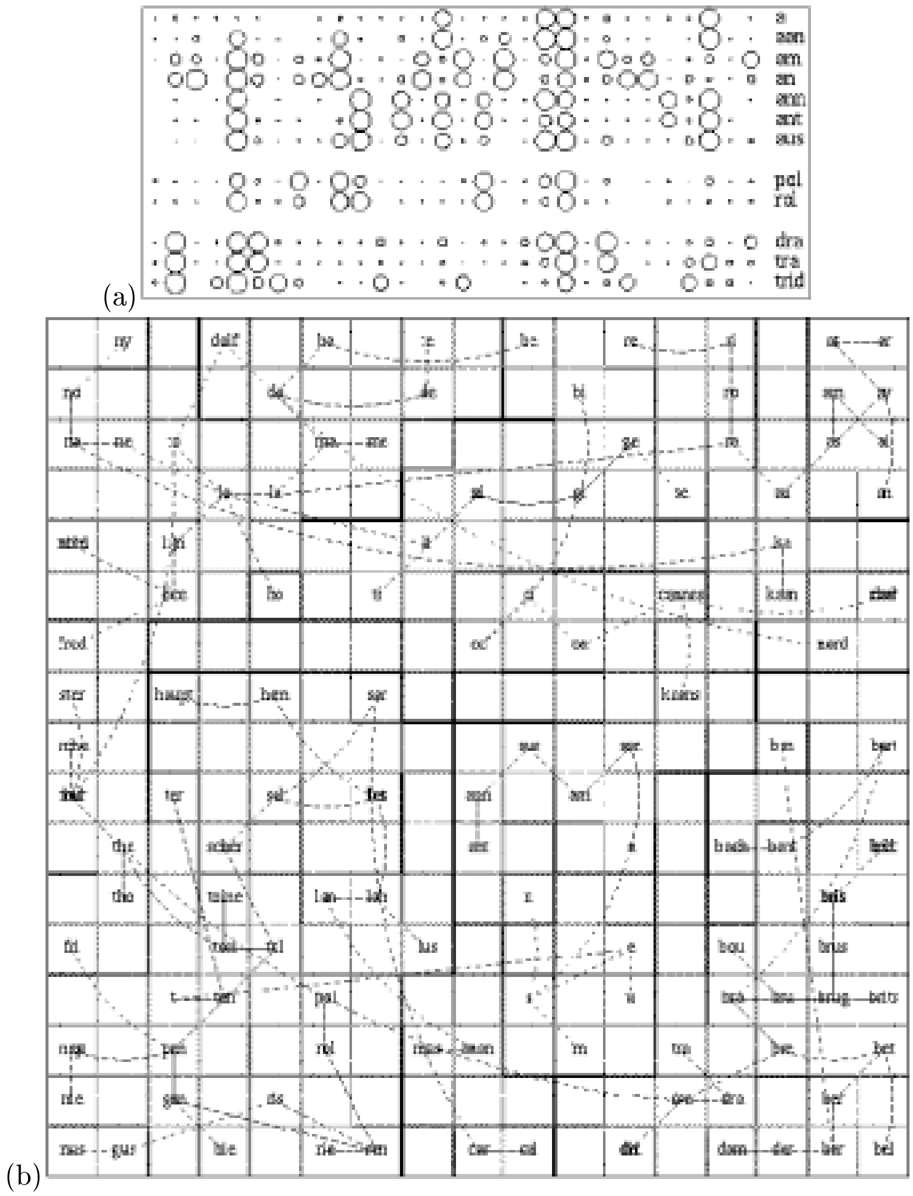


Figure 6.4: (a): Distributed representations (DR) developed by RAN. Each line stands for one DR. Each circle represents one value of a DR. The larger the circles, the greater the correspondent values. (b): A Kohonen Map trained to cluster these DRs (see text). The network maps similar syllables into close positions, which means that their representations are close to each other. The lines connecting cells represent a minimal spanning tree.

As expected, syllables with same initial substrings are located at similar positions (e.g., *ant, ann, aus, am, aan, a*), but also, syllables with similar final substrings are placed at close positions (e.g., *pol* - *rol*; *tra* - *dra*). The RAN very clearly has captured some common external features among the training sequences: similar sequences are mapped into close positions, that is, their distributed representations are similar. This raised expectations that the network would generalise, that is, encode and decode unseen input sequences.

This hypothesis was tested on a corpus of unseen syllables, of which some 90 syllables were correctly processed. This result is quite modest – just about 65% generalisation (as compared to the number of training syllables), but the network was trained on a very small number of sequences. Still, it demonstrated that the RAN did learn some basic principles of how to build such representations. I expect that The more training data, the more knowledge would be gained and the better the generalisation should be.

6.4.3 RAN Experiments – II: Developing Representations of Numbers

While the above experiment demonstrated the capacity of the Recurrent Autoassociative Networks to develop unique representations of a set of sequences (syllables), it did not explore a more *complete* space of sequences. The complete set of letter sequences build of up to 4 letters has about a half a million of sequences – currently infeasible to process. Although the next Chapter 7 presents a work on a larger number of words, the set used there is still far from a complete set. Yet, it is interesting from a theoretical point of view to explore the network's capacity to develop representations of such a full set of sequences. Therefore, in this section, RANs will be trained to develop distributed representations of numbers, ranging from "0" to "$9,999$", represented as sequences of digits in a database M. This way, all possible ordered combinations with repetition of digits, ranking from 1 to 4, will be used.

The main point of interest in this experiment was studying the *generalisation* capacity of the model. For that purpose, the original database M was split into five different training (M_1) /testing (M_2) subsets: M^{50}, M^{33}, M^{20}, M^{10} and M^5, whose training subsets contained correspondingly 50%; 33%; 20%, 10% and 5% of all sequences. RAN models will be trained on those training sets and the generalisation will be tested on the testing sets.

As far as the encoding of the digits building the numbers is concerned, a binary 4-bit (neuron) feature-based representation was used. In addition, there was one more neuron to represent the triggering pattern and two more neurons reserved for further studies, set to zero in this experiments.

Learning all numbers

Before studying the generalisation, a pilot experiment was run aiming at determining the hidden layer size necessary for a RAN to encode the whole data set M. A network with only 20 hidden nodes failed in learning M (34% error). A RAN that managed to score almost perfect performance had 30 hidden neurons – a size that will be used in the course of the main set of experiments.

The RAN with 30 hidden neurons reproduced (and developed representations of) all numbers but two – only "479" and "7984" were reproduced wrongly as "471" and "7904", which is mis-reproducing one bit in each of these cases. However, this almost perfect score could not be achieved by training directly on all words, but only by gradual increasing of the training database, similarly to (Elman 1993).

The main problem in learning M when applying the standard training procedure was the learning of short sequences – the network learned numbers with 3 and 4 digits quickly, but shorter numbers were difficult to learn. Therefore, a new strategy was used, in which the network was first trained on all numbers from "0" to "99", which was followed by training on a set of all numbers from "0" to "999" and then, the network was finally trained successfully on all numbers.

It is interesting that this strategy worked in this case, because in my earlier experimental work it had no positive effect on NN training, similarly to other reports in the literature about this matter (Rohde & Plaut 1997). This means that the success of this strategy might be task-dependent, this problem particularly benefiting from it. Yet, in the following experiments on generalisation, such a gradual training set increase will not be applied, because it would complicate too much the experiments without changing significantly the outcomes.

