Chapter 9

RECURRENT AUTOASSOCIATIVE NETWORKS: DEVELOPING DISTRIBUTED REPRESENTATIONS OF HIERARCHICALLY STRUCTURED SEQUENCES BY AUTOASSOCIATION

Ivelin Stoianov

Department Alfa-Informatica
University of Groningen
The Netherlands

I. INTRODUCTION

In spite of the growing research on connectionist Natural Language Processing (NLP), there are still a number of challenges to be solved, for example, the development of proper linguistic representations. Natural language is a dynamic system with underlying hierarchical structure and sequential external appearance. Therefore, NLP systems need an adequate hierarchical system of linguistic representations. What we roughly distinguish as letters, words, sentences, etc., needs to be encoded in a proper and systematic manner, permitting direct, "*holistic*" operations over the resultant abstract representations rather than over external sequential forms [Chalmers, 1990, Blank, 1992, Hammerton, 1998]. Those representations should be static, unique characterizations of the original objects, which is necessary for reproducing them back into their sequential form. They should allow holistic transformations and associations to representations from other modalities – visual, effectual, etc. Natural language is not the only structured process where static representations at different levels are necessary for modeling: consider composite actions, dynamic visual processes and so on. We find other examples also outside of cognitive modeling, such as modeling economic processes and physical phenomena.

A widely used practice in connectionist natural language modeling is localistic and handcrafted feature based encoding [Seidenberg, 1989, Elman 1990, Plaut, 1996, Henderson, 1998], which restricts the capacity of the processing system. It would be preferable that those representations evolve in the course of experiencing the language in its external sequential form, which is in accordance with our capacity to learn any language without any prior knowledge of it. A first attempt to build such representations was suggested by Pollack, [1990]. He extended the static Multilayered Perceptron [Rumelhart, 1986] to the Recursive Auto Associative Memory (RAAM) model, which develops compact distributed representations of the static input patterns through an autoassociation. RAAM was further extended to a Sequential RAAM (SRAAM) for sequential processing. Different implementations of the latter model had variable success even when applied to trivial data [Chalmers, 1990, Blank, 1992, Blair, 1997, Kwasny, 1995, Hammerton, 1998].

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

In this chapter, I propose a novel connectionist architecture designed to build and process a hierarchical system of static distributed representations of complex sequential data. It follows upon the idea of building complex static representations of the input sequence, but has been extended with the ability to reproduce these static representations in their original form, by building unique representations for every input sequence. The model consists of sequential autoassociative modules – Recurrent Autoassociative Networks (RANs). Each of these modules learns to reproduce input sequences and, as a side effect, develops static distributed representations of the sequences. If requested, these modules unpack static representations into their original sequential form. The complete architecture for processing sequentially represented hierarchical input data consists of a cascade of RANs. The input tokens of a RAN module from any but the lowest level in this cascade scheme are the static representations that the RAN module from the lower level has produced. The input data of the lowest level RAN module are percepts from the external world. The output of a module from the lowest level can be associated with an effector. Then, given a static representation set to the RAN hidden layer, this effector would receive commands sequentially during the unpacking process.

RAN is a recurrent neural network which conforms to the dynamics of natural languages. Also, RANs produce representations of sequences and interpret them by unpacking back to their sequential form. The more extended architecture – a cascade of RANs – resembles the hierarchy in natural languages. Furthermore, given representative training environment, this architecture has the capacity to develop the 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 and Pylyshin [Fodor, 1988] that only classical symbolic systems can guarantee systematicity (see Aydede [1997] for review). Connectionist systems claimed to meet this challenge are the

RAAM and SRAAMs, and the Smolensky's tensor products [Smolensky, 1990] among others. Later in this chapter I will argue that RANs provide an account of systematicity, too. Therefore, I believe that the RAN and the RAN cascade can participate in a more global cognitive model, where the distributed representations they produce are extensively transformed and associated.

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

II. SEQUENCES, HIERARCHY, 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 another 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 are entities by themselves and consist of strings of tokens, but these sequences 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 a long sequence composed of basic tokens. For example, in natural languages, there are basic tokens – phonemes or letters; next, there are words consisting of sequences of letters or phonemes, sentences consisting of sequences of words, and so on (1). That is, in natural language, dynamic objects are part of other, more complex dynamic objects. Hierarchical objects naturally evolved during evolution are better suited to represent, process, and transmit information, than linear objects. Another advantage of this hierarchy is the redundancy among the linguistic objects, which makes the transmission of the information content in this sequence more reliable.

((J o h n) (l o v e s) (M a - r y)) (1)

Sequential data that have such composite structure might have very long external representations, that is, representations consisting of rows of lowest level tokens. For example, in natural languages average sentences have some 50 characters and the current chapter has more than 15,000 characters. This data is difficult to represent and process in this external form. It has structure and I believe we organize and remember the language we experience in accordance with this structure. In the natural languages, for example, there are mechanisms for referring to some substructures – e.g., definite markers and pronouns. When it comes to processing those structures as single entities, we prefer to use internal representations of those structures rather than their external forms. Definite noun phrases and pronouns are just the external expressions of those internal representations and we use them very often. Single internal representations are much more economical to use when associating linguistic expressions to visual objects, actions, etc. Those associations are made between the internal representations of those complex objects rather than between their external representations. Therefore, if we want to model a cognitive system dealing with such a variety of data, it should properly organize a system of such representations.

Similar systems of representations naturally occur in symbolic approaches when modeling cognition: the external *terminal* tokens are organized by 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 organization of the symbolic approach and its unlimited representational capacity, cognitive modeling should be based on the classical "language of thoughts" [Fodor, 1988], meaning the 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 representations (symbols) and objects in the environment where a cognitive agent exists [Searle, 1984, Dorffner, 1991]. Connectionist architectures are particularly good at associating low level data – percepts or effectors – to higher level processing systems. Another problem concerning the symbolic approach is related to the "hardness" of its rule-based logical computations [Smolensky, 1991]. Models should accommodate the "softness" of cognition, as connectionism does, by processing data in a more fuzzy, stochastic manner.

Nevertheless, connectionism is still looking for an answer to the question of how to organize the information coming from the external sensors. Some of the available connectionist solutions are still not satisfactory. Although the implementation of the sequential RAAM proposed by Kwasny and Kalman [Kwasny, 1995] was promised to provide a better solution than the standard sequential RAAM, Hammerton [Hammerton, 1998] found that in practice this model did not learn even trivial data well. Therefore, the door is still open for other more optimal solutions and, in section four of this chapter, I propose another model: a cascade of Recurrent Autoassociation Neural Network to build a system of such representations.

In the next section I will present some details about a few connectionist architectures for sequence processing, but before that, I will outline more explicitly some features that connectionist systems and distributed representations must meet.

First, these representations should develop *emergently*, in the course of experiencing the external training data. Different levels of representations should develop *consequently*, one after another, possibly refining the representations from the lower levels. The lowest level of representations should be *perceptual* in case of dealing with sensors or *effectual* when producing actions. This gradual representation development, together with latter associations to representations from other modalities, solves the *symbol grounding problem*. Multi-modal associations should develop in simultaneously processing different modalities, for example, linguistic and visual.

Next, in order to meet the cognitive modeling requirements [Fodor, 1988], these representations should have a kind of *structure* allowing a combinatorial syntax, not necessarily explicit as in the classical symbolic systems, but understandable for other connectionist modules. This requirement is necessary for emergence of *systematicity* among those representations and for *holistic* computations [Hammerton, 1998, Smolensky, 1991]. The latter are structural operators acting upon the whole complex static representations, rather than on the token parts of the structure (sequence). Systematicity is the key to higher-order cognitive processes.

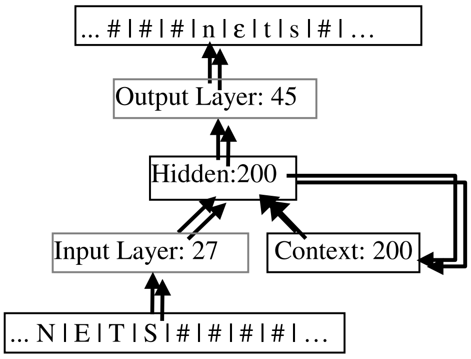
A failure to accommodate these features, when modeling language, results in models with limited capacities and no lasting implications, which is typical for many of the experiments reported that feature, for example, static language processing and hand-coded data encoding. Neither of those two popular approaches is in accordance with the spirit of natural language. We acquire language sequentially, hearing sounds and building or recognizing gradually a number of objects or temporal structures, such as, words, phrases and sentences. Similarly, when thinking, we produce and articulate language sequentially (possibly silently) by translating those structures back into temporal events, finishing this process by executing motor commands. 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 implies that those objects should develop constantly during the communication process, from the early childhood.

III. NEURAL NETWORKS AND SEQUENTIAL PROCESSING

A. ARCHITECTURES

For more that a decade, neural network research has been considered important not only because it makes efforts toward explaining our intelligence, but also because it provides effective working models for solving a wide range of practical problems. Numerous researchers (e.g., [Rumelhart, 1986, Grossberg, 1982, Kohonen, 1984, and Hopfield, 1982]) established the theoretical background in this field. The models they have developed – the Multilayer Perceptron trained with supervised Backpropagation learning algorithm, the ART & ART-Map, the self-organizing Kohonen Maps and the Hopfield Networks, correspondingly – were theoretically [Hornik, 1989] and experimentally [Lawrence, 1995] proven to be capable of solving many static tasks. Although the response of some of these models to the input pattern depends on their internal dynamics (Hopfield and Grossberg Neural Network models), they are not endowed with the capacity to process dynamic or sequential patterns. Hopfield NNs are designed to search iteratively for one of the encoded attractors. ART models develop localistically represented categories, which restricts their capacity when processing large variety of sequences, even with some special encoding schemes.

