Connection Science, Vol. 16, No. 2, June 2004, 71–99

Self-refreshing memory in artificial neural networks: learning temporal sequences without catastrophic forgetting

Bernard Ans*, Stéphane Rousset*, Robert M. French† and Serban Musca*
**Psychology and NeuroCognition, Pierre Mendes-France University, Grenoble 2—CNRS UMR 5105, BP 47, 38040 Grenoble cedex 09, France*
email: Bernard.Ans@upmf-grenoble.fr
tel: +33-476-825-674
fax: +33-476-827-834
†*Quantitative Psychology and Cognitive Science, University of Liège (Bat B32), Sart Tilman, 4000, Liège, Belgium*

Abstract. While humans forget gradually, highly distributed connectionist networks forget catastrophically: newly learned information often completely erases previously learned information. This is not just implausible cognitively, but disastrous practically. However, it is not easy in connectionist cognitive modelling to keep away from highly distributed neural networks, if only because of their ability to generalize. A realistic and effective system that solves the problem of catastrophic interference in sequential learning of 'static' (i.e. *non-temporally ordered*) patterns has been proposed recently (Robins 1995, *Connection Science*, **7**: 123–146, 1996, *Connection Science*, **8**: 259–275, Ans and Rousset 1997, *CR Académie des Sciences Paris, Life Sciences*, **320**: 989–997, French 1997, *Connection Science*, **9**: 353–379, 1999, *Trends in Cognitive Sciences*, **3**: 128–135, Ans and Rousset 2000, *Connection Science*, **12**: 1–19). The basic principle is to learn new external patterns interleaved with internally generated 'pseudopatterns' (generated from random activation) that reflect the previously learned information. However, to be credible, this self-refreshing mechanism for static learning has to encompass our human ability to learn serially many *temporal sequences* of patterns without catastrophic forgetting. Temporal sequence learning is arguably more important than static pattern learning in the real world. In this paper, we develop a dual-network architecture in which self-generated pseudopatterns reflect (non-temporally) all the sequences of temporally ordered items previously learned. Using these pseudopatterns, several self-refreshing mechanisms that eliminate catastrophic forgetting in sequence learning are described and their efficiency is demonstrated through simulations. Finally, an experiment is presented that evidences a close similarity between human and simulated behaviour.

Keywords: artificial neural networks, temporal sequence learning, catastrophic forgetting, self-refreshing memory.

1. Introduction

Humans simply forget; many connectionist networks, on the other hand, forget catastrophically. Specifically, the broad and widely used class of highly distributed connectionist networks—in particular, multi-layer networks trained with the backpropagation algorithm—suffer from catastrophic interference (McCloskey and Cohen 1989, Ratcliff 1990), which occurs when newly learned information completely erases previously learned information. This is not just implausible cognitively, but

Connection Science ISSN 0954-0091 print/ISSN 1360-0494 online © 2004 Taylor & Francis Ltd
http://www.tandf.co.uk/journals
DOI: 10.1080/09540090412331271199

disastrous for practical applications, which is why numerous authors over the last decade and a half have developed ways to overcome this problem (Hetherington and Seidenberg 1989, McCloskey and Cohen 1989, Kortge 1990, Ratcliff 1990, Lewandowsky 1991, 1994, Murre 1992, French 1992, 1994, 1997, McRae and Hetherington 1993, Lewandowsky and Li 1995, McClelland *et al.* 1995, Robins 1995, 1996, Sharkey and Sharkey 1995, Ans and Rousset 1997, 2000, Robins and McCallum 1998, 1999, French *et al.* 2001 and others; for a review, see French 1999). The heart of the problem is that the very property—a single set of weights to encode information—that gives connectionist networks their remarkable abilities to generalize and to have a gradual drop off in performance in the presence of incomplete information is also the root cause of catastrophic interference. When such a system learns a new set of items, a large majority of its weights that were adjusted for previously learned items will be modified, which frequently results in a severe loss of the old information. This is the 'sensitivity–stability dilemma' (Hebb 1949) or 'stability–plasticity dilemma' (Grossberg 1987, Carpenter and Grossberg 1988). While it is true that memory models that use non-overlapping, or sparsely distributed, representations face this dilemma to a considerably lesser extent (e.g. Hintzman 1986, Grossberg 1987, Kanerva 1988, Kruschke 1992, 1993, Ans *et al.* 1998), good generalization often requires a highly distributed system; and along with this high degree of distribution comes catastrophic interference.