Studying the generalisation

Since the experiments in this section aimed at quantifying the RAN generalisation, three networks in a pool were trained in parallel, on each of the training databases, independently of each other, and the result of each of the runs will be presented. The results reported here represent the percentage of the incorrectly encoded / decoded numbers (sequences of digits).

	Training/Generalisation Error(%) vs. Training set size(%)				
Network#	5	10	20	33	50
RAN_1	0.6/21.5	0.4/4.5	1.1/2.6	0.5/1.0	0.3/0.6
RAN_2	0.6/14.5	0.5/3.3	2.8/9.3	0.5/1.5	0.7/1.5
RAN_3	4.0/20.5	3.9/9.2	3.2/8.4	1.1/1.7	16.5/18.2

Table 6.1: Performance and Generalisation of three RANs (rows), independently trained to develop representations of strings of digits (numbers from "0" to "9999"), for variable size of the training databases – from 5% to 50% (columns). The generalisation improves as the percentage of the training dataset increases. Reasonably good generalisation appears at the 33% training dataset (0.5% training error / 1.0 test error%).

The networks in both Table 6.1 and in the following explanation will be reported in an order of performance: the better networks first. Note, also, that the generalisation in this set of experiments is not measured as earlier (the number of successfully encoded/decoded unseen sequences vs. the size of the training data set), but rather, as percentage of correctly processed unseen strings, as taken from the whole set of unseen strings. This is because the sizes of the full and the test sets are feasible to explore, unlike in the experiments with syllables and words.

Initially, three RANs were trained on M_1^5. The first two of the networks learned to encode and decode all but 0.6% of the numbers, which means 3 improperly encoded/decoded numbers. The third network fell into a local minimum at 4.0% error. However, the generalisation of all of the networks was very poor – their performance on the testing dataset M_2^5 was 21.5%, 14.5% and 20.5%, correspondingly (see Table 6.1, the first data column). Although the network correctly encoded about 80% of the testing set, since we would like to use the representations developed in other tasks (e.g., for holistic computations, Chapter 7), we would like to have as good performance of the RANs on all targeting sequences as possible.

Training on 10% of the numbers lead to a little better performance on the training set, but also improved the generalisation quite a lot (Table 6.1, the second data column). In training, two of the networks learned to encode and decode all but 0.4% and 0.5% of the numbers from M_1^{10}, which means 4 and 5 improperly encoded/decoded numbers. One network again fell into a local minimum at 3.9% error. However, the generalisation of the first two networks was much better, even if not good enough – their performance on the testing dataset M_2^{10} was 4.5% and 3.3% correspondingly. These percentages may look acceptable, but they are still far from the perfect generalisation, which we would need for more complex problems.

The next experiment was training on the M_1^{20} dataset. This time the networks met more difficulty in learning those 20% of the 10,000 numbers. Only one of the three networks managed to achieve 1.1% error on the training set and 2.6% on the testing M_2^{20} dataset. The other two networks fell into different sorts of local minima – at 2.8% and 3.2%. Even so, the generalisation of the first network is better than the generalisation of the network trained on 10% of the numbers – an expected outcome following the emerging trend of increased generalisation.

The percentage of words that seems to be necessary to have reasonably good generalisation is about 33% – see Table 6.1, the fourth data column. Two of the networks in this experiment learned to decode the training words with 0.5% error and they scored 1.0% and 1.5% generalisation error on the testing M_2^{33} dataset. A more deliberate training on all single and double-digit numbers, together with the training on some 33% of the words would result in excellent generalisation, I suppose.

The last experiment of this serial – training on 50% of the words (M_2^{50}) – confirmed the trend of ever increasing generalisation as the size of the training set increases. The networks here scored quite well, the best of them achieving 0.3% and 0.7% error. Yet, one network once again fell into a local minimum at 16.5% error, which it would have taken long time to exit from, and therefore the training was ceased. Regarding the generalisation, as expected, it was even better than in the earlier case, with 0.6% and 0.7% error for the first two networks.

The important conclusion that could be drawn from this set of experiments is that given enough training data – at least some 30% of the whole dataset – the RAN model may be expected to generalise very well, that is, to encode and decode unseen sequences belonging to the same group as the training data set. The network also generalises reasonably well given smaller percentage of the data – some 10% of the whole data, but then the expected generalisation is somewhat smaller.

It is not surprising that RANs generalise that well – generalisation is one of the most important properties of the connectionist models. This property will also be the basis for another, even more important property of the RAN models – systematicity in the way the DRs are developed – which will be discussed in latter in section 6.7.

6.5 A Cascade of RANs

Recurrent Autoassociative Networks develop distributed representations of sequences consisting of tokens belonging to one level of complexity. Yet, in natural language as in other dynamic objects, data is organised in a hierarchical system, as discussed in section 6.2. Although the distributed representations developed by the RAN model are also organised in a way following the principles of the organisation of the data they represent, it is also interesting to develop a model that *explicitly* represents the underlying hierarchical structure of the external data.

In natural languages, this would be explicit structuring of phonemes into syllables; syllables into words; words into phrases; phrases into sentences, and so on. This hierarchical organisation is expressed quite universally in languages, and it might be easier to model languages by using this hierarchy in a cascaded model, instead of training a single-level model to discover it. This way, RANs used as building blocks in a cascaded organisation would discover the finer organisation of the data at each explicitly stated level and develop representations of subsequences at that level. The representations at any level might be processed in any desired way, including the required communication between the RANs of adjacent levels and possible associations to representations from other modalities, which in turn will be explored in the next chapter.

Now, this idea will be outlined more formally. Firstly, let the sequential input data $(c_1 c_2 \dots)$ have the following hierarchical organisation (6.2). Tokens $c \in \alpha$ at the bottom input level are organised into sequences at level one (S^1); in turn, sequences of S^1 form sequences at level two (S^2), and so on, till the highest level.

$$\begin{aligned} InputData &= (S_1^k S_2^k \dots) \\ S_i^k &= (S_1^{k-1} S_2^{k-1} \dots S_{|S_i^k|}^{k-1}) \\ &\qquad\qquad \dots \\ S_j^1 &= (c_1 c_2 \dots c_{|S_j^1|}) \end{aligned} \tag{6.2}$$

This might also be presented in a graphical way, as in Figure 6.5.

A RAN-cascade developing representations for this data is defined as an ordered set of RAN networks (RAN^1, RAN^2, … RAN^K), where each level RAN^l processes sequences of distributed representation DR^{l-1} from the lower level $l-1$ (of sequences or symbols) and develops distributed

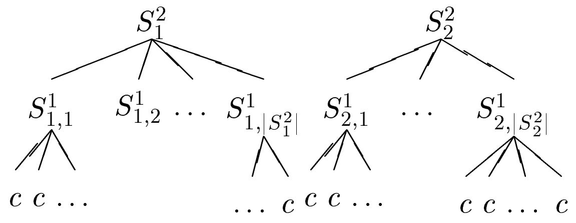


Figure 6.5: Exemplary graphical representation of the structure of two sequences of tokens $S_1^2 = \langle cc \ldots c \rangle$ and $S_2^2 = \langle cc \ldots c \rangle$ with internal three-level hierarchical organisation (external data, representations at level one: S_i^1 and representations at level two: S_i^2). Here c stands for any element of the input (alphabet).

representations DR^l of sequences at level l. The input data of RAN^1 are tokens c from the external (perceptual) level: $(c_1 c_2 \ldots c_{|S|})$. The input data of all other networks $RAN^{l>1}$ are sequences of distributed representations DR^{l-1}: $(DR_1^{l-1}, DR_2^{l-1} \ldots DR_{|S^l|}^{l-1})$.