Problems such as language processing and robot control, which are essentially dynamic, pushed the connectionist investigations toward searching for NN models capable of handling such dynamic data. The first NN models were still static, encoding limited dynamics by means of a window, shifting over sequential data. The NETtalk model [Sejnowski, 1987] was trained to produce phonetic representations of words, where the context required to map the current letter was encoded within a shifting window of size seven – three letters on the left and three letters on the right side of the letter to be pronounced. This is an example of the so-called *F*inite *I*nput *R*esponse filter, where the system response to a given input is limited to a predefined number of steps.



... # | # | # | n | ɛ | t | s | # | ...
Output Layer: 45
Hidden:200
Input Layer: 27
Context: 200
... N | E | T | S | # | # | # | # | ...

Figure 1. Simple Recurrent Network that "reads" words, that is, maps orthographic to phonologic lexical representations.

The first real recurrent models were extensions of the Multilayer Perceptron with recurrent connections. They implement another type of dynamics – *I*nfinite *I*nput *R*esponse – where the input at a certain time influences the system response until the dynamics are externally reset. Several recurrent versions of the MLP were developed. In one of them [Jordan, 1986], the network state at any point at a time is a function of the input at the current time step, plus the state of the output units at the previous step. In another recurrent model proposed by Elman [Elman, 1990] – Simple Recurrent Networks (SRNs) – the network's current state depends on the current input and its own internal state, which is represented by the activation of the hidden units in the previous moment – see Fig.1. This internal state is considered as a context that provides information for the past. The SRNs were successfully employed for many linguistic and other tasks where the objects have sequential nature [Reilly, 1995, Wilson, 1996, Cairns, 1997, Stoianov, 1998, Stoianov, 1999]. Simple Recurrent Networks were initially trained by Elman with the standard backpropagation (BP) learning algorithm, in which errors are computed and weights are updated at each time step. While more biologically motivated because of the local-in-time weight adjustments, the BP is not as effective as the backpropagation through time (BPTT) learning algorithm, in which error signal is propagated back through time and temporal dependencies are learned better. A detailed technical description of the SRNs and the BPTT algorithm is presented in section four.

The static self-organizing Kohonen Map neural network was extended with recurrent connections too, which made the network responses dependent on both the current input and the last neural map activations. Models following this idea are the Temporal Kohonen Map (TKM) by Chappell and Taylor [Chappell, 1993] and the Self-Organizing Feature Map for Sequences (SARDNET) by James and Miikkulainen [James, 1995], among others. There are still other connectionist architectures able to process dynamic data, usually by inducing dynamics in existing static models with recurrent connections among neurons (*global memory*) or implementing dynamics in neurons (*local memory*). The latter types of architectures vary with regard to the place of the dynamics – in the weights, in the activation function, or both [Lawrence, 1995, Tsoi, 1994].

A common restriction on most of the recurrent models is that the input data they process has to be linear in the temporal dimension. These networks are able to recognize and classify the temporal sequences they have been trained on (SRNs and Jordan Networks) or they have clustered during the self-organization process (TKM and SARDNET), but they can not extract more complex temporal features or substructures. In addition, as the length of the sequences becomes greater, the performance worsens. This problem has been recognized by a number of authors. Bengio, Simard, and Frasconi [Bengio, 1994] showed that earning long-distance dependencies is difficult even for very simple tasks (long strings of a few basic symbols). Miikkulainen and Dyer [Miikkulainen, 1991] emphasized that the required network size, the number of training examples, and the training time become intractable as the sequence temporal complexity grows.

B. REPRESENTING NATURAL LANGUAGE

As I discussed earlier in the previous section, an important moment when dealing with sequences is the ability to develop a hierarchical structure of representations of the processed sequences. This question is especially apparent in natural language modeling. In earlier connectionist models, the lexemes were represented in a static manner with some artificial and not always effective encoding schemes (e.g., in Seidenberg [1989] and Plaut [1996]), and sentences consisted of some artificial and very limited in number words [Elman, 1990, Miikkulainen, 1991, Tabor, 1997]. When modeling some other problems, for example learning lexical phonotactics [Stoianov, 1998, Stoianov, 1999] or learning the mapping from orthographic to phonetic representations for certain language [Stoianov, 1999], one does not really need static representations of words, but in other cases, such as holistic computations or static associations, this is obligatory. To my knowledge, there are no approaches that model people's full capacity to deal with structured dynamic data using connectionism.[1] People associate the sounds they hear with visual patterns or actions. Similarly, they associate actions with words and sentences. All of these objects are realized as sequences of small parts, spanning time. In order to treat them as single entities, we shall enclose them and represent them statically. This naturally leads to a search for methods packing sequences into static representations and unfolding them back into their sequential form.

High-level connectionist language modeling has focused on small illustrative problems: as the learning is restricted to simple grammars and a very limited number of words [Elman, 1990, Tabor, 1997] or word TAGs [Henderson, 1998]. I attribute this not to the connectionist models' inability to learn complex dependencies, but rather to the absence of adequate concepts of how to develop high-level distributed representations. Thus far, representations have been designed mostly by hand, either feature based or even simpler – localistically. This works well at the bottom stages of language processing, where a very limited number of characters and phonemes constitute words, but not at higher linguistic levels, where large number of words, phrases, sentences build more complex structures. The variety at those levels is enormous. How can representations be developed for such objects? Hand-crafting here simply does not work. Some cognitive scientists suggest "tensor products" [Smolensky, 1990], which expand with the increase of the data complexity. Others suggest syntactic structures to be represented both in time and space (Temporal Synchrony Variable Binding [Henderson, 1998], but these solutions do not produce fixed-sized static representations for input objects of variable complexity. Therefore, others support the more plausible idea of static, fixed-size, emergently developed connectionist "symbols." But how can such representations be developed?

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 MLP. The RAAMs auto-associate the input data, which is a concatenation of two patterns, and use the activations of the hidden layer neurons to represent this concatenation (Fig. 2a). This is equivalent to the learning of a simple symbolic rule. When applied recursively, that is, when using the developed representations as an input for another compression, 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 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, 1995].

RAAMs were theoretically extended to model a stack, which made the model capable of learning and representing sequences – Sequential RAAM (SRAAM). However, this model needs an external stack during the training, which is a step

[1] See the following discussion on the (S)RAAM connectionist models.

back from connectionism, as commented earlier. It is difficult for SRAAM to learn even trivial structures and sequences, which makes it an impractical model. Also, 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 certain level should be produced only if there is a necessity of using 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 combination of mixed items is not so useful.

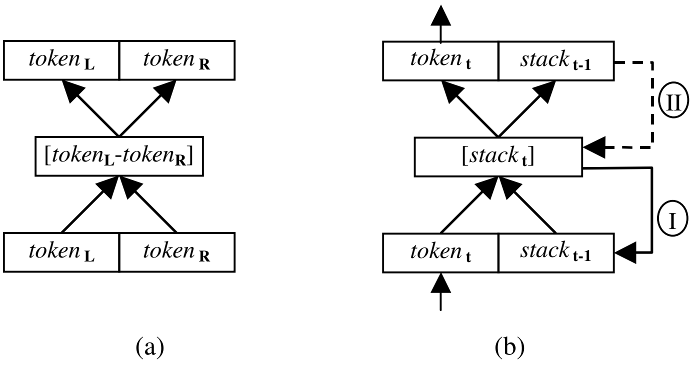


Figure 2. (a) RAAM: left and right input tokens are autoassociated, which results in a single compact representation of the input tokens at the hidden layer. (b) SRAAM: based on RAAM and SRN. (I) *Compression*: tokens apply one at a time to the input and they are autoassociated, together with the previous state of the context. This results in ever more compact representation of the input sequence at the hidden layer. (II) *Decompression*: 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.

Another implementation of the sequential RAAM was presented by Kwasny and Kalman [Kwasny, 1995]. Their SRAAM combines the architecture of the Simple Recurrent Networks and the RAAM idea for autoassociation (Fig. 2b). 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 resulted neuron activations. When combined with a modified conjugate gradient training algorithm, this reportedly improved the learning. And 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 [Hammerton, 1998] attempted to recreate the Chalmers [Chalmers, 1990] experiments using the Kwasny and Kalman SRAAMs, which was promised to learn faster and more reliably. For this purpose, he used the corpus from Chalmers, [1990] – a small corpus containing 250 sequences built out of 13 distinct items. Standard backpropagation learning algorithm and two variations of the Kwasny and Kalman training algorithm were utilized. Hammerton reported that, with the best learning algorithm (the noted earlier modified error function 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 (Hammerton, p. 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 those 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 organization, the input sequences are reproduced inversely. Therefore, one would need another external mechanism to reverse those sequences. The solution I propose in the next section is based on an autoassociation and SRN, too, but it implements a *queue* rather than a *stack* mechanism, which leads to reproducing the sequences in the right order.