Networks that suffer from catastrophic interference obviously cannot learn patterns sequentially. This is why backpropagation networks must learn collections of patterns 'concurrently', i.e. modifying the network's weights a very small amount for each item and making many passes through the training set to settle gradually on an appropriate set of weights for the entire set of items. Numerous ways are known to achieve sequential learning in these networks, the simplest of which is to 'rehearse' the old items as new learning occurs. In other words, when new information is presented to the network, various amounts of the previously learned information are mixed in with the new information, and catastrophic interference is effectively eliminated. This solution, however, requires the unrealistic assumption of permanent access to all patterns on which the network was previously trained. Various authors (Robins 1995, 1996, Ans and Rousset 1997, 2000, French 1997, 1999) have developed systems that rehearse on *pseudo*-episodes (or pseudopatterns), rather than on the real items that were learned previously. The basic principle of this mechanism is to learn new external patterns interleaved with *internally generated* pseudopatterns. A set of these patterns reflects (but is not identical to) the previously learned information. It has now been established that this pseudopattern rehearsal method does indeed eliminate catastrophic forgetting, thereby allowing effective sequential pattern learning.

A serious problem remains, however, which is: cognition not only involves being able to learn sequentially a series of 'static' (i.e. non-temporally ordered) patterns without interference, but, of equal importance, also requires us to be able to learn serially many *temporal sequences of patterns*, one after another. In other words, while it is important to be able to learn sequentially to recognize a series of faces, much of what we do consists of mastering the serial order of pattern sequences. This includes everything from starting the car in the morning to baking a loaf of bread, from getting dressed to answering the phone. In fact, virtually everything we do involves, in one way or another, sequences of events in time. So, how can we get a network to learn several temporal sequences without catastrophic interference? We shall present an approach that overcomes this problem using a technique closely related to the original pseudopattern approach (cf. early results presented in Ans *et al.* 2002).

A remarkable result of this paper is that, in order for the network to learn multiple temporal sequences, interleaving *pseudo-sequences* with the new sequences to be learned simply does not work. Rather, once a network has learned a particular sequence, it may be used to *generate pseudopatterns that collectively embody information about the entire sequence (or set of sequences) that was learned.* In other words, when learning a new sequence, simple rehearsal with these 'non-ordered' pseudopatterns will prevent catastrophic forgetting of the previously learned sequence(s).

The network on which the dual-network architecture described in this paper is based is a modified version of the well-known simple recurrent network (SRN) proposed by Elman (1990) for temporal sequence learning. The proposed modification to the standard SRN involves adding hidden-to-input layer connections in order to create an autoassociative part required by the reverberation process introduced by Ans and Rousset (1997), a process that greatly increased the effectiveness of pseudopatterns in eliminating catastrophic forgetting. We call the modified network a 'reverberating SRN' (hereafter, RSRN). The addition of an autoassociative part to a SRN was first proposed in a different framework by Maskara and Noetzel (1992, 1993), who called their network an 'autoassociative recurrent network' (AARN). The aim was to improve the prediction ability of SRNs, and it has been shown that AARNs can do sequence-learning tasks that cannot be done with a standard SRN (Maskara and Noetzel 1993, Cleeremans and Destrebecqz 1997).

We coupled two RSRNs to create a dual-network architecture that is similar to the model originally developed by Ans and Rousset (1997, 2000). Among other models of temporal sequence learning (e.g. Jordan 1986, Ans 1990, Reiss and Taylor 1991, Ans *et al.* 1994), the SRN model was chosen because it is the most widely used.

The remainder of this paper is organized as follows. We shall briefly review the standard dual-network pseudopattern solution to the problem of catastrophic forgetting in static pattern learning. We go on to show why the use of pseudo-sequences to solve this problem in the framework of multiple sequence learning is inappropriate. We then demonstrate that simple pseudopatterns instead of pseudo-sequences can be used to overcome the problem of multiple sequence learning. We discuss how these techniques can be used to learn long sequences by learning short sub-sequences of the long sequence, and we discuss a number of possible ways in which the self-refreshing mechanism could be generalized. Further, we check the robustness of this mechanism with regard to sequence complexity. We do this by showing that the mechanism works without difficulty on so-called second-order conditional sequences (or SOCs, Cohen *et al.* 1990, Reed and Johnson 1994), complex sequences of items in which no single item ever deterministically predicts its successor. Finally, to sustain the self-refreshing process, we carried out a behavioural experiment with human subjects in order to track in detail the temporal course of retroactive interference during sequence learning. These results were then compared with those from the simulation with the dual-network RSRN.

2. Overcoming catastrophic interference with pseudopatterns

Before discussing the dual-network RSRN architecture that is the subject of this paper, we need briefly to describe what pseudopatterns are and how they can be used to reduce catastrophic interference.