That is, RANs from each level are fed with sequences of patterns developed at the immediate lower-level RAN; they produce static representations of those sequences, and provide them as input patterns to the immediately higher-level RANs. Also, whenever they are requested, those RAN modules decode (unpack) representations, for example, if the immediate higher-level RAN module needs to decode some object into its external sequential representation (Fig. 6.6).

Discussion

Following the line of representations developed in a RAN-cascade, from the lowest level, to the highest RAN-level, note that this cascade architecture gradually transforms the *temporal*-dimension complexity into a *spatial*-dimension complexity, that is, long sequences of patterns of simple elements are transformed into shorter sequences of complex static representations, distributed among a (possibly increasing) number of neurons. This way, trains of percepts that implicitly contain high-level sequentially represented concepts are transformed into static representations of those concepts and those representations can also be reconstructed back to their external sequential form. This is important if we want to construct a cognitive system

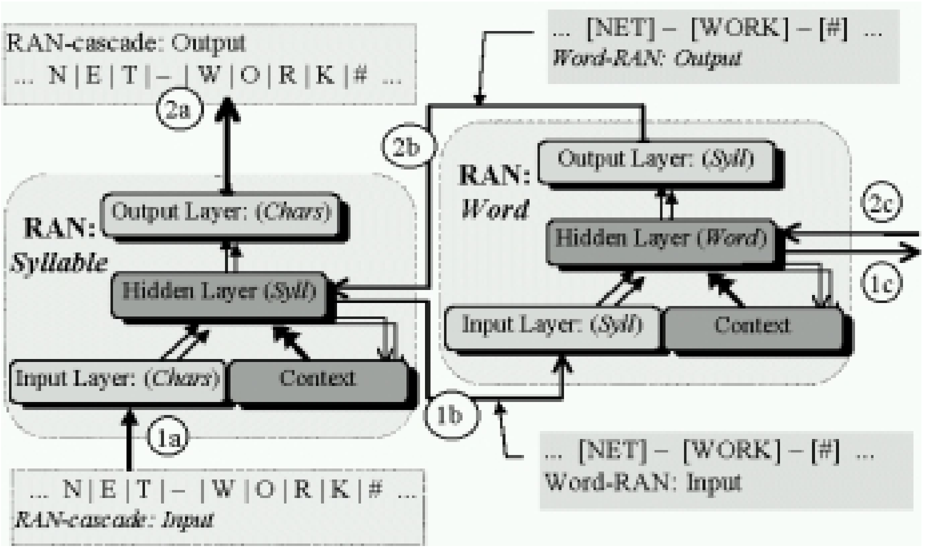


Figure 6.6: RANs and the mechanism of multi-level sequence processing. (1a) A stream of letters is presented to the input of the syllable-RAN, which builds syllable-representations and (1b) provides them to the word-RAN, which, in turn, builds word-representations and (1c) exports them further. On the other hand, if (2c) a word-RAN is presented with a word-representation, it will unpack it to a sequence of syllable-representations and (2b) will provide them to the syllable-RAN, which in turn will unpack them to train of letters and (2a) export at the output of the cascade.

that encodes data from different modalities, associates them, and decodes then back, as it will be done in Chapter 7 on Holistic Computations.

The RANs also provide an account of systematicity among the representations built. The more sequences learned at a certain level, the better the network generalisation will be, that is, after exploring many combinatorial possibilities among the input data, the RAN modules will build static representations in a systematic manner. This was shown with the experiments on the complete set of numbers composed out of sequences of digits in the previous section: the larger the training set, the better the generalisation, that is, more systematically those representations are developed. I will further discuss the problem of systematicity in section 6.7.

Still, there are some questions to be answered. For instance, why do we need such a hierarchical structure, when a single RAN can be trained to produce the same highest-level static distributed representations and to output sequential associations? There are two points which speak in favour of a cascade structure.

First, it is difficult for one homogeneous network to learn long distance relationships (Miikkulainen & Dyer 1991, Christiansen & Chater in press). BPTT learning algorithm propagates error back in time, but the more steps it propagates back, the smaller the influence of those errors is to the earlier steps. Studying different patterns of recursion, Christiansen and Chater found that SRNs can handle well some 5 levels depth of embedding (recursion). The experiments on GPC learning in Chapter 5 showed successful processing of sequences containing up to 10-15 symbols, but there the longer the sequences were, the larger the error was. Hence, learning sequences longer than some 10 tokens would be more difficult rather than learning shorter ones. Even if one uses some techniques to improve the learning, the general tendency to perform worse on longer sequences remains. Hierarchical organisation limits sequential size severely, hopefully limiting error as well.

Another advantage of the hierarchical system of representations is the possibility of accessing intermediate representations of meaningful sub-sequences of the input train of patterns (e.g. words), for example if we need to apply holistic operations or to interpret them. In natural languages, when we hear a sentence, we have an access to its constituents, each of which has some relations with other objects. Only a hierarchical system can handle such intermediate representations and in an RAN cascade static representations at meaningful levels are developed.

However, this cascade structure has one important limitation – that the cascade should be designed in advance and remain fixed throughout its life. This raises two design questions – how to determine the data structure and how to determine the size of the RAN hidden layers, that is, the size of the representations at each level. A related question regards segmentation mechanisms that signal the end of the sub-sequences for every cascade level.

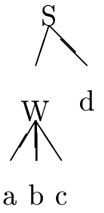
A certain fixed hierarchical structure may be selected according to the natural hierarchy in the input data. For example, when learning natural language, one might favour learning representations of syllables, words, sentences and so on. The input sequence might be split into sub-sequences either by use of some external markers – syllabic delimiter, space between words, full-stop (or even larger pause), or by learning phonotactics and word order and segmenting at proper points, where low-frequency combinations are about to be formed. Cairns et al. (1977) have connectionist experiments on this matter. Likewise, if we use the approach in which RAN both predicts the following pattern and reproduces the sequence (see the previous section), then RAN itself could be used to segment the input sequences.

With regard to the structures that such a cascade architecture can represent, this model imposes a few restrictions. First, one cascade may develop representations of fixed-depth trees, where RANs at each cascade-level process one tree-level. As a consequence, true recursive structures can not be fully represented, only approximated up to a certain depth. On the one hand, this is a disadvantage of the model, but on the other hand, humans do have limitations of their cognitive capacities, which may speak in favour of such an architecture.