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 and McClelland [St. John, 1990] – the Story Gestalt Model. This model comprises two networks – a Jordan Recurrent Network which 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 *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. 3). Another similar model, the Movie Description Network by Cottrell, Bartell, and Haupt [Cottrell, 1990] uses the 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.

The capacity of SRN to process sequential data was exploited in another approach, aimed 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

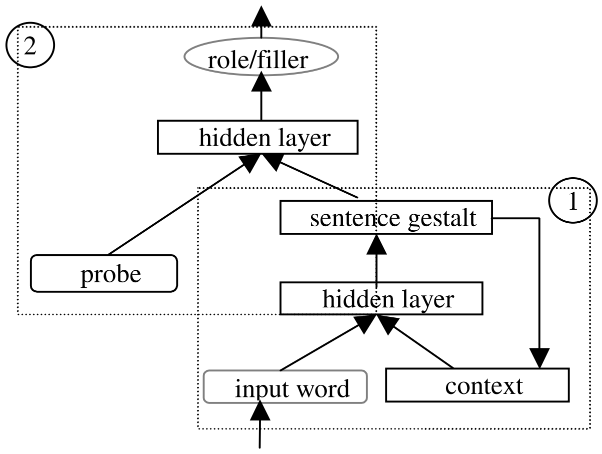


Figure 3. Story Gestalt Model by St. John. In this architecture, a Jordan recurrent network (1) gradually produces representations of the input sequence. Next, a MLP (2) extracts from these representations some of the constituents of the sentence.

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. Having originally the task of word recognition, this scheme requires that both phones and lexical representations be offered during training. Static syllable representations are resulted as a side effect. This approach takes a direction which is opposite to the gradual building of language representations. Instead of building word representation, it does the opposite – it produces syllable representations in the course of word recognition.

The approach presented here takes another direction, consistent with the principle of gradual language evolution and learning, by processing and evolving language items of increasing complexity. Another problem with the solution presented by Gasser is that because the packing and unpacking processes are split, this method requires the training sequences during both learning tasks, which is less plausible and increases the learning time. The sequential autoassociative task in my approach requires only a short-term memory to keep the sequence to be presented for a second time at the output layer and the system only has to perceive the input the environment provides throughout the learning (which may last indefinitely: we can keep the learning going non-stop).

IV. 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 Networks [Elman, 1990]. Recall that SRN is a recurrent neural network architecture based on the feedforward Multi Layered Perceptron with a global context memory storing the recent activation of the SRN hidden layer,[2] which is fed back as an additional input to the hidden layer itself (Fig. 1). The context layer has the capacity to encode all the information for the input provided to the network, since the beginning of the sequence. Hence, if we reset the context layer and apply a sequence, at every moment the hidden layer and the context layer (after a short delay) will contain static distributed information for this sequence, which as we will see later is not necessarily a representation of the input sequence, but rather depends on the learning task.

There are different possibilities to obtain static representations of the input sequence. One of them is just to use the context layer activation after the whole sequence has been processed in an item prediction task – similarly to the *gestalt* models by St. John and McClelland [St. John, 1990] and Cottrell, Bartell, and Haupt [Cottrell, 1990]. The networks there were trained to predict the next input token, which forces the networks to learn information specific to this particular task, but is insufficient for developing complete representations of the input sequences. We need an organization or a learning task that guarantees that the distributed representations developed by the network (1) contain all information about the sequence, (2) are unique for each sequence, and (3) contain enough information to reproduce the sequence, which is a consequence of (1) and (2).

Representations satisfying the above requirements evolve naturally, if we train the network on an autoassociation task, that is, to reproduce the input sequence. But then, 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, but this will not produce any useful hidden layer representations – there will be no need of a context for this task. A delay of one step, or two, or some other fixed number of steps would train the network to develop information specific for the prediction task, but the representations would still not necessarily satisfy the conditions 1 – 3. In order to produce such representations, the network has to be trained on an autoassociative task in which the input sequence starts to be reproduced after the whole input sequence has been represented to the input, followed by a unique pattern, a *trigger*, indicating the end of the sequence. This way, if the training set contains sequences, which in turn contain another sequence in the training corpus as an initial sub-sequence, then the network will still produce distinct representations for both sequences. The static representation of the sequence will be just the

[2] The term "layer activation" denotes a numerical vector with the activations of all neurons in that layer.

hidden layer activations at the moment when the trigger has been applied and processed by the network (Fig. 4).

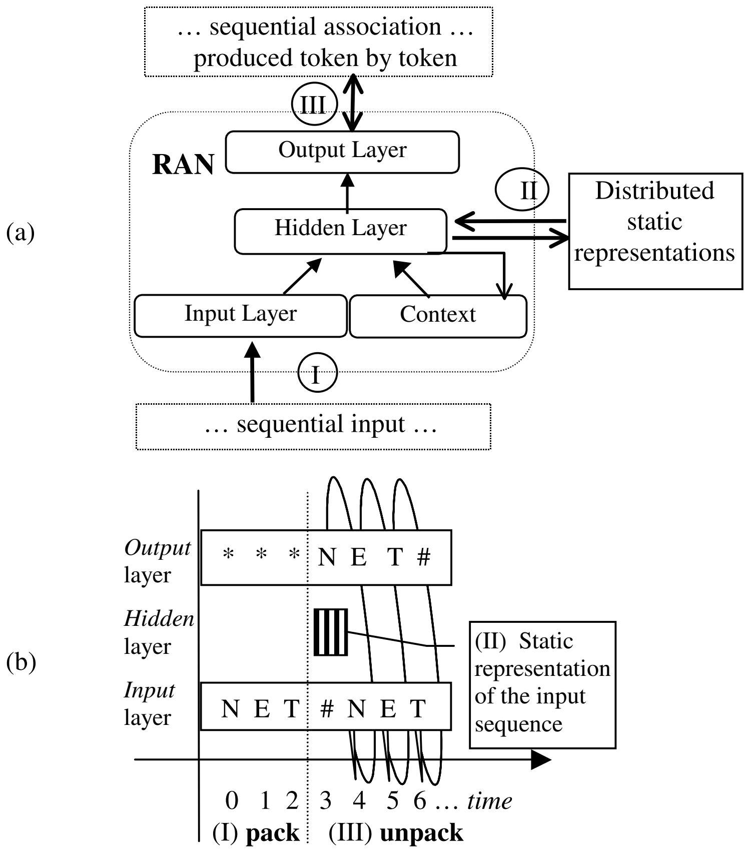


Figure 4. Recurrent Autoassociative Network: (a) architecture and (b) functional temporal unfolding. Operations: (I) input sequence packing (time steps 0–3), (II) obtaining or setting a static representation (time step 3), and (III) unpacking the static representation to its sequential form (time steps 3-6).

There are some other details to be specified, namely the input pattern after the trigger has been applied to the input layer, the target output pattern 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, labeled *end-of-sequence*, which will signal that the whole sequence has been produced. The input patterns after the trigger might be either the same trigger, repeatedly provided until the network produces the end-of-sequence pattern, or, what I found more helpful to the network in learning this difficult task, the last ouput pattern provided to the inner larger. The latter approach provides guiding information about how far the network has progressed in reproducing the sequences, and experiments have demonstrated that, indeed, it is easier for the network to learn the task with this approach.

With regard to the output target patterns at the time when the sequence is still being provided to the input, the learning algorithm used – backpropagation through time [Haykin, 1994] – 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, which originates in the second phase of the autoassociation (Fig. 4, step III).

Another possibility is to train the network at the same time on a prediction task. First the network produces its anticipations about the coming patterns. Next, it reproduces the whole sequence using no external input, but only what the network has encoded at the context layer, perhaps reusing the produced output pattern as input. The latter approach is more plausible considering what each of us has observed when listening to a speech in a noisy environment, that we use anticipations in interpreting that speech. In such a noisy environment, having received some initial context, the network can produce the most probable sequence for this left context and next to produce the static representation for this sequence by presenting the triggering pattern. 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 in the second approach has to learn two tasks, which makes the learning harder. When using the first approach, the network learns the autoassociative task faster and with fewer errors. The first approach satisfies the above outlined requirements, concerning the representations produced, and therefore it was used in the experiments reported on here.

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 static distributed representations of these sequences. The hidden layer activations, at the moment the triggering pattern is 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. 4: steps I and II) and secondly, to reproduce the original sequential form of a static representation, when the hidden layer is set to the static representations (Fig. 4: steps II, III).

A. TRAINING RAN WITH THE BACKPROPOGATION THROUGH TIME LEARNING ALGORITHM

The RAN (Fig. 4) and the SRN (Fig. 1) models have the same architecture and share similar feedforward processing steps and learning algorithms. However, when SRNs are trained on prediction task they can also be trained with the standard error backpropagation algorithm, while RANs can only be trained with backpropagation through time learning algorithm (Haykin 1994), because in the first feedforward processing phase (Fig. 4, phase I), there is no error signal originating at the output layer and backpropagated through the network, but only error signal computed in the second phase (Fig.4, phase III) and backpropagated through time. With regard to this, any other learning algorithm that backpropagates error signal through time can be used, too. What is important, when using BPTT to train RANs, is to set properly the sequence of input-output training data, according to the scheme outlined earlier in this section. The BPTT learning algorithm is generally known to the potential readership, but for completeness, it will be described in this subsection.