2.1. *Simple pseudopatterns*

Assume we have a three-layer feedforward network with k binary input units and m binary output units. The network learns a number of patterns drawn from some distribution. Thereafter, these patterns are no longer available. How can one determine, even approximately, what the network has learned? Answer: by 'bombarding' the input of the network with *random* binary vectors (i.e. vectors consisting of 0s and 1s, each with a probability of 0.5 of being chosen) of length k and collecting the associated output vectors of length m. Each input–output pair of vectors produced in this way will constitute a *pseudopattern* that will be a reflection of the function previously learned by the network.

2.2. *How pseudopatterns are used to reduce catastrophic interference*

The basic idea is as follows (Robins 1995). The network learns a first set of patterns, $\{P_i\}$. Then before it begins to learn the second set of patterns, $\{Q_i\}$, noise is fed through the network to produce a set of pseudopatterns, $\{\psi_i\}$. These pseudopatterns are added to the new patterns to be learned and the network continues to train on this larger set until all of the new patterns $\{Q_i\}$ are below criterion. When the network is tested on the originally learned patterns, $\{P_i\}$, it has not forgotten them catastrophically.

Where are the internally generated pseudopatterns stored so that they can be interleaved with the new patterns? One answer is to generate them on the move in a separate network (Ans and Rousset 1997, 2000, French 1997). Let us consider one way that a single new pattern, Q, might be learned in this dual-network model. We assume that new patterns are learned by one network (NET 1), while previously learned patterns are stored in another network (NET 2). NET 1 learns Q as follows:

(i) pattern Q is input to NET 1, which modifies its weights once.
(ii) Noise is input to NET 2, which generates a pseudopattern and
(iii) this pseudopattern is input to NET 1, which modifies its weights once.
(iv) If Q has been learned to criterion, stop; otherwise go to (i).

We call this the *interleaving phase.* Once the output error for pattern Q has dropped below criterion, we transfer this information to NET 2 as follows:

(i) noise is input to NET 1, which generates a pseudopattern and
(ii) this pseudopattern is input to NET 2, which modifies its weights once.
(iii) Go to (v). Loop X times, for some predetermined value of X.

We call this phase, when information is transferred to NET 2, the *transfer phase.*

2.3. *'Reverberating' feedforward networks*

A reverberating feedforward network (RFN) is a standard three-layer backpropagation network that integrates an autoassociative processing constraint (Ans and Rousset 1997, 2000). This architecture can be visualized in two equivalent manners. In the first (see figure 1(a)), the reverberating (i.e. autoassociative) part is shown in an 'unfolded' manner, which emphasizes precisely how the learning algorithm works, at the expense of an intuitive grasp of the reverberation. In this visualization, the output layer is divided into the output nodes of the autoassociative part (on the left in figure 1(a)) and the nodes for the targets of the patterns to be learned (on the right in the figure). On the other hand, the second way of visualizing the architecture (see figure 1(b)) emphasizes the crucial reverberating part of the network, but renders the learning algorithm somewhat more difficult to understand.

Consider figure 1(a) and assume the network is to learn a pattern $P: I \rightarrow T$ consisting of an input I and a target T. I is presented on input and is sent through the network to the output layer. For all of the nodes in the output layer, an error is calculated. For the 'autoassociative' output nodes, the error is based on the difference between the network