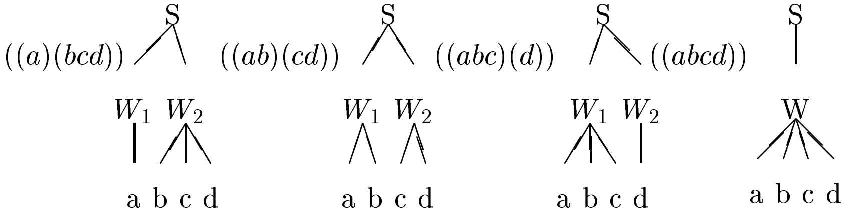
Next, leaves (terminal patterns) may occur only at the bottom level of these trees. For example, sequence (6.3) with internal structure (6.4) is illegal in a two-level cascade because the token 'd' is at the second level rather than at the bottom level. Four of the five structures of this sequence allowed in a two level cascade are given in (6.5). This limitation corresponds to the particular interpretation we have imposed on well-founded representations (see above, section 6.2). Actually, for each structure G in which there are leaves at levels above the deepest (perceptual) level of G, there is at least one alternative structure G' in which the leaves are located at the bottom, perceptual level. To build this structure, one can simply project each leaf to the deepest level, connecting it with intermediate non-terminal tokens. For example, the structure that we need for (6.4) is the third structure in (6.5).

$$abcd \qquad (6.3)$$

$((abc)d)$



(6.4)



(6.5)

An alternative way to deal with truly recursive structures was suggested by Kwasny & Kalman (1995), who proposed to transform tree structures into plain sequences by marking the structures with brackets. Then, such tree structures can be handled by one RAN module. This approach, however, almost triples the size of the original sequences, which in turn makes the RAN learning task very difficult. Also, this method needs an external symbolic device that transforms the structures in both directions, which is a departure from connectionism.

Another solution might involve marking the distributed representations with level-labels and sending them directly to the correspondent RANs when they are encountered. This solution needs a supervisor that distributes the patterns, and it is more plausible than the former solution. However, it violates the principle of well-foundedness.

Finally, sequences with complex (recursive) structure, such as sentences in natural language, may simply be processed with one RAN-level, too. In this case, the structure (e.g., syntax) may evolve implicitly among the distributed representations, instead of being taught explicitly. However, this hypothesis needs extensive examination, which is beyond the scope of this work. Similarly, Elman (1991) trained SRNs to predict words of sentences and later found that the context layer representations developed at each time step also represent the syntactic category of the sequence processed thus far, that is, such implicit structures had been developed, indeed.

One important question concerning SRN cascades is the order in which the networks should be trained at the different levels, and what training regime to select. This includes the question of whether to train the different RAN levels gradually and then keep them fixed, or to keep training them, while training the higher-level networks simultaneously. We can also refine them at a later stage of the training of the higher-level network, because initially the current network will generate too large an error, which might destroy the representations developed. A completely different strategy for training the whole cascade is to train all RANs simultaneously. However, this is a very complicated learning task and it is doubtful whether the network cascade would get to the solution in reasonable time. Instead, building the lower levels first and leaving a small amount of freedom for later change is preferred. A behaviour close to this strategy appears to work with humans – initially people learn to produce simple syllables, next more complex syllables; then words, small phrases, and so on. Which strategy is better is a question for experiments. In the rest of the experiments, the gradual development strategy was chosen – training the networks gradually, starting from the lowest level and keeping them fixed later.

6.5.1 Size of Distributed Representations

Another pre-training step that concerns the design of a single RAN or a RAN-cascade is the size of the hidden layers where the DRs will be developed. Actually, this question has two aspects: on one hand, the hidden layer must store DRs and all other intermediate states necessary to develop those DRs, and on the other, the hidden and the context layers provide driving signals to the SRN in order to get it to perform its task.

So far, little is known about the size of the hidden layer that is required for a normal feed-forward MLP or a recurrent SRN to perform well, that is, to learn its task. This problem is solved empirically rather than theoretically.

In this section I will elaborate on the other aspect, namely the hidden layer viewed as storage for DRs. From that point of view, I will study the question of what size vector space is required in order for this space to represent a certain amount of information. From here on, the size h of a RAN hidden layer should be based on the informational content of the sequences that this RAN is to encode. Parameters in this measure are: the number of distinct input tokens (patterns) $|C|$, the number of sequences to be encoded $|S|$, the maximal length of the sequences to be represented k and the number of distinct neuron states b.

First, let us enumerate the maximal number of distinct patterns P_{max} that the RAN should encode in the hidden layer. The number of strings composed of up to k tokens is $(|C|^{k+1}-1)/(|C|-1)-1$, which is the total number of permutations with repetition of $1,2,3,\ldots k$ items selected from $|C|$ distinct items. To show this, let me remind that the number of strings of length l over an alphabet C is $|C|^l$. Then, the size of the set containing all strings of length $l=1\ldots k$ over an alphabet C is the sum $\sum_{l=1}^{k}|C|^l=(|C|^{k+1}-1)/(|C|-1)-1$.

In the same way, the actual number of training strings is $|S|$. We should count the maximal number of strings, because the network is expected to generalise after the training, that is, to reproduce combinations of items, unseen during training. Next, the number of distinct patterns that RANs need to reproduce a sequence of k items is $2k+1$, which is the number of context states used when auto-associating the input sequence (see section 6.4)[2]. Therefore, the maximal number P_{max} of distinct patterns necessary to produce unique representations of all strings composed of up to k tokens is $[(|C|^{k+1}-1)/(|C|-1)-1](2k+1)$. Therefore, the actual number of necessary distinct patterns P satisfies condition (6.6).

[2] Actually, only one context pattern is used to represent the input sequence, the rest are used (a) to build the desired representation and (b) to reproduce the sequence.

$$|S|(2k+1) \leq P \leq [\frac{|C|^{k+1}-1}{|C|-1} - 1](2k+1) \tag{6.6}$$

Next, the number N of patterns that h neurons can represent, each of them having b distinct states is $N = b^h$, so that the number of neurons necessary to represent N patterns is $h = log_b(N)$. Therefore, by applying a logarithm to (6.6), we derive the number of hidden neurons necessary to encode P distinct patterns: (6.7)

$$log_b(|S|(2k+1)) \leq h \leq log_b([\frac{|C|^{k+1}-1}{|C|-1} - 1](2k+1)) \tag{6.7}$$

from which follows (6.8):

$$log_b(|S|) + log_b(2k+1) \leq h < (k+1)log_b(|C|) - log_b(|C|-1) + log_b(2k+1) \tag{6.8}$$

Formula (6.8) estimates the minimal number of neurons that is necessary to *represent* certain number of sequences with RAN. As mentioned earlier, the right-hand side number should be used in order to allow perfect generalisation. However, formula (6.8) does not guarantee that this number of hidden neurons is enough for the network to *perform* the autoassociation task for all strings and develop their distributed representations. As noticed earlier, in (recurrent) neural networks, hidden layer neurons have other functions, too, related to the network processing. In addition, given the enormous complexity of the learning problem, it is very difficult for the learning algorithm to find proper weights producing these particular representations. Usually, more neurons are necessary to learn a particular task than this theoretical estimations. Therefore, a scaling coefficient $\gamma > 1$ will be applied to (6.8) that will account for these and other factors related to the network processing mechanisms. This will give to the learning algorithm more freedom to find a proper weight set solving the training problem.

Examples

Now, let us find an estimation of the necessary RAN hidden layer size in the previous examples by using (6.8). The base b will be set to 2 states and the coefficient γ to 3.0

The first experiment was learning 140 syllables ($|S| = 140$) built out of 26 distinct letters ($|C| = 26$). The maximal string length was 4, ($k = 4$). Then according to (6.8), $30 < h_{RAN_{syll}} < 63$. In the reported experiments, 30 hidden neurons were enough to encode almost all training sequences.