SRNs have two working regimens – a utilization of a trained network and network training. The first one is simply applying a forward pass, where the current input signal is propagated forward throughout the network and the current context layer activation is used. After each forward step, the hidden layer activation is copied to the context layer, to be used later. The network utilization is the same as the forward step in the BPTT, which in turn is described in the next section. The BPTT learning algorithm itself is more complicated. It includes: firstly, a forward pass (2-5) for all input tokens, keeping the network activations in a stack; secondly, a backward through time pass (6-9), where the errors are computed at the output layer and backpropagated through the network layers and through time, and thirdly, updating the weights with the accumulated weight-updating values (10). During the second step, at each time moment but the last one, a *future error* is used, processed and backpropagated further through time.

1. Forward Pass

In the following description, $|IL|$, $|HL|$, $|CL|$, $|OL|$ stand for the size of the input, hidden, context and output layers, correspondingly. The input signals provided to the hidden layer neurons and output layer neurons are noted as $net^H_i(t)$ and $net^O_l(t)$. Next, $in_j(t)$, $cn_k(t)$, $hn_i(t)$ and $on_l(t)$ stand for the activations of the j-th input, k-th context, i-th hidden and l-th output neurons at time t. And finally, w^H_{ij}, w^H_{ik}, and w^O_{li} are the weights of the connections between j-th input neuron and i-th hidden neuron, k-th context neuron and i-th hidden neuron, and i-th hidden neuron and l-th output neuron, respectively. For convenience, the bias for all layers is encoded as an extra input neuron (j=0; i=0) with constant activation 1. The activation function $f(.)$ is of sigmoidal type – the logistic function or the hyperbolic tangent function.

The items of the training/testing sequence $S=[c_1c_2\ldots c_{|S|}]$ are referenced with an index t, set to zero in the beginning of the sequence processing. Also, before applying a new sequence, the context layer is reset by setting all context neurons $cn_k(t=0)$ to zero ($k=1\ldots|CL|$). The sequences are presented to the network one token c_t at a time. For each token, a forward pass is processed. Firstly, the hidden layer is activated in accordance with (2) and (3):

$$net^H_i(t) = \Sigma_{j=0\ldots|IL|} w^H_{ij}\, in_j(t) + \Sigma_{k=1\ldots|CL|} w^H_{ik}\, cn_k(t) \qquad (2)$$
$$hn_i(t) = f(net^H_i(t)) \qquad (3)$$

After the activation of the hidden neurons, their activation values are copied to the context neurons. Next, the signal is propagated further to the output layer, by activating all neurons at the output layer: (4,5).

$$net^{O}_{l}(t) = \Sigma_{i=0\ldots|HL|}\, w^{O}_{li}\, hn_{i}(t) \qquad (4)$$
$$on_{l}(t) = f(net^{O}_{l}(t)) \qquad (5)$$

Specifically for the RANs, if the forward pass is a part of the BPTT learning algorithm, a training sequence S of input/output patterns according the scheme in Fig. 4b is built. On another hand, if the network is used for packing only, the static distributed representation of a sequence being applied to the input layer is the activation of the hidden layer at the moment when the delimiter pattern has been processed. In turn, in order to unpack (decode) the static representation of a sequence, it is applied to the hidden layer and propagated forward; the resulted output pattern is the first element of the sequence. Next, this output pattern is provided as an input to the network and propagated forward, by using the last hidden layer activation as a context. This process is repeated until a pattern recognized as a delimiter pattern is produced at the output layer.

2. Backward Through Time Pass

The second step of the BPTT learning algorithm for a given training sequence is propagating the error signal back through the network and time. We suppose that the forward steps for each token in the sequence are already done, keeping the activations and the target patterns in a stack. Next, error and weight updating values are computed in an earlier time cycle, that is, starting from the last token. Firstly, error deltas at the output layer and the updates of the weights connecting the hidden layer to output layer are computed with (6) and (7). Note, that S stands for the whole training sequence (the original sequence presented at the input layer and targeted later at the output layer). Also, τ denotes a global time index and $\Delta w(\tau)$ stands for the accumulated $\Delta w(t)$ for all items from the current sequence.

$$\delta^{O}_{l}(t) = f'(net^{O}_{l}(t))\,(C_{l}(t) - on_{l}(t)) \qquad (6)$$
$$\Delta w^{O}_{li}(\tau) = \eta.\, \Sigma_{t=1\ldots|S|}\, \delta^{O}_{l}(t)\, hn_{i}(t) \qquad (7)$$

In (6), $C_l(t)$ denotes the desired activation of the l-th output neuron (l=1…|OL|) at time t. Provided that the activation function f(x) is the logistic function $f(x) = (1+e(-x))^{-1}$, the derivative of $f(x)$ is $f'(x) = x(1-x)$. Next, deltas and updating values of the weights connecting the hidden layer to the input and the context layers are computed in accordance to (8) and (9):

$$\delta^{H}_{i}(t) = f'(net^{H}_{i}(t))\,[\,\Sigma_{l=1\ldots|OL|} w^{O}_{li}\delta^{O}_{l}(t) + \Sigma_{k=1\ldots|CL|} w^{H}_{ik}\delta^{H}_{k}(t+1)\,] \qquad (8)$$
$$\Delta w^{H}_{ij}(\tau) = \eta\ \Sigma_{t=1\ldots|S|}\, \delta^{H}_{i}(t)\, n_{j}(t-1) \qquad (9)$$

where $i = 1 \ldots |HL|$, $j = 0 \ldots (|IL| + |CL|)$ and $n(t)$ is a joined vector containing both $in(t)$ and $cn(t)$. The second sum in (8) represents the context layer delta-term $\delta_k^C(t)$, computed by backpropagating the delta $\delta^H_i(t+1)$ through the weights connecting the context neurons to the hidden neurons. And finally, all weights are updated according to (10) with the accumulated weight-updating values, computed with (7) and (9).

$$w(\tau) = w(\tau\text{-}1) + \Delta w(\tau) \tag{10}$$

Error Back propagation learning algorithms are known for the possibility of getting stocked in a local minima on the error surface. There are number of techniques designed to overcome this problem. The most useful technique is to apply a momentum term α to (10), as is done in (11). The momentum term keeps the movement over the weight error space for some time, even if the network has fallen into a local minimum. Usually, $\alpha = 0.7$.

$$\Delta w'(\tau) = \alpha\, \Delta w(\tau\text{-}1) + (1\text{-}\ \alpha)\, \Delta w(\tau) \tag{11}$$

Another technique that has similar effect is to apply initially a higher learning coefficient η and, next, to decrease it gradually. This implements a quicker rough search for the region where the global minimum is located. Later, the exact location of the error minimum is searched with smaller steps. Usually, the initial $\eta = 0.2$ and the decrease might be exponential with a very small step (e.g., 0.9995). For further reading about SRN, BP, and BPTT and other recurrent learning algorithms, one can refer to Haykin [1994].

B. EXPERIMENTING WITH RANs: LEARNING SYLLABLES

The idea of using RAN to develop static representations of sequences was tested on natural language data. A set of 140 distinct syllables was collected from a list of 100 polysyllabic Dutch words. The syllables were represented as sequences of Latin characters. The mean length of the syllables was 4.1 ± 1.12 $\boldsymbol{\sigma}$. The characters were represented 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 27th 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 respectively, which set the working regimen of the neuron activation functions within an 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 learning algorithm was backpropagation through time. The training was organized in epochs, in which all patterns were presented randomly, according to their frequency of occurrence in the corpus. The words were taken from the

CELEX[3] lexical data base. The network error was measured after each training epoch as percent character and syllable misprediction. During the training process, the network error followed the standard pattern of quick initial error drop and subsequent slow rate of decrease. After approximately 50 epochs, the network error was reduced to 1%. Further training would reduce the error even more, but it would take much more time.

From an implemental point of view, it was interesting to test different strategies for representing the trigger and 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 three cases, the performance was worse than the approach with an extra, switching neuron. Therefore, the later 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 patterns are used to represent data.

This first experiment suggests that SRNs can learn such a task. Now, it is interesting to see what kind of static representations the network has developed after the training. A simple observation of those vectors does not say much (Fig. 5, top), because the network has organized those 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 form. Yet, some analysis might be useful in order to persuade the reader that those representations are worth something. For this purpose, a Kohonen Map neural network was trained to organize those representations. The Kohonen Map is known as very useful for clustering such a data. The resultant map and a minimal spanning tree are given in Figure 5, at the bottom.

As expected, syllables with same front parts are located at similar positions (e.g., *ant*, *ann*, *aus*, *am*, *aan*, *a*), but also, syllables with similar ends are placed at close positions (e.g., *pol* - *rol*; *tra* - *dra*). The RAN very clearly has captured the common external features among the training sequences: similar sequences are mapped into close positions; that is, their distributed representations are close. This raised expectations that the network would generalize, that is, reproduce unseen input sequences (and produce their static representations). This hypothesis was tested and the results demonstrated that RAN did generalize, although very modestly. Among the tested sequences, unseen during the training, only 15 syllables were successfully reproduced at the output, that is, about 10% generalization. But the network was trained on a very small number of sequences, therefore, I predicted that if the network were trained on a larger data set, the percent of the generalization would be even larger. The second experiment in section five confirmed this hypothesis.