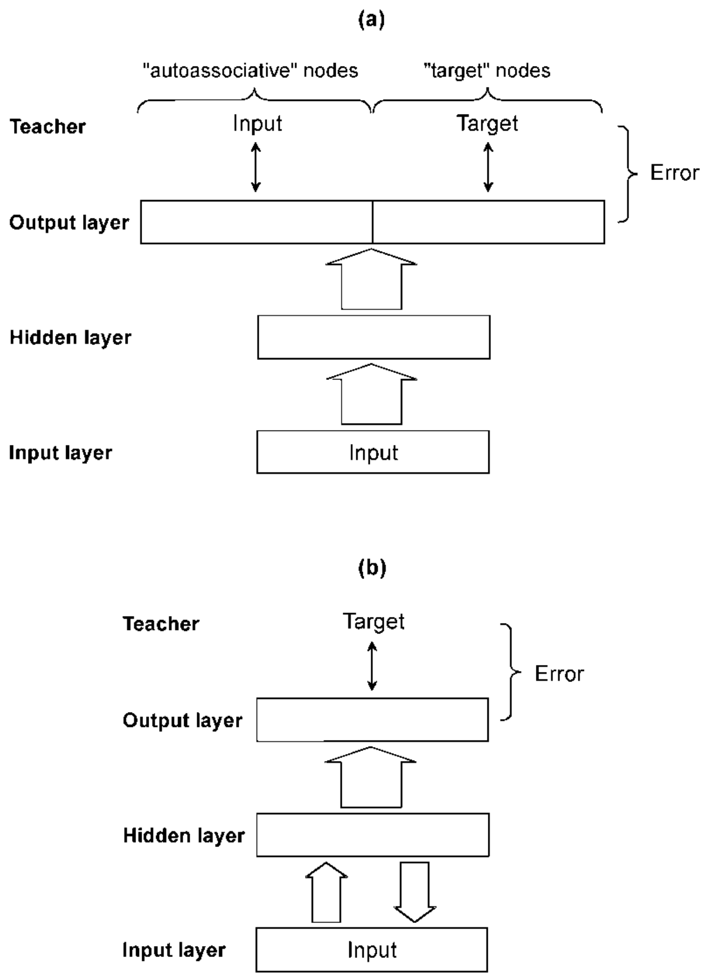


Figure 1. (a) The RFN architecture integrates an autoassociative processing constraint into a standard backpropagation network (large arrows represent full connectivity with modifiable weights). Here the emphasis is on the learning algorithm. The network is shown learning a pattern P: Input $\rightarrow$ Target. (b) An equivalent visualization of the RFN architecture emphasizing the input–hidden layer reverberation. It is crucial to note that the updating of the hidden-to-input weights depends not only on the autoassociative error between the original input and the reverberated input, but also on the difference between the network's actual output and the target. As above, the network is shown learning a pattern, P: Input $\rightarrow$ Target.

output and the original input I, whereas for the 'target' nodes, this error is based on the difference between the 'heteroassociative' output and the desired target T. The errors associated with both the 'autoassociative' and the 'target' output nodes are then backpropagated through the network in order to change the network's weights.

2.4. *'Attractor' pseudopatterns and their generation in RFNs*

To generate pseudopatterns in an RFN, a random input i_ψ is presented to the input layer of the network and fed through to the output layer. The activation values of the 'autoassociative' nodes in the output layer (nodes on the left of the output layer in figure 1(a)) constitute a new input, i'_{ψ}, which is then sent through the network (the activation values on the 'target' nodes in the output layer are ignored). This produces a pattern of activation on the 'autoassociative' output nodes, i''_{ψ}, which is then presented to the input nodes of the network, and so on. After a number R of reverberating cycles through the network, a final 'reverberated' input i^R_{ψ} is sent through the network and the activation vector of all the output nodes (the 'autoassociative' and the 'target' output nodes), o_{ψ}, is used to produce a pseudopattern Ψ: $i^R_{\psi} \rightarrow o_{\psi}$. The advantages of using an autoassociative part with a feedforward-backpropagation network have been shown elsewhere (Ans and Rousset 1997, 2000). Suffice it to say that the reverberation process transforms a pure random input i_{ψ} into an *attractor* of the entire system, ψ. As such, the 'attractor' pseudopattern produced provides a much better reflection of the old patterns than a pseudopattern produced from simple random noise on input. It is this reverberation technique that is largely responsible for the power of this technique, whether in standard feedforward or simple recurrent networks.

2.5. *Simple recurrent networks*

The simple recurrent network (SRN, see Elman 1990) is designed to learn temporally ordered sequences of patterns $S(0)$, $S(1), \ldots, S(t), \ldots, S(n)$, where t is the time parameter. The basic idea is that, in order to learn an 'ambiguous' sequence (e.g. one that contains two identical patterns), it is necessary to include context to distinguish the different parts of the sequence in which the identical items are found. For this, 'context' units are added to the input (see figure 2). For a given item $S(t)$ in the sequence, we input not only $S(t)$ to the network, but also $H(t-1)$, which is the hidden layer activation vector associated with $S(t-1)$. The standard backpropagation algorithm is used to change the weights of the SRN. For the first learning iteration (i.e. $S(0) \rightarrow S(1)$), the context units on the input layer are all initialized to 0.5. Hereafter, this initializing context pattern will be referred to as the 'neutral context'.