The second experiment on learning DRs of numbers was more complete, in terms of training set – all strings composed out of up to $k = 4$ digits, each having $|C| = 10$ distinct states. The number of sequences $|S|$ varied, from 100 to 5,000 in the different training sets. Then, the estimation for the hidden layer size h, derived from (6.8) is $30 < h_{RAN_M} < 45$. In that experiment the numbers were encoded with a RAN with 30 hidden neurons.

Finally, I want to note that the computations provided here give just a very rough estimation of the hidden layer size necessary for a particular task. The very important factor of number of hidden neurons necessary for the normal processing of the RAN as a SRN model is not estimated at all, but it is just taken into account with the γ scaling coefficient.

6.5.2 Simulation III: representing polysyllabic words with a RAN-cascade.

A step toward building a hierarchical model of natural language according to the hierarchical design presented earlier is a cascade model producing representations of natural language polysyllabic words. This model involves two RAN modules: a syllable-RAN building static representations of syllables (as sequences of characters), and a word-RAN building static representations of words (as sequences of syllables).

The experimental work reported on here is a natural extension of the experiment described earlier on syllable modeling. The syllable-RAN is the same as before – with 27 input and output neurons, and 30 hidden neurons. This implies that the word-RAN input and output layers should have 31 neurons – one more neuron used to encode the trigger and the end-of-word patterns. The size of the hidden layer of the word-RAN is 30, which is determined by the insignificant complexity of this concrete learning task – there are only 100 sequences to be learned, consisting of some 140 possible syllables, with an average length of the input sequence 4 syllables. In a more realistic model, more hidden neurons would be needed for both RANs, since many more syllables and polysyllabic words should be encoded.

Given a pre-trained syllable-RAN, the training procedure of the word-RAN is organised as follows: First, a training word is randomly selected from the training corpus, which contains pre-syllabified words. Next, for each syllable in the selected word, the syllable-RAN produces a corresponding static representation, which in turn is provided to the word-RAN input layer. The static representations of the syllables belonging to the current word are kept in a buffer until the learning procedure for the current word is finished. When all syllabic patterns from the current polysyllabic words are presented to the input of the word-RAN, it is fed a triggering pattern and the syllabic patterns processed thus far are presented as target patterns to the output layer, one at a time, and error is calculated. The same targeting patterns, with one step delay, are presented to the input layer again (see the previous section and Fig. 6.3 for details). Next, during the second phase of the BPTT learning algorithm, the accumulated error is propagated back through time till the beginning of the sequence. Finally, the weights are updated with the accumulated weight-updating values. After one training epoch – when the network is trained on all words from the corpus – the word-RAN is tested and the training halts or continues, depending on the performance.

The RAN-cascade is tested by encoding the training/testing words and decoding (unfolding) the static representations of these words back to their sequential forms (string of letters), and comparing the resulting strings with the expected strings. The word-RAN error is measured as a percentage of erroneous predictions of (1) letters, (2) syllables and (3) words. Syllables are considered to be predicted correctly if all the correspondent letters are reproduced correctly. Similarly, words are learned if their syllables are reproduced correctly. The performance of the word-RAN in this experiment, after 100 training epochs was as follows: 1.8% character error, 4% syllable error and 6% word error.

Although the model did not perform perfectly, this first experiment with the cascaded model demonstrated that the Recurrent Autoassociative Networks handle distributed patterns produced by the same model well, and that they are capable of developing distributed representations of structured sequences.

6.5.3 A more realistic experiment: looking for systematicity

In this subsection a more realistic experiment will be presented – building distributed representations of some 850 polysyllabic Dutch words, consisting of about 600 distinct syllables. The less complex examples studied earlier suggested what the static representations were like (Fig. 6.4), and more importantly, confirmed that the network generalisation increases as the number of combinations learned by the networks increases.

The cascade processing polysyllabic words consists again of two RANs

Error (%) / Hidden layer	30	50	100
Syllables	72	30	2.9
Chars	18	7.2	1.1

Table 6.2: Performance of the Syllable-RAN trained on 600 distinct syllables, when varying the hidden layer size (30, 50, 100). Syllabic error (1st raw) and char error (2nd raw) is given.

– a syllable RAN and a word RAN. The necessary size of the hidden layer could be estimated also with formulae (6.8). For the syllable-RAN, which deals with letters as input tokens, $|C| = 26$; the number of the studied syllables is $|S| = 600$; their average length $k = 5$. Then, according to (6.8), $30 < h_{RAN_{Syll}} < 100$. Similarly, for the word-RAN: the number of the distinct input patterns is $|C| = 600$, the number of sequences studied is $|S| = 850$, length $k = 5$. Then, we compute that $40 < h_{RAN_{Word}} < 150$.

Still, those estimations are quite approximate and therefore experiments with different hidden layer sizes were run in order to find the most proper ones. The first level (syllable) RAN was tested with 30, 50 and 100 hidden neurons. The network with largest hidden layer resulted in best performance – 0.6% erroneous letter prediction and 2.5% erroneous syllable prediction. As the size decreased, the performance deteriorated significantly. The syllable-RAN with 50 hidden neurons learned some 70% of the syllables, while the network with 30 hidden neurons learned only 28% of them (Table 6.2).

The second level (word) RAN was constructed with 101 input and output neurons, because the syllable-RAN with 100-hidden neurons was used as a first-level network for syllable encoding. To find the most appropriate hidden layer size of the word-RAN, again three networks were tested: with 150, 250 and 350 hidden neurons. This time, the hidden layer estimations here did not work as well as for the first-level network: only a network with 350 neurons managed to learn some 86% of the words. The other networks performed much worse (Table 6.3). Although the performance of the RAN_{Word} measured as percentage of correct letter prediction is much better (2.7%), error accumulates, making the word-level error much larger. Therefore, for perfect performance, more training is necessary.

These experiments support to some extent the estimation of the hidden

Error (%) / Hidden layer	150	250	350
Words	23.1	19.2	14.1
Syllables	7.8	6.8	5.0
Chars	4.2	3.6	2.7

Table 6.3: Performance of the word-RAN trained on 850 polysyllabic words with input vector size 100, when varying the hidden layer size. Word error (1st row), syllabic error (2nd row) and char error (3rd row) is given.

layer size with formula (6.8) that is based on the information content of the hidden layer and size of data to be encoded. Decreasing the number of hidden neurons below the suggested size caused the performance to deteriorate. Besides the minimum number of hidden neurons necessary to encode certain amount of data, another reason for worsening of performance is that the back-propagation learning algorithm more easily finds escape routes from local minima when there are more weights – if a set of weights are trapped into a valley on the multi-dimensional error surface, other weights can drive the network from this point. Therefore, the more complex the task is, the more neurons are necessary. If the number of the neurons seems very large, consider the brain, where billions of neurons participate in different cognitive tasks. Practically, with the ever increasing computational power, this question will be outdated in a few more years.

The more interesting question now concerns the generalisation of the syllable-RAN and the word-RANs, that is, how many words unseen during training would be reproduced correctly by the model. Tested on a larger corpus with 9,000 words and 2,320 distinct syllables, the syllable-RAN successfully reproduced, (and generated unique representations of) other 1150 syllables, which is more than 190% generalisation as opposed to the first experiment with only 65% generalisation (as compared with the number of training examples). These results show that networks trained with more combinatorial possibilities generalise better. In turn, this shows the RAN capacity to produce static distributed representations systematically (see section 6.7 for discussion).