[3] The CELEX lexical database contains lexical data for Dutch, German, and English languages. Address: Center for Lexical Information, POBox 310, 6500 AH Hijmegen, The Netherlands, http://www.kun.nl/celex/ Email: celex@mpi.nl

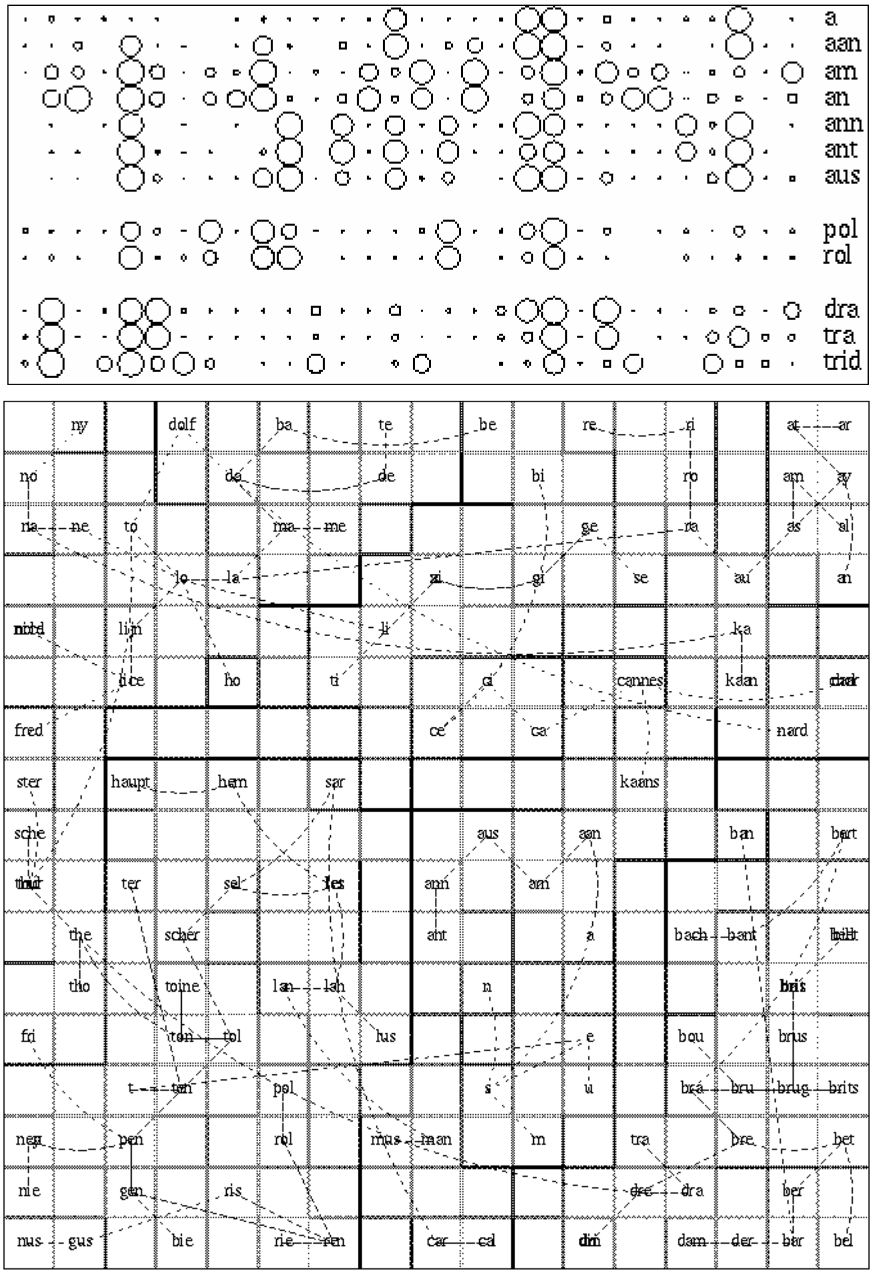


Figure 5. (top): Representations developed by the RAN. Each line stands for one vector. Each circle represents one value in the representation. The larger the circles, the greater the correspondent values. (bottom): A Kohonen Map neural network trained to cluster those representations (see text). The network maps similar syllables into close positions, meaning that those representations are close to each other. The lines connecting cells represent a minimal spanning tree.

V. A CASCADE OF RANs

The main goal of this work, as stated earlier in the chapter, is to build a connectionist model that develops static distributed representations of a set of hierarchically structured sequences. Single Recurrent Autoassociative Networks produce single-level static representations only. A natural development of RANs to cope with hierarchically structured sequential data is to build a cascade of RANs, in which each RAN deals with subsequences of the external patterns at certain level. That is, the RANs from each level are fed with sequences of patterns developed at lower-level RAN, produce static representations of those sequences, and provide them as input patterns to the following-level RANs. Also, whenever they are requested, those RAN modules will decode (unpack) representations, for example, if the following-level RAN module needs to decode some object into its sequential representation (Fig. 6).

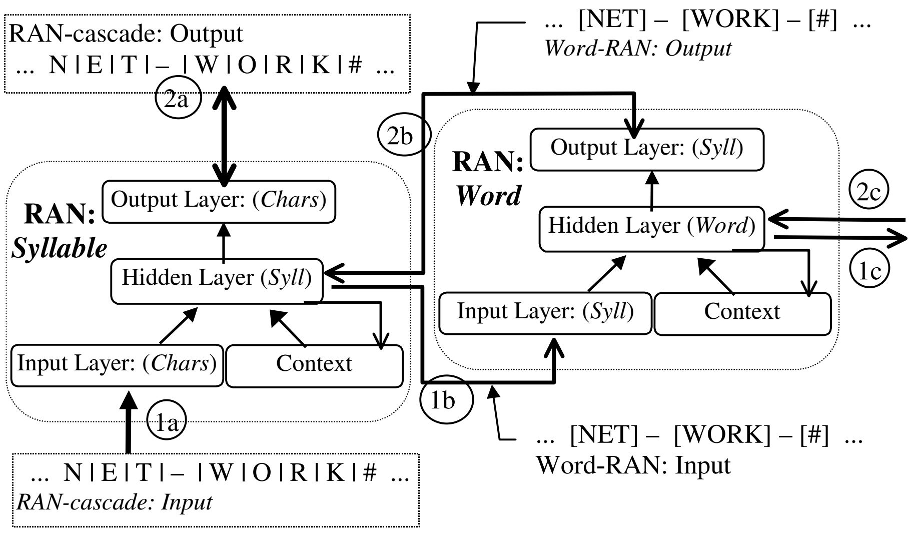


Figure 6. RANs and the mechanism of multi-level sequence processing. Stream of chars is presented to the input of the *syllable*-RAN (1a), which builds *syllable*-representations and provides them to the *word*-RAN (1b), which, in turn, builds *word*-representations and exports them further (1c). Similarly, if *word*-RAN is presented with a *word*-representation (2c), it will unpack it to train of *syllable*-representations and will provide them to the *syllable*-RAN (2b), which in turn will unpack them to train of chars (2a).

Following the representations developed from the lowest to the higher level RAN, note that this cascade model gradually transforms the temporal-dimension complexity into a spatial-dimension complexity, that is, long sequences of patterns of simple elements will be transformed into shorter sequences of complex static representations, distributed among an increasing number of neurons. This way, trains of percepts that implicitly contain high-level, sequentially represented concepts will be transformed into static representations of those concepts and these representations will also be reconstructed back to their external sequential form. This is important, as we will see in the next section, for solving the *symbol grounding* problem. RANs also provide an account of another important property – *systematicity* among the representations built. The more sequences learned at a certain level, the larger the network generalization will be; that is, after exploring many combinatorial possibilities among the input data, the RAN modules will build static representations in a systematic manner.

Still, there are some questions to be answered. For instance, why do we need such 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 favor of the cascade structure. First, it is difficult for one homogeneous network to learn long-distance relationships [Miikkulainen, 1991] for discussion [Christiansen, 1999]. BPTT learning algorithm propagates back error, 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 the depth of embedding (or recursion) SRNs can handle well is in the range 3 – 5 steps. Hence, learning sequences longer than 5 tokens would be more difficult. Even if one uses some techniques to improve the learning, the general tendency to perform worse on longer sequences remains.

Another advantage of the hierarchical system of representations is the possibility to access intermediate representations of the input train of simple elements (e.g., words), 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 a RAN cascade automatically develops such static representations.

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. Also, there should be a segmentation mechanism that signals the end of the sub-sequences for every cascade level.

Different levels may be selected according to the natural hierarchy in the input data. For example, when learning natural language, one might favor 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, Shillcock, Chater, and Levy [Cairns, 1997] 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. Still, recursive structures can be approximated up to a certain depth. Next, leaves (terminal patterns) may occur only at the bottom level of these trees. For example, sequence (12a) with internal structure (12b) is illegal in a two-level cascade because the token '*d*' is at the second level rather than at the bottom level. The allowed structures of this sequence are given in (12c). This limitation corresponds to the gradual way of information entrance and processing in the cognitive systems, starting from the bottom, perceptual level.

a b c d (12a)
((*abc*) *d*) (12b)
((*a*) (*bcd*)) ((*ab*) (*cd*)) ((*abc*) (*d*)) (*abcd*) (12c)

In case we need to develop representations of any tree structures, such as (12b), those tree structures can be transformed into plain sequences as in Kwasny, [1995]. However, this method needs an external symbolic device that transforms structures in both directions. Another solution might involve marking the distributed representations with level-labels and providing them to the correspondent RANs. This solution needs a supervisor that distributes the patterns, and it is biologically more plausible than the first solution.