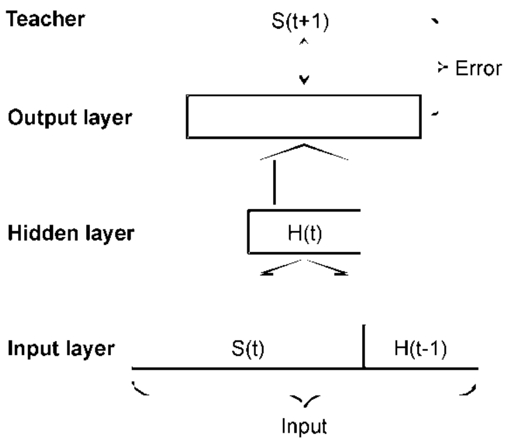


Figure 2. A standard SRN network that is designed to learn a sequence $S(0)$, $S(1), \ldots, S(t), \ldots, S(n)$. At each time t, the relation between item $S(t)$ and the associated target item $S(t+1)$ is learned along with the context $H(t-1)$, a copy of the hidden layer activation from time $t-1$ when the network was learning the previous association $S(t-1) \rightarrow S(t)$.

2.6. *Reverberating simple recurrent networks*

A RSRN (see figure 3) works very much like a standard SRN. Just as the RFN network involves adding 'autoassociative' nodes to the output layer of a BP network, a reverberating SRN involves adding 'autoassociative' nodes to the output layer of a standard SRN. The full input to the network consists of the 'standard' input, i.e. the input from the sequence item $S(t)$, and the 'context' input, $H(t-1)$, which is the activation vector of the hidden layer associated with the previous item in the sequence, $S(t-1)$.

2.7. *'Attractor' pseudopattern generation in a RSRN*

The principle of pseudopattern generation in a RSRN is similar to that used in a RFN. A random input i_ψ is presented to the input layer of the network (i.e. random binary values, with 0.5 probability, for both standard input units and context units) and fed through the network to the output layer. Only the activation values of the 'autoassociative' nodes in the output layer (see figure 3) are used to constitute a new input, i'_ψ (as in the RFN, the activation values on the 'target' nodes in the output layer are ignored). This vector i'_ψ is then fed through the network in order to produce a pattern of activation on the autoassociative output nodes, i''_ψ. This vector i''_ψ is then presented to the input nodes of the network, and so on. In a manner exactly analogous to pseudopattern generation in the RFN, after a number R of these 'reverberating' cycles through the network, the final 'reverberated' input i^R_ψ is sent through the network. The activation vector on all output nodes ('autoassociative' and 'target' output nodes), o_ψ, is used to produce a pseudopattern Ψ: $i^R_\psi \to o_\psi$.

Crucially, once the RSRN has learned a sequence of items, a set of pseudopatterns generated as above reflects *the entire learned sequence(s)*. The larger this set of pseudopatterns, the better it will reflect the previously learned sequence(s). This is precisely why pseudopatterns are able to eliminate catastrophic forgetting of the old sequence when a new sequence is learned.

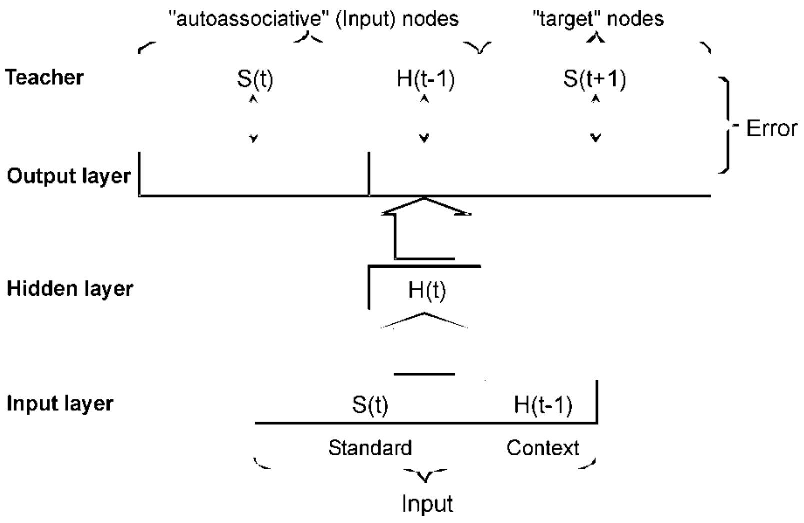


Figure 3. A reverberating SRN. This architecture can also be visualized as in figure 1(b) to emphasize the input reverberation between the input and hidden layer.

3. A dual-network architecture with self-refreshing memory to overcome catastrophic forgetting in temporal sequence learning

3.1. *An overview of the basic dual-network architecture*

Ans and Rousset (1997, 2000) proposed a reverberating dual-network architecture with a self-refreshing memory to avoid catastrophic forgetting in static pattern learning. It is necessary to describe this basic model in order to show how an analogous architecture for multiple sequence learning can overcome catastrophic forgetting in temporal sequence learning.