The word-RAN generalised well too, with successful reproduction of 1,500 words unseen during the training, which is about 180% generalisation. It is interesting to note that the word-RAN generalised almost as well as the syllable-RAN after learning fewer combinatorial possibilities than the syllable-RAN did (850 words made out of 600 distinct syllables, while the 600 syllables are made out of 26 distinct letters). I attribute this to the nature of the input data of those two RANs. On the one hand, the syllable-RAN is provided with localistically encoded letters, which gives no prior information about the similarity among the classes they represent. On the other hand, the word-RAN is supplied with much more *meaningful* distributed representations, systematically produced by the syllable-RAN.

This also suggests that if the letters were represented with features (consonant/vowel, voiced, place, manner ...), perhaps the syllable-RAN would learn the task even easier, with fewer hidden neurons and would generalise better.

6.6 Toward Cognitive Modeling

Once we have a method to represent holistically (statically) complex structured data that we experience externally in its dynamic (sequential) form, we can go further and learn some relations between representations coming from the same or different modality channels. Those relationships can be simple associations, or more complex *holistic* transformations – a problem which will be explored in Chapter 7. A complete architecture consisting of channels organising the data (cascaded RAN-modules) and horizontal links across those channels might provide a basis for a cognitive system that would extensively organise data in holistic representations and perform holistic transformations.

For example, processing acoustic input with the cascaded RAN model results in an auditory modality cascade that produces representations of acoustic objects, which at the higher cascade levels might be interpreted as representations of linguistic objects. In the same manner, visual input might be organised in RAN-cascades for visual modality . Similar to the perceptual signals, commands controlling various effectors (e.g., muscles, glands) could also be organised into such cascades. Having a system that develops representations of objects from different modalities, other neural networks could learn various associations between them. The set of associations of higher-level abstracted representation plus the correspondent encoding/decoding cascades might provide a basis for grounding multi-modal representations.

As far as the nature of the static associations is concerned, such complex multi-modal mappings can be effected with any static connectionist model – self-organising or trained by a teacher. What is more, the use of both learning paradigms would be cognitively motivated, because associations could be learned whenever two representations are simultaneously activated and there is a "will" or attention to learn this coincidence. Then, this coincidence would naturally provide a teaching signal to be used by the learning method. Neural Network models that might be used for this purpose are the supervised Multilayered Perceptron, the ART-Map network by Carpenter & Grossberg (1992) or the autoassociative memory by Hopfield (1982), among other connectionist models.

A very sketchy graphical view of this model is given in Fig. 6.7, in which each column represents one RAN-cascade responsible for the organisation of the sequential data incoming from a given modality. On the other hand, the RAN-blocks at each level represent a given conceptual level of data complexity. Those levels should either be designed in advance, or developed gradually, one after another. The communication between RANs from different levels is marked with vertical arrows. In turn, the horizontal links represent holistic (static) associations, which might be uni- or bi-directional.

When sequential data is input to a given modality, that is, to its corresponding lowest-level RAN, higher level holistic representations will be gradually produced. If there are associations learned between the currently active representation and other columns, and if there is a signal that could spread through certain links, the holistic pattern that has just been activated from the bottom-up signal could be transformed into corresponding patterns at (an)other column(s), which in turn might be decoded to lower level sequences. High level representation might also be activated through any channel and decoded to the lowest level sequences, and sequences within other modalities might be produced as well.

Another possible extension toward a global cognitive model might be developing representations of composite input patterns at certain levels of the RAN cascade. This composite pattern might simply be the concatenation of distributed representations from different modalities (Fig. 6.8). The motivation for developing representations of such composite sequential data comes from natural language. Word representations developed on the basis of the external form of the words would only capture systematic dependencies related to combinations of letters (phonemes) into words, but not categorical or semantic information, which might be necessary when processing sentences. Therefore, if we need representations that share properties from different modalities, we could apply this multi-modal approach. For example, the input to a sentence-RAN might consist of static lexical representations and patterns from the visual modality. This would let the sentence-RAN develop sentence representations that properly reflect the vi-

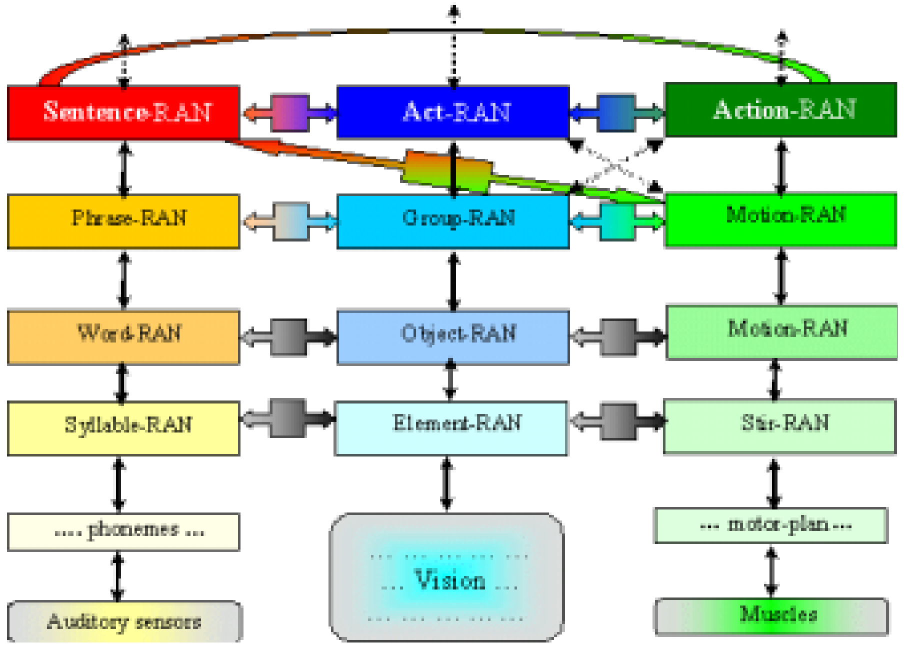


Figure 6.7: Cognitive model based on a network of RAN cascades. Each RAN-cascade (a column) stands for a different modality. The RANs in each cascade represent different conceptual levels. The horizontal and diagonal bi-directional arrows represent static associations between different modalities. In addition to this picture, there should be a central attentive system that directs the flow of activations.

sual meanings of the words, not only their external auditory or visual form.

By using this approach, complex associations that a cognitive system would encounter could be learned. However, there are still a lot of other questions to be answered before talking of a working system – data synchronisation, signal direction, better learning, etc. This huge net of associations needs a central supervisor directing the spread of activations through and across the different channels. Modeling cognitive processes such as attention, awareness, etc., perhaps would resolve some of those questions.

Developing such a cognitive system is not an easy task. The sketchy ideas that were presented here need an extensive elaboration in order for a real

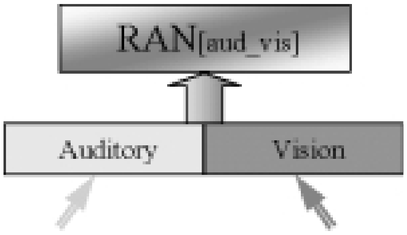


Figure 6.8: RAN developing static representations of multi-modal sequential data (e.g., auditory and visual). The distributed representations developed could feature multi-modal systematicity.

functioning system to be developed. This, however, is not the purpose of this work, which rather aims at modeling various lexical linguistic phenomena. Therefore, I will confine my further remarks to elaborating some holistic transformations in the next Chapter 7.