Sequences with complex structure, such as sentences in natural language, may be processed with one RAN-level, too. The sentence structure that is known as syntax may simply evolve among the distributed representations, instead of being taught explicitly. Similarly, Elman [Elman, 1990] trained the SRNs on a prediction task and later found that the developed context layer representations correspond at each time step to the syntactic category of the sequence processed thus far.

The size h of the hidden layers at each level should be based on the informational content of the sequences the correspondent RAN is going to pack. Parameters in this measure are the number of distinct tokens $|C|$, the number of possible sequences $|S|$, the maximal length of the sequences to be represented k, and the number of 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 maximal 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, … k items selected from $|C|$ items. 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 generalize 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 when autoassociating the input sequence (see section 4). Therefore, the maximal number P_{max} of distinct patterns necessary to produce unique representations of strings composed of up to k tokens is $[(|C|^{k+1} - 1) / (|C| - 1) - 1](2k + 1)$. The actual number of necessary distinct patterns P satisfies condition (13).

$$|S|\,(2k+1) \leq P \leq [\,(|C|^{k+1} - 1)\,/(|C| - 1) - 1\,]\,(2k+1) \qquad (13)$$

Next, the number N of patterns that h neurons can represent, each of which having b distinct states is $N=b^h$. That is, the number of neurons necessary to represent N patterns is $h=log_b(N)$. Therefore, the number of hidden neurons necessary to encode P distinct patterns is

$$log_b(|S|\,(2k+1)) \leq h \leq log_b([\,(|C|^{k+1} - 1)\,/(|C| - 1) - 1\,]\,(2k+1)\,) \qquad (14)$$

from which we derive :

$$log_b(|S|)+log_b\,(2k+1) \leq h < (k+1)log_b(|C|) - log_b(|C|-1)+log_b\,(2k+1) \qquad (15)$$

Formula (15) estimates the minimal number of neurons that is necessary to <u>represent</u> a certain number of sequences with RAN. As mentioned earlier, the right-hand side number should be used in order to achieve better generalization. However, formula (15) does not guarantee that this number of hidden neurons is enough for the network to <u>perform</u> the autoassociation task for all strings and develop their distributed representations. In a recurrent neural network, hidden layer neurons also have other functions, 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 the theoretical estimations. Therefore, a scaling coefficient $\gamma>1$ will be applied to (15) 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.

Now, let us find an estimation of the necessary RAN hidden layer size in the previous example by using (15). The base b will be set to 2 states and the coefficient γ to 5.0. The 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 (15), $50 < h_{syll\_RAN} < 110$. In the reported experiments, 30 hidden neurons were enough to encode almost all training sequences.

Finally, let me discuss the strategy of 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 in 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 developed representations. Another 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 behavior 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 of a lot of 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.

A. SIMULATION WITH A CASCADE OF RANs: REPRESENTING POLYSYLLABIC WORDS.

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, which builds static representations of syllables, and a word-RAN, which builds static representations of words. In this subsection I will present an experiment which is a natural extension of the experiment described in the previous section. The syllable-RAN is be the same as before – with 27 input and output neurons and 30 hidden neurons. This means that the word-RAN input and output layers have to have 31 neurons, the last one standing for representing the trigger and end-of-word patterns. The size of the hidden layer is again 30, which is determined by the complexity of the concrete learning task – there are only 100 sequences to be learned, consisting of some 140 possible syllables, with average length of the input sequence 4 syllables. In a more complicated case, we would need many more hidden neurons (see the previous subsection).

The network training (BPTT) is organized as follows: First, a training word is selected from the training corpus, containing pre-syllabified words. Next, for each syllable in the selected word, the syllable-RAN produces the correspondent 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 the syllabic patterns are presented to the input of the second RAN, a triggering pattern is provided to the word-RAN and the processed syllabic patterns 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. 4 for details). Next, during the second phase of 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.

The cascade is tested by encoding the training or testing words and decoding (unfolding) the developed static representations of those words back to sequential forms (string of letters), and comparing the resulting strings with the expected strings. The RAN error is measured as the percent of erroneous predictions of letters, syllables, and words. Syllables are considered to be predicted correctly if all the correspondent letters are reproduced correctly.

Similarly, words are learned if all corresponded syllables are reproduced correctly. The performance of the word-RAN after 100 training epochs was as follows: 1.8% character error, 4% syllable error, and 6% word error.

B. A MORE REALISTIC EXPERIMENT: LOOKING FOR SYSTEMATICITY

In this subsection a more realistic experiment will be presented – building the representations of some 850 polysyllabic Dutch words, consisting of about 600 distinct syllables. The reason less complex examples were presented in the earlier sections was, firstly for the reader to get an idea of the developed static representations (Fig. 5) and, more importantly, to show that generalization increases by increasing the number of combinations learned by the networks.

We will use again (15) to estimate the necessary hidden layer size. For the syllable-RAN, $|C|=26$, $|S|=600$, $k=5$. Then $70 < h_{syll\_RAN} < 150$. Similarly, for the word-RAN, $|C|=600$, $|S|=850$, $k=5$. Then, $100 < h_{word\_RAN} < 300$.

The cascade consists again of two RANs – a syllable RAN and a word RAN. The syllable-RAN has 100 hidden neurons. The word-RAN is set to 350 hidden neurons. All other conditions are the same as in the first experiment. The training of the syllable-RAN resulted in 0.6% erroneous letter prediction and 2.5% erroneous syllable prediction, that is, some 15 syllables were not entirely learned. Further error analysis showed that there was one mispredicted letter among those syllables, which means that those syllables were produced almost correctly (3/4). The word-RAN did not reach the success of the syllable-RAN, with 2.7% letter misprediction, 5.0% syllable misprediction and 14.1% word misprediction.

In order to examine the influence of the hidden layer size to the performance, similar experiments were conducted with smaller hidden layers. The syllable-RAN was tested with 50 and 30 hidden neurons. Word-RANs were tested with 300 and 250 hidden neurons, using the earlier reported syllable-RAN with 100 hidden neurons as syllabic pattern builder. With decreasing the hidden layer size, the performance of both networks gradually dropped. The syllable-RANs learned 70% and 28% training syllables, correspondingly (Table 1). The word-RAN performance decreased too: 80.8% and 76.9% of the words were entirely learned (Table 2). The performance measured at item-level decreased more gradually. For the word-RAN, fewer syllables were erroneously reproduced and even less letters were mistaken.

Table 1. The performance of the *Syllable*-RAN trained on 600 distinct syllables, when varying the hidden layer size.

Syllable RAN	Error (%)	Hidden layer size		
		30	50	100
Mispre-dicted	Syllables	72	30	2.9
	Chars	18	7.2	1.1

Those experiments support to some extent formula (15) that is based on the information content of the hidden layer and number of data to be encoded. With decreasing the number of hidden neurons below the suggested size, the performance deteriorates. Nevertheless, there is another reason for this. The error backpropagation learning algorithm more easily finds escape routes from local minima when there are more weights – if some set of the weights are trapped into a valley on the multi-dimensional error surface, other weights would let the network drive out of this point. Therefore, the more complex the task, 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 will not be a question in a few more years.

Table 2. Performance of the *word*-RAN trained on 850 polysyllabic words with input vector size 100, when varying the hidden layer size.

Word RAN	Error (%)	Hidden layer size		
		250	300	350
Mispre-dicted	Words	23.1	19.2	14.1
	Syllables	7.8	6.8	5.0
	Chars	4.2	3.6	2.7

The more interesting question now concerns the generalization of the syllable-RAN and the word-RANs. Tested on a larger corpus with 9,000 words and 2,320 distinct syllables, the syllable-RAN successfully reproduced, that is, generated unique representations of another 1150 syllables, which is more than 190% generalization as opposed to the first example with only 10% generalization. This result shows that a network trained with more combinatorial possibilities generalizes better. In turn, this shows the RAN capacity to produce static distributed representations *systematically* (see section 7 for discussion).

The word-RAN generalized well too, with successful reproduction of 1,500 words unseen during the training, which is about 180% generalization. It is interesting to note that the word-RAN generalized 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 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, etc.), perhaps the syllable-RAN would learn the task even more easily, with fewer hidden neurons, and would generalize better.

VI. GOING FURTHER TO A COGNITIVE MODEL

Once we have a method to represent the complex structured data that we experience externally in some dynamic (sequential) form, we can go further and learn some relations between representations coming from different modalities. For example, the auditory modality would produce representations of linguistic objects, just as we discussed earlier in the chapter; visual modality is a source for even more complex objects. In addition to those sensory modalities, there are effector modalities – muscles, glands, and so on. Having representations of objects of those modalities, we can make associations between them. And those associations are the sources for representation *grounding* – multi-modal associations.

Complex multi-modal mappings can be effected with any static connectionist model – self-organizing or trained by a teacher. And both would be biologically motivated, because those associations can be made whenever two representations occur in the same time and there is a will or attention to learn those coincidences. Neural Network models that might be used for this purpose are the supervised Multilayered Perceptron, the ART-Map network by Carpenter and Grossberg [Carpenter, 1992] or the autoassociative memory by John Hopfield [Hopfield, 1982], among other connectionist models.

In the framework of RAN cascades, sequential patterns from each modality have to be divided into different conceptual levels and correspondent RANs have to be trained to produce static representations. Next, static associations (mappings) between patterns from different modalities have to be learned with static neural networks (Fig. 7). Then, when a sequence is applied as input to a learned modality, that is, to its corresponding lowest-level RANs, higher level representations will be produced. This in turn will activate corresponding patterns in other modalities, which might be expanded to lower level sequences. Activation of a high-level representation might also cause expansion to the corresponding lowest-level sequence, as well as producing sequences of other modalities.