The basic dual-network architecture consists of two coupled RFNs (see figure 1(a)), denoted NET 1 and NET 2. NET 1 is the primary network that interfaces with the environment and that learns the new patterns. The secondary network, NET 2, is a 'storage' network because information initially learned in NET 1 will ultimately be transferred to NET 2. This secondary network has the same input layer and output layer structure as the primary network. (The two networks can, theoretically, have different numbers of hidden layers of different sizes. For the sake of simplicity, the two networks described in this paper are identical.) The basic idea underlying the operation of this system is the following. First, suppose that some previously learned information is already stored in NET 2. When a new pattern P^{new} (recall that a pattern consists of an input *and* a target) arrives at NET 1, this network performs one learning pass with this pattern (i.e. one feedforward pass and one weight change backpropagation pass). In the mean time, noise is sent through NET 2 in order to produce a pseudopattern (that will reflect the contents of the previously learned information stored in NET 2). This pseudopattern (pseudo-input and associated network output) is then presented to NET 1, which performs a single learning pass on this pseudopattern. (Note that more than one pseudopattern from NET 2 can be presented to NET 1 in this way after each P^{new} learning pass.) P^{new} is then presented to NET 1 again and it performs another learning pass on this pattern. Another pseudopattern is then generated by NET 2 and NET 1 does another learning pass on this pseudopattern. This process continues until P^{new} is learned to criterion by NET 1.

The same procedure applies if a set of new patterns is to be concurrently learned, the only difference being that NET 1 performs a single feedforward-backpropagation pass for each of the new patterns to be learned and for each of the pseudopatterns generated by NET 2. This continues until all of the new patterns are learned to criterion.

As discussed in section 2.2, the phase in which NET 1 learns new information from the environment, augmented by pseudopatterns from NET 2, is called the *interleaving phase* of the system. There is also a *transfer phase* during which the information learned in NET 1 is transmitted to NET 2. This involves NET 1 generating a number of pseudopatterns and, for each of these NET 1 pseudopatterns, NET 2 performs a single learning pass, thereby transferring information stored in NET 1 to NET 2.

3.2. *Learning in RSRN dual-networks*

The basic principles of RSRN dual-network learning, where each of the networks in the system is a RSRN, are very similar to those underlying dual-networks of RFNs. Dual-network RSRNs are designed to learn multiple sequences of patterns.

Assume that we have two RSRNs (see figure 3), which, as before, we shall designate as NET 1 and NET 2. NET 1 and NET 2 are identical networks. New sequences will be learned only by NET 1, while NET 2 will store the previously learned information. The learning procedure is similar to that of the basic RFN. A sequence, $S = S(0)$, $S(1), \ldots, S(n)$, is presented to NET 1. The network makes a single pass through the entire sequence, updating its weights once for each item in the sequence. This defines one learning 'epoch' corresponding to one presentation of all items in the sequence in order. Next, NET 2 generates a number of pseudopatterns (e.g. 10 per learning epoch), as described in section 2.7. These pseudopatterns are *close to attractor states* of NET 2, which makes them particularly good vehicles for information transfer from NET 2 to NET 1. For each NET 2 pseudopattern, NET 1 performs one feedforward-backpropagation learning pass. Once this is completed, a new learning epoch starts and NET 1 makes another pass through the sequence *S*. NET 2 generates new pseudopatterns, each of which is learned for one feedforward-backpropagation pass by NET 1; and so on. This is the interleaving phase for a RSRN dual-network.

It is extremely important to realize that the pseudopatterns generated by NET 2 are *static* information representing a previously learned *dynamic* sequence. This is what gives this system its power: there is no need to attempt to reproduce an entire *pseudo-sequence* that will then be interleaved with the new sequence being learned (and, in fact, as we shall see, this intuitive solution does not work). Rather, we need only to interleave with the new sequence to be learned a set of pseudopatterns that reflects—independently of the occurrence order of each individual pseudopattern—the information of the entire previously learned temporal sequence (or sequences).

To transfer a sequence learned by NET 1 to NET 2, we again make use of pseudopatterns. This time the pseudopatterns are generated by NET 1 and learned by NET 2. For each pseudopattern generated by NET 1—again, these will be close to attractor states of NET 1—NET 2 performs a single feedforward-backpropagation learning pass.