6.7 Discussion

The basic question addressed in this chapter was how to build static representations of complex sequential data with connectionist models, concerning primarily natural language. Dynamic data exist in all cognitive modalities and it is important to have a mechanism that compresses and uncompresses external dynamic objects into the more convenient static, or “holistic” representations, in order to process them further. Furthermore, I postulated some principles about the way those representations should be developed, in order (a) properly to reflect the well-founded organisation of the data in our cognitive system and (b) to enable further complex processing on those representations, including various holistic transformations. Among those principles are the gradual emergent development, similarity and systematicity.

Association is one of the basic forms of learning. Repetition, or autoassociation provides a powerful mechanism for cognitive development, too. We can observe this mechanism throughout animal species. Baby animals develop initial behaviour without being initially taught, but just by attempting to repeat, imitate their parents. Humans develop language in a similar way: infants initially start to repeat sounds (babbling), next they repeat simple words, small phrases and so on, until they develop full-scale language capacity (Jusczyk 1997). Infants away from language environments simply can not develop language, or have great difficulties in developing language later. Similar motivations drove me to use autoassociation in a connectionist model for developing representations of dynamic data from the external world.

Recurrent Neural Networks are connectionist models that allow us to process dynamic sequential data. The more specific Simple Recurrent Network is a powerful universal model which I have exploited for this purpose, by setting it an autoassociation task and arranging the data to develop the desired static representations. The suggested architecture is called a Recurrent Autoassociative Network. The model was extended further to a cascade of RANs, aiming at developing static representations of hierarchically structured sequential data. In this cascade, RAN modules at each level are designated to develop static representations of different levels of complexity (or different conceptual levels) – words, sentences, and so on.

What is the importance of this model? To what extent does it increase the capacity of connectionist modeling? The discussion in section 6.2 on the representations of sequences that one needs when modeling natural languages and the capacities of the connectionist models presented in section 6.3 clearly demonstrate that the question of how to develop distributed representations of composite dynamic data is still open. Local encoding is restricted, random representations lack systematicity, and the feature-based representations are limited and rather artificial. All those representations keep NNs away from real data. In order to get closer to our cognitive capacity, we need a mechanism that builds representations, starting from the bottom and gradually building ever more complex representations – what RANs do.

The similar idea of building a *gestalt* representing input data was promising, but it was not elaborated further. The RAN model has something in common with the earlier works on gestalt: they all develop representations of the sequential input data in the network global memory. However, they differ in the learning task they set to the SRN and in the way they use this context. The gestalt models use the context developed for information retrieval at every moment, which causes uncertainties – it uses the classical prediction task – while RANs develop unique static representations of the input sequences by a specially arranged learning task that makes the same RANs able to reproduce the input sequence, which in turn does not hold for the gestalt models. The gestalt model needs a second network trained to extract information from the context of the first network, but this information does not necessarily describe the input sequence completely.

The other similar and important architectures RAAM and SRAAM were initially reported to work well on sequential and recursive data, but later experiments did not confirm the initial optimism (Hammerton 1998*b*). Also, the RAAMs need an external stack and boot RAAM and SRAAM reproduce the sequences in inverse order. This might cause interpretation problems when processing longer sequences – it would require an external stack to invert those sequences back to the normal order. Still, RAN clearly owes the idea of autoassociation as a source of developing compact representations to the RAAM model.

The experiments in section 6.4.3 on developing representations of numbers and in section 6.5 on modeling sequences with a cascade of RAN modules demonstrate that RANs can handle both orthogonal and feature-based low-level data (we can assume that this is a kind of perceptual data), and also continuous-values data produced by RANs themselves. The network learned to autoassociate in both cases, although there were more difficulties in learning the latter types of patterns. I attribute this to insufficient computational resources, because the larger the hidden layer is, the better the performance is (see table 6.3). On the other hand, word-RAN generalised very well given the small number of training combinations of syllables, which is due to truly distributed syllabic patterns, as opposed to the localistically encoded input for the syllable-RAN. Similarly, the number-RANs could also generalise well by learning just some 10-20% of all possible combinations of digits, which is also due to the distributed encoding of the numbers – its input patterns – and the general capacity of NNs to generalise on distributed data.

With regard to the hidden layer size that is required for a RAN to learn a particular task, it is difficult to find a theoretical measure for it because the representations are continuous and, theoretically speaking, even one real value number can encode any sequence. However, limitations from the limited effective working range of the sigmoidal activation functions apply and by enumerating the maximal number of distinct patterns to be encoded in the hidden layer, an estimation of the required hidden layer size was derived (formula 6.7). Still, more theoretical and systematic experimental investigation is necessary in order to determine other factors related to hidden layer size, such as factors related to the way the data is processed and encoded in the neural networks, especially in recurrent models.

Systematicity

Neural Networks were reproached by Fodor & Pylyshyn (1988) for not being able to produce systematic representations, which inspired a serious debate about the validity of this challenge (Smolensky 1991, Aydede 1997) among others. Besides having an impact on connectionist modeling, the debate on this human cognitive property is important because it leads toward an explanation of our capacity to think and provides ideas how to model it, that is, how to design thinking machines. The ongoing debate inspired the development of a number of architectures that more or less meet the requirements characterising systematicity (Smolensky 1990, Smolensky 1991, Henderson 1996). In this subsection I will explain how the distributed representations developed by RAN account for systematicity.

It is not easy formally to explain what actually this property means and its definitions often contain tautologies. To begin with, we may say that systematicity concerns the intrinsic connections between some thoughts and some other thoughts. Fodor and Pylyshyn ask why there are such relationships and what cognitive architecture can account for them. The answer they give views the mind as a symbolic system and cognition as symbol manipulation. Accordingly, they claim that the phenomena of systematicity can be explained only with their “Language of Thought” because of its syntactic basis. They also explicitly claim that connectionism can not account for systematicity, because connectionist representations lack syntactic and combinatorial structures, according to them.

A classical example for systematicity is that if we can think of ”Mary loves John”, then we can think of ”John loves Mary”, too. With this simple example, one can distinguish a few descriptive characteristics of systematicity:

- *compositionality* : thoughts are made up of constituents (e.g., “Mary”, “John”, “loves”). However, these constituents do not necessarily have to be explicit in internal representations of thoughts, like in holistic representations.

- *combinativity*: various tokens might be used at the same position in a relation (that is, to have the same function), e.g., “Mary” and “John”. Or, any possible ordered combination of tokens is possible, although some restrictions apply, due to the following property.

- *generalisation*: similar tokens can occupy a given position and therefore sequences/thoughts that have not been experienced earlier can

> be induced. In this example, the tokens “John” and “Mary” are (semantically) similar and therefore “John loves Mary” is inducible from “Mary loves John”.