Yet another possible extension toward a more global cognitive model would be a composite input pattern for certain RANs in the cascade. This composite pattern might be the concatenation of representations from different modalities (Fig. 8). The reason for such a concatenation will be presented with an example from natural language. Word representations developed on the basis of the external form of the words which would 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, the input to a sentence-RAN might consist of the developed lexical static representations and the associated patterns from visual modalities. This would let the sentence-RAN develop sentence representations that properly reflect the meanings of the words, not only their external auditory or visual form. This makes the picture more difficult to implement, but the brain we are trying to model is not less complex.

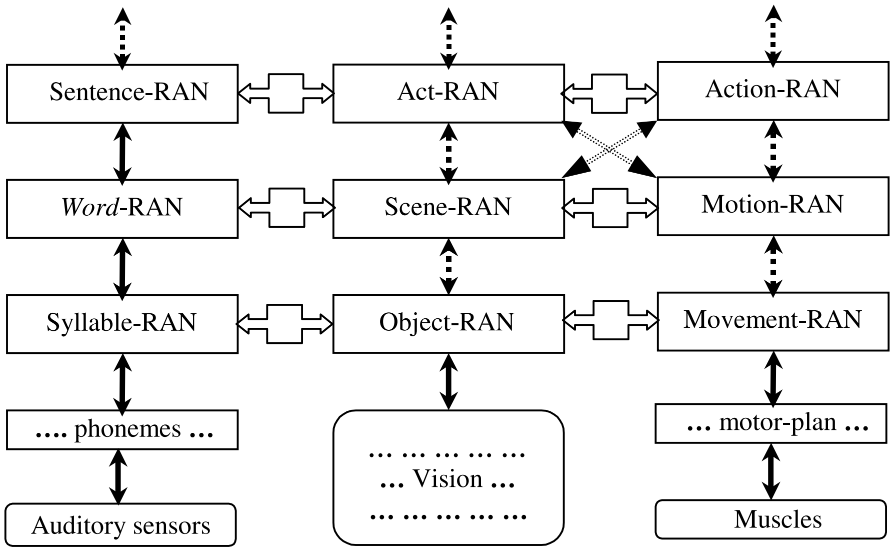


Figure 7. Cognitive model based on a network of RAN cascades. Each RAN-cascade (a column) stands for different modality. The RANs in each cascade represent different conceptual levels. The horizontal and diagonal bi-directional arrows represent static associations between different modalities. In addition to this picture, there should be a central attentive system that directs the flow of activations.

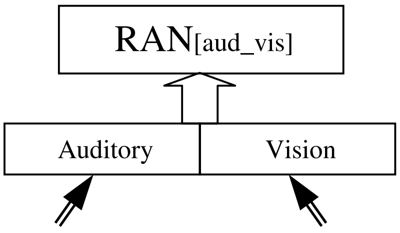


Figure 8. RAN developing static representations of multi-modal sequential data (auditory and visual). The distributed representations developed would feature multi-modal systematicity.

Using these complex schema, we can model complex associations we encounter in our life. Still, there are a lot of other questions to be answered – synchronization, more optimal learning, etc. This huge net of associations needs some central supervisor directing the spread of activations. Modeling cognitive processes such as attention, awareness, etc. maybe would resolve some of those questions.

VII. DISCUSSION

The basic question addressed in this chapter is how to build static representations of complex sequential data with connectionist models, concerning primarily natural language. Dynamic data exist in any cognitive modality, and it is important to have a mechanism that compresses (and uncompresses) dynamic objects into the more convenient static representations. Also, it would be useful if those representations were produced in a systematic way. This would allow further complex processing (holistic computations), e.g., asking questions, change of tense, or even mathematical operations.

Association is one of the basic forms of learning. Repetition, or auto-association, provides a powerful mechanism for cognitive development, too. We can observe this mechanism throughout the animal species. Baby animals develop initial behavior without being taught, but just by attempting to 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 left out of language environment simply can not develop language or have great difficulties developing language later. Similar motivations drove me to use autoassociation in a connectionist model for developing representations of the external world.

Regarding the connectionist models that can process dynamic data, recurrent neural networks are such connectionist models that allow us to process sequential data. The more specific simple recurrent network is a powerful universal model which I have exploited for this purpose, by setting an autoassociation task and arranging the data to develop the desired static representations. The suggested architecture is called 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 level 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 two on the representations of sequences that one needs when modeling natural languages and the capacities of the connectionist models presented in section three 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.

The 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. 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, while the RANs develop unique static representations of the input sequences by a special attack on the learning task, which makes the same RANs able to reproduce the input sequence and which is not the case with the gestalt models. The gestalt model needs a second network trained to extract information from the context of the first network.

The other similar and important architectures: RAAM and SRAAM were initially reported to work well on sequential and recursive data, but other experiments did not confirms the expected performance (Hammerton 1998). Also, the RAAMs need an external stack and boot RAAM and SRAAM reproduce the sequences in inverse order. This might cause interpretational problem when processing longer sequences – it would require an external stack to invert them back to the normal order. Still, the idea of RAN clearly owes the idea of autoassociation as a source of developing compact representations to the RAAM models.

The experiments in section five on modeling sequences with a cascade of RAN modules demonstrate that RANs handle reasonably well both locally encoded low-level data (we can assume that this is a kind of perceptual data) and continuous distributed data. 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 2). On the other hand, word-RAN generalized very well given the small number of training syllabic combinations, which is due to the truly distributed syllabic patterns, as opposed to the localistically encoding input for the syllable-RAN.

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 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 (15). Still, more theoretical and systematic experimental research is necessary in order to determine other factors related to the hidden layer size, such as factors related to the way the data is processed and encoded in the neural networks, especially in recurrent models.

Neural Networks were reproached by Fodor and Pylyshyn [1988] for not being able to produce systematic representations. The ongoing debate on this challenge inspired the development of a number of architectures that more or less meet the requirements characterizing systematicity [Smolensky, 1990, Smolensky, 1991, Aydede, 1997]. In this subsection I will explain how the distributed representations developed by RAN account for systematicity. The debate on this human cognitive property is important because it explains our capacity to think. 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* (atoms constitute thoughts), *generalization* (the atoms "John" and "Mary" are semantically similar and therefore interchangeable) and exploration of *combinatorial* possibilities (similar atoms might apply at same position).

The Fodor's classical "Language of Thoughts" respects fully the first and the third characteristics and partially the second. The hard logic rules, which underlie symbolism – the background of the classical cognitive explanation – can not account for similarities across all items because they do not make use of continuous metrics to compute such similarities. Therefore, in the symbolic systems, similarities across items do not give rise to generalization unless a system of artificially developed features characterizing the items is applied.

On the contrary, an important property of connectionism is generalization, but some connectionist representations do not feature the other characteristics, with the localistic representations being such a very strong example. Feature-based encoding comprises compositionality and allows combinatorial possibilities, but it is rather artificial and symbolic in spirit. 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,1992, Hammerton, 1998].

Similarly to the RAAM and symbolic models, RAN models produce composite representations too. 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 numerically to analyze data and eventually extract useful features from this data. RANs can unpack representations to the original row of tokens – this is part of the autoassociative task. With regard to the other models, the Kohonen Map that was trained in section 4.2 to cluster the distributed representations of syllables clearly demonstrates that other models can "understand" those representations, too (Fig. 5). In this case, the Kohonen Map was just an instrument to persuade the reader that those representations are organized in a systematic way. In addition, similarly to the ability of RAN to decode the distributed representations, other connectionist models should also be able to extract information for the encoded items, that is, to do holistic computations. Experiments with distributed representations produced by RAAM and SRAAM show that this is possible [Chalmers, 1990, Hammerton, 1998] and RANs produce distributed representations following the same principles as 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 the distributed representations developed by the RAN, too.

With regard to the explorations of combinatorial possibilities, the models should not only have the capacity to explore different combinations, but the training environment should provide them to the networks, too. Similarly, symbolic learning algorithms can extract rules only if the learning data provide different examples. This is the same with humans, too. We start to combine words properly after having had 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 sequence. Similarly to Deacon, [1997] I hypothesize that a systematic organization *emerges* in RANs after exploring a great number of possible combinations of patterns and starting to use and rely on some common features among the representations rather than on particular patterns, which Deacon characterizes also as example "forgetting."

VIII. CONCLUSIONS

Sequential processing is recognized 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 processing structured sequential data, as found in natural languages, movies, actions, and so on. The approach is based on the idea that by sequential autoassociation, a single recurrent NN – recurrent autoassociative network – can develop static representations of sequences composed of uniform items. A hierarchical set, or cascade of such networks, develops static distributed representations of ever more complex sequences, where each structural level in the data is processed by one RAN module, and the input sequences for the upper levels are developed by the 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. 4). Those static representations are used as interface patterns for the next level RAN (Fig. 6). The static representations at the highest level RAN are the distributed representations of the most complex data or whole input sequence, e.g., sentences or stories. In section five, an example was given of how this model might work for developing representations of Dutch polysyllabic words. Further, in section six, it was suggested how the cascade model might be extended to a more global cognitive model, where the static representations at each level were suggested to be associated with other static representations (of sequence of other modalities) via static mappings. Such a net of multi-modal associations, I believe, would be an implementation of natural language *grounding* and a base for *semantics*.