4. Simulations

4.1. *Overview*

We shall present six simulations. The first will demonstrate the severity of catastrophic interference in multiple sequence learning in standard SRNs. We shall then show, in a second simulation, that the intuitively plausible idea of using pseudo-*sequences* to alleviate this problem does not work. The third simulation will demonstrate that interleaving a set of pseudo*patterns* with the new sequence being learned does, indeed, eliminate catastrophic interference. The fourth simulation will show how this technique can be used to allow networks to learn long sequences by sequentially learning sub-sequences of the long sequence. The fifth simulation will show that pseudopatterns reflecting the previously learned sequence can be inserted randomly *within* the new sequence as it is being learned (rather than, as in the other simulations, requiring them to be inserted only after each learning epoch of the new sequence) and this will still allow the network to learn the new sequence and not forget the old one. Finally, in the sixth simulation, we shall show that the self-refreshing mechanism can cope with second-order conditional sequences (or SOCs, Cohen *et al.* 1990, Reed and Johnson 1994), highly ambiguous sequences in which no single element alone uniquely predicts its successor.

4.2. *Network architecture and parameter settings*

A standard SRN was used for the first simulation that demonstrates the severity of catastrophic interference in multiple sequence learning. In the second simulation, two standard SRN networks were used to explore learning with refreshing by interleaving pseudo-sequences. A dual-network architecture consisting of two coupled RSRNs will be used for the remaining four simulations. Each RSRN has an input layer with 100 'standard' input units (corresponding to the length of the items, $S(t)$, in the sequence) and 50 'context' units. The hidden layer consists of 50 units. Finally, the output layer consists of 250 units: 150 'autoassociative' inputs that are identical to the input layer, plus 100 'target' units (see figure 3).

Learning a given sequence involves presenting it repeatedly to the network until the network can generate the entire sequence (with a predefined degree of precision), by starting with the initial item of the sequence (paired with the neutral context; see section 4.3) and feeding each output back into the input, thereby producing each successive item of the sequence. The network weights are updated by backpropagation once per presentation of each sequence item. A cross-entropy error function is used (Hinton 1989a, Plaut *et al.* 1996) with a learning rate of 0.01, a momentum of 0.5 and a 1.0 bias term. All weights are randomly initialized between -0.5 and 0.5.

Each random input vector is 'reverberated' five times (i.e. $R = 5$; see section 2.7) before this reverberated input is sent through the network to the target output units to create the pseudopattern that will be used. Ten pseudopatterns from NET 2 are interleaved after each learning epoch of the new sequence to be learned by NET 1 (except in the fifth simulation where this number will be varied). During transfer of the information from NET 1 to NET 2, 10^4 pseudopatterns are used (except in the sixth simulation in which this number will be greater).

4.3. *Measuring sequence learning and forgetting*

A recall test of a given sequence $S(0), S(1), \ldots, S(t), \ldots, S(n)$ is performed as follows. First, the input layer of the network is initialized by the starting item $S(0)$ and the neutral context vector. This concatenated vector is sent through the network and produces an output vector consisting of an autoassociative part and a heteroassociative (i.e. target) part (see figure 3). The heteroassociative part of the output, denoted by $\sigma(1)$, is then *reinjected* to the 'item' part of the input vector while the activity of the hidden layer is copied to the 'context' part of the input vector. This new input vector is sent through the network and produces an output vector consisting, as before, of an autoassociative part and a heteroassocative part, $\sigma(2)$, and so on. Hence, a series of heteroassociative outputs $\sigma(1)$, $\sigma(2), \ldots, \sigma(t), \ldots \sigma(n)$ are sequentially generated by the network from the successive reinjections at the input layer of hidden and output layer activity *generated by the network itself.* It is important to note that this test of sequence recall is considerably more stringent that the local prediction tests often used in the sequence-learning literature, where what is evaluated is the ability of the network to predict the subsequent item of a sequence when the *true* preceding item is provided to the network. In this paper, we are interested in the network's ability *to self-reproduce a whole sequence*, from beginning to end, from a single starting input activation, without any information provided from the environment (i.e. the true preceding elements).

For each input, $\sigma(t)$, reinjected from the generated sequence and fed through the network again, we evaluate the discrepancy between the resulting computed heteroassociative output $\sigma(t+1)$ and the desired target item of the sequence $S(t+1)$. We calculate the difference $S(t+1) - \sigma(t+1)$ for each of the 100 'target' output units and count the number of output units for which the absolute value of this difference is above a given learning criterion (except where specified, this criterion was set to 0.1). So, for example, if on the output layer 14 of the 'target' output units of $\sigma(t+1)$ differ from the corresponding units of $S(t+1)$ by more than 0.1, we say that there are 14 'incorrect' units. A sequence is considered to have been learned if, from its starting item $S(0)$, associated with the neutral initial context, the network produces a series of computed vectors $\sigma(t)$, each of which has all of its units within 0.1 of the corresponding unit of the expected sequence item, $S(t)$.