Fodor’s classical “Language of Thought” respects fully the first and the second characteristics, and partially the third one. The hard rules of logic which underlie symbolism – the background of the classical cognitive explanation – can not account for similarities across all symbols because they do not make use of continuous metrics to compute such similarities. Therefore, in symbolic systems, external (physical) similarities across symbols do not give rise to generalisation unless a system of artificially developed features characterising the symbols is applied or a complicated grammar specifying various classes of symbols is developed. In the above example, if “Mary” and “John” are defined as $\{$*noun*, *animated*, $\dots$ $\}$, then the well-formedness of “John loves Mary” will be inducible from the well-formedness of “Mary loves John”.

On the contrary, an important property of connectionism is generalisation, which is shared by the RAN model, too. Similar items are treated in a similar way. Due to the distributed way of representing data and “rules”, and due to sharing one set of weights throughout the different time steps, similar sequences develop similar representations, that is, similarity in time transfers into similarity in space, as discussed earlier. The similarity across the representations developed is the basis for the generalisation. In this particular example, if there is a more global cognitive system featuring other modalities, such as the visual modality, then similarity between two items in one or more modalities will transfer to other modalities, e.g., language, and therefore the above two sentences will be inducible from each other. On the other hand, similarities across examples within one modality lead to generalisation in this modality. Even more, as noted earlier, generalisation in SRNs/RANs also occur in time due to shared weights.

Yet, some other connectionist representations do not feature some of the other characteristics: localistic representations do not allow similarity to be computed. Feature-based encoding compromises compositionality and enables combinatorial possibilities by using different slots for representing different micro-features, but it is rather artificial and hand-crafted. Models featuring excellent combinativity are those based on the synchronous oscillation (Henderson 1996). However, as I already noted, they do not produce static representations – they span time and space. Other distributed representations, such as those produced by the RAAM and the SRAAM models, show that neural networks can produce distributed representations that have compositional structure, although in an implicit manner (Chalmers 1990, Blank et al. 1991, Hammerton 1998*b*).

Similarly to the RAAM and the symbolic models, the RAN model also produces composite representations. Of course, RANs are not aimed at producing distributed representations understandable by humans. This is reserved for symbolism. The representations that RANs produce are designed to be understandable, firstly, by the RANs themselves and, secondly, by other computational models able to analyse data numerically and eventually extract useful features from this data. RANs can decode representations to an original input sequence – this is a part of the autoassociative task. With regard to the other models, the Kohonen Map that was trained in section 6.4 to cluster the distributed representations of syllables clearly demonstrates that other models can "understand" those representations, too (Fig. 6.4). In this case, the Kohonen Map was just an instrument to persuade the reader that those representations are organised in a systematic way. In addition, similarly to the ability of the RAN to decode the distributed representations, other connectionist models should also be able to extract (information about) the encoded items and to do holistic computations. Experiments with distributed representations produced by the RAAM and the SRAAM show that this is possible (Chalmers 1990, Hammerton 1998*b*); RANs produce distributed representations following the same principles as the SRAAM, only the order of reproducing the sequences is different. The (S)RAAM models implement a stack, while the RANs implement a queue. Therefore, I expect holistic computations will be able to apply on distributed representations developed by the RAN, too – a question that will be explored in the next Chapter.

With regard to the other characteristic of systematicity – exploration of the combinatorial possibilities – the model can indeed learn any ordered combination of tokens and this was shown with a number of experiments. However, this is not enough. Not only should models have the capacity to explore different combinations, but the training environment should provide them, too. Similarly, symbolic learning algorithms can evolve rules in the course of the training only if the learning data provide different examples. This is the same with humans, too. We start to combine words properly after having enough experience in a language environment. Another example is related to algebra. Students learn to add and subtract first by example, and then they realize the nature of the operators "add" and "subtract". With regard to the capacity to explore such combinations, RANs have this capacity, by allowing tokens to take any position in the input sequences.

Similarly to Deacon (1997) I hypothesise that a systematic organisation emerges in the RANs after exploring certain subset of all possible combinations of the input tokens, when the network starts to rely on some common features among the input representations rather than on particular patterns, which Deacon characterises also as example "forgetting". Indeed, the experiment on developing representations of numbers (represented as sequences of digits) in section 6.4.3 showed that the network started to generalise relatively well after observing as little as 10% of the complete dataset, and achieved excellent generalisation after experiencing just some 30% of the data, which shows that the network has discovered the relationships in the sequential data and has started to use them by developing representations in a very systematic manner, so that after the forced training, it developed representations of unseen numbers in just the same way as it did on the training examples.

I will end this section with the conclusion that the Recurrent Autoassociative Networks model can, indeed, develop distributed representations in a very systematic way, provided sufficient training data, which I showed by explaining how the RAN can account for three characteristics of systematicity: compositionality, combinativity and generalisation.

6.8 Conclusions

Sequential processing is recognised as a difficult problem, especially when sequential complexity, in terms of length and internal structure, increases. In the present chapter I proposed a framework for representing statically structured sequential data, as found in natural languages, perceptions of movements, actions, and so on. The approach is based on the idea that by sequential autoassociation, a single recurrent NN – the Recurrent Autoassociative Network – can develop static representations of sequences composed of uniform items. A hierarchical set, or a cascade of such networks, develops static distributed representations of ever more complex sequences, where each predefined structural level in the data is processed by one RAN module, and the sequential input to the upper levels is developed by the immediate lower level RAN. For this purpose, recurrent networks are trained on autoassociative tasks (RAN modules), and they develop unique static representations of the input sequences at their hidden layers (Fig. 6.3). Those static representations are used as interface patterns for the next level RAN (Fig. 6.6). The static representations at the highest level RAN are the distributed representations of the most complex data or whole input sequences, e.g., sentences or stories. In section 6.5, an example was given of how this model might work for developing representations of Dutch polysyllabic words, which includes grouping phonemes into syllables and syllables into words.

Further, it was suggested in section 6.6, how the cascaded RAN model might be extended to a more global cognitive model, where it was suggested that the static representations at each level were associated with or transformed into other static representations (of sequences of the same or other modalities) via static mappings. Such a net of multi-modal associations, I believe, might provide an implementation of natural language grounding and is a base for semantics modeling. Yet, the development of a full-scale autonomous cognitive system also involves many other unexplored sub-systems, such as signal control, attention, short-term and long-term explicit memory, etc.

The model has some practical limitations, too. Although I claim that the RANs can theoretically be used for representing hierarchically structured sequences of any complexity, there are practical problems that currently do not allow its full realisation. These may be due to the currently limited computational power and the absence of real neuro-chips that would allow a parallel implementation of large-scale artificial neural networks. Among the problems is also the not very efficient BPTT learning algorithm, which has the very difficult task to learn a multiple-pattern sequential autoassociation. One possibility to solve this problem is to improve the BPTT learning algorithm as suggested in (Kuan et al. 1994, Arai & Nakano 2000). Alternatively, other dynamic NNs, possibly more neurobiologically oriented, could be used, too, providing the possibility to organise a sequential auto-association in a manner similar to what was explained earlier in the chapter.

In spite of those limitations, I showed that the model contributes positively to the ever-going connectionist-symbolic debate, by demonstrating that it can develop meaningful distributed representations of sequences in a very systematic way. Those representations can be used for various high-level cognitive tasks, including holistic computations, which in turn will be the subject of the following chapter. Therefore, I believe the suggested model is an important step in connectionist modeling, and I strongly encourage the reader to experiment with the RAN cascade on different problems.