Although I claim that this model will be able to solve the problem of developing representations of hierarchically structured sequences, there are still some questions that remain open, especially if we want to develop an autonomous cognitive model. For instance, the learning processes and flow of activations should be driven by a supervisor, similar to the attentive system. Also, the learning algorithm can be replaced with a more effective one. Next, instead of using SRN-based autoassociators, one might use other, more effective or more neurobiologically motivated learning algorithms and neural network models, for example, recurrent self-organization networks and the other models presented in this book. Nevertheless, 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, especially to investigate holistic computations with the distributed representations developed by RANs.

ACKNOWLEDGMENTS

I am grateful to my supervisor, Prof. Dr. John Nerbonne for his useful advice and comments on this work, and to James Hammerton for his important remarks and suggestions on (S)RAAMs and *holistic* computations. Also, I am thankful to Dr. Larry Medsker for his cooperation in the course of writing this chapter. The Kohonen Map tool for data analysis and visualization was provided by Peter Kleiweg. The work is supported by a grant from the Behavioral and Cognitive Neuroscience School at Groningen University, The Netherlands.

REFERENCES

Aydede, M., Language of thought: the connectionist contribution, *Minds and Machines* (7), 57, 1997.

Altman, G. T. M., *Cognitive Models of Speech Processing*, MIT Press, Cambridge, 1990.

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

Blank, D. S., Meeden, and L., Marsgall, J., Exploring the symbolic/subsymbolic continuum: a case study of RAAM, in *Closing the Gap: Symbolism vs. Connectionism*, Dinsmore, J., Ed., Lawrence Erlbaum Assocuiates, 1992.

Blair, A. D., Scaling-up RAAMs, *Tech. Report CS-97-192*, University of Brandeis, 1997.

Cairns, P., Shillcock, R., Chater, N., and Levy, J., Bootstrapping word boundaries: a bottom-up corpus-based approach to speech segmentations, *Cognitive Psychology*, 32(2), 111, 1997.

Carpenter, G. and Grossberg, S., A self-organising neural network for supervised learning, recognition and prediction, *IEEE Communication Magazine*, 38, 1992.

Chalmers, D. J., Syntactic transformations on distributed representations, *Connection Science*, 2(1,2), 53, 1990.

Chappell, C. J. and Taylor, J. G., The temporal Kohonen map, *Neural Networks*, 6, 441, 1993.

Christiansen, M. H. and Chater, N., Toward a connectionist model of recursion in human linguistic performance, *Cognitive Science,* (in press), 1999.

Clark, A., Systematicity, structured representations and cognitive architecture: a reply to Fodor and Pylyshyn, in *Connectionism and the Philosophy of Mind*, Horgan, T. and Tienson, J., Eds., Kluwer Academic Publishers, Dordrecht, 198, 1991.

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

Cottrell, G. W., Bartell, B., and Haupt, C., Grounding meaning in perception, *German Workshop on Artificial Intelligence*, 1990.

Deacon, T. W., *The Symbolic Species*, W. W. Norton & Co. Inc, New York, 1997.

Dorffner, G., "Radical" connectionism for natural language processing, *Proceedings of Spring Symposium on Connectionism*, NLP, Standford, 95, 1991.

Elman, J. L., Finding structure in time, *Cognitive Science*, 14, 179, 1990.

Elman, J. L., Bates, E., Johnson, M. H., Karmiloff-Smith, A., Parisi, D., and Plunket, K., *Rethinking Innates*, MIT Press, Cambridge, MA, 1996.

Fodor, J. and Pylyshyn, Z., Connectionism and cognitive architecture: a critical analysis, *Cognition*, 28, 3, 1988.

Fodor, J. and McLaughlin, B., Connectionism and the problem of systematicity: why Smolensky's solution doesn't work, in *Connectionism and the Philosophy of Mind*, Horgan, T. and Tienson, J., Eds., Kluwer Academic Publishers, The Netherlands, 1991.

Gasser, M., Learning distributed representations for syllables, *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, 396, 1992.

Grossberg, S., *Studies of Mind and Brain*, Kluwer Academic Publishers, Boston, 1982.

Hammerton, J. A., Exploiting holistic computations: an evaluation of the sequential RAAM, Ph.D. thesis, University of Birmingham, UK, 1998.

Haykin, S., *Neural Networks*, Macmillan College Pub., 1994.

Henderson, J. Connectionist architecture with inherent systematicity, *Proceedings of the 18th Conference of the Cognitive Science Society*, La Jolla, CA, 1996.

Henderson, J. and Lane, P., A connectionist architecture for learning to parse. *Proceedings of COLING-ACL Conference*, Montreal, 531, 1998.

Hirshman, E. and Jackson, E., Distinctive perceptual processing and memory, *Journal of Memory and Language*, 36, 2, 1997.

Hornik, K., Stinchcombe, M., and White, H., Multilayer feedforward networks are universal approximators, *Neural Networks*, 2, 359, 1989.

Hopfield, J. J., Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences*, USA, 79, 2554, 1982.

James, D. L. and Miikkulainen, R., SARDNET: a self-organising feature map for sequences, in *Advances in Neural Processing Systems*, Tesauro, Touretzky, and Leen, Eds., 7, 1995.

Jordan, M. I., Attractor dynamics and parallelism in a connectionist sequential machine, *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, Amherst, MA, 531, 1986.

Jusczyk, P. W., *The Discovery of Spoken Language*, MIT Press, Cambridge, MA, 1997.

Kohonen, T., *Self-Organisation and Associative Memory*. Springer-Verlag, New York, 1984.

Kwasny, S. C. and Kalman, B. L., Tail-recursive distributed representations and simple recurrent networks, *Connectionist Science*, 7(1), 61, 1995.

Lawrence S., Giles, C. L., and Fong, S., On the applicability of neural networks and machine learning methodologies to natural language processing, *Technical Report UMIACS-TR-95-64 and CS-TR-3479*, University of Maryland, 1995.

Miikkulainen, R. and Dyer, M., Natural language processing with modular PDP networks and distributed representations, *Cognitive Science*, 15(3), 343, 1991.

Plate, T. A., Distributed representations and nested compositional structure, Ph.D. thesis, University of Toronto, 1994.

Plaut, D. C., McClelland, J., and Seidenberg, M., Connectionist models of memory and language, in Levy, J., Bairaktaris, D., Bullinaria, J., Cairns, P., Eds., UCL Press Ltd., London, 1995.

Plaut, D. C., McClelland, J., Seidenberg, M., and Patterson, K., Understanding normal and impaired word reading: computational principles in quasi-regular domains, *Psychological Review*, 103, 56, 1996.

Pollack, J. B., Recursive distributed representation, *Artificial Intelligence*, 46, 77, 1990.

Reilly, R. G., Sandy ideas and coloured days: some computational implications of embodiment, *Artificial Intelligence Review*, 9, 305, 1995.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., *Parallel Distributed Processes*, 1&2*,* MIT Press, Cambridge, MA, 1986.

Searle, J. R., *Minds, Brains and Science*, Harvard University Press, Cambridge, MA, 1984.

Seidenberg, M. S. and McClelland, J. L., A distributed, developmental model of word recognition & naming, *Psychological Review*, 96, 523, 1989.

Sejnowski, T. J. and Rosenberg, C. R., Parallel networks that learn to pronounce English text, *Complex Systems*, 1, 145, 1987.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L., Graded state machines: the representation of temporal contingencies in simple recurrent networks, *Machine Learning*, 7, 161, 1991.

Smolensky, P., Tensor product variable binding and the representation of symbolic structures in connectionist systems, *Artificial Intellig*ence, 46, 159, 1990.

Smolensky, P., The constituent structure of mental states: a reply to Fodor and Pylyshyn, in *Connectionism and the Philosophy of Mind*, Horgan, T. and Tienson, J., Eds., Kluwer Academic Publishers, Dordrecht, 281, 1991.

Sperduti, A., On the computational power of recurrent neural networks for structures, *Neural Networks*, 10(3), 395, 1997.

St. John, M. F. and McClelland, J. L., Learning and applying contextual constraints in sentence comprehension, *Artificial Intelligence*, 46, 217, 1990.

Stoianov, I. P., Recurrent autoassociative networks and sequential processing, *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, 1999.

Stoianov, I. P., Nerbonne, J., and Bouma, H., Modelling the phonotactic structure of natural language words with simple recurrent networks, in *Computational Linguistics in the Netherlands*, Coppen, P.-A., van Halteren, H., and Teunissen, L., Eds., Rodopi, Amsterdam, 77, 1997.

Stoianov, I. P., Stowe, L., and Nerbonne, J., Connectionist learning to read aloud and correlation to human data, *21st Annual Conference of the Cognitive Science Society*, Vancouver, 1999.

Tabor W., Juliano, C., and Tanenhaus, M. K, Parsing in an dynamical system: an attractor-based account of the interaction of lexical structural constraints and sentence processing, *Language and Cognitive Processing*, 1997.

Tsoi, A.C. and Back, A. D., Locally recurrent globally feedforward networks: a critical review of architectures, *IEEE Transactions on Neural Networks*, 5, 229, 1994.

Wilson, W. H., Tower networks and letter prediction, *Cognitive Modelling Workshop of the 7th Australian Conference on Neural Networks*, http://psych.psy.uq.oz.au/workshop.html, 1996.