The measure of how well the network has learned (or forgotten) a sequence after a given number of learning epochs will be the number of incorrect units for each item of the sequence. In simulation 5 we shall indicate only the total number of incorrect units over the sequence.

4.4. *Simulation 1: catastrophic forgetting in multiple sequence learning*

To illustrate the severity of catastrophic forgetting in multiple sequence learning, we shall consider two sequences, A and B, and have a SRN attempt to learn them sequentially. The sequences are constructed as follows. Twenty-two distinct random binary vectors of length 100 are created (i.e. vectors consisting of 0s and 1s with a selection probability of 0.5). Half of these vectors are used to produce the first ordered sequence of items, A, denoted by $A(0), A(1), \dots, A(10)$. The remaining 11 vectors are used to create a second sequence of items, B, denoted by $B(0), B(1), \dots, B(10)$. In order to introduce a degree of ambiguity into each sequence (so that a simple BP network would not be able to learn them), we modify each sequence so that $A(5) = A(8)$ and $B(1) = B(5)$. First, sequence A is perfectly learned by the network. Then sequence B is learned and, during the course of learning, we monitor at regular learning-epoch intervals (using recall performance tests as defined in section 4.3) how much of sequence A has been forgotten by the network.

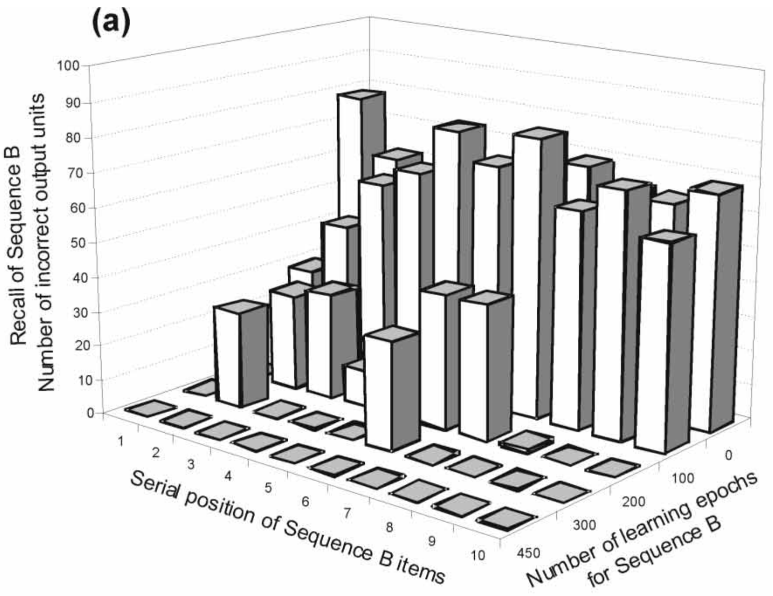
Figure 4(a) shows the progression of the network's learning of sequence B. The number of incorrect units for each serial position of the sequence, as defined earlier, is shown. (Note: in figure 4, as well as in subsequent figures, serial positions start at 1 since the sequence item $S(0)$ is used only to initialize sequence learning and recall. $S(1)$ is the first sequence item actually recalled by the network.) As expected, it is harder for the network to learn $B(2)$ and $B(6)$ since their immediate predecessors are identical and, in order to distinguish them, the network needs additionally to take into consideration the context of the preceding items in the sequence. After 450 epochs, the network has completely learned the entire sequence B.

During learning of sequence B, we monitored the forgetting of the previously learned sequence, A (see figure 4(b)). Initially (i.e. before the network began learning sequence B), it can be seen that sequence A was completely learned. But very early on, as sequence B is learned, the network's memory of sequence A is overwritten. By five epochs after beginning to learn the sequence B (not shown in figure 4(b)), the network gets an average of 40% of the units of the items of sequence A wrong. By 250 epochs, the network's performance on sequence A is essentially no better than chance and, by 450 epochs, sequence A is, to all intents and purposes, completely forgotten. In short, there is a severe catastrophic forgetting of sequence A. To ensure that our measure of catastrophic forgetting was not due to the use of an overly severe criterion of correctness, after learning sequence B, sequence A was checked with a test criterion of 0.5 (that is, checking whether computed outputs were at least on the correct side of 0.5). Even with this weak criterion, the results still showed severe catastrophic forgetting.

4.5. *Simulation 2: catastrophic interference is not overcome with pseudo-sequences*

Using pseudo-*sequences* to refresh a sequence-learning network is an intuitively plausible idea to solve the problem of catastrophic forgetting, being directly analogous to the use of pseudopatterns to refresh a network learning new static patterns. The idea is that, after the network had learned sequence A, one would generate a number of pseudo-sequences that, hopefully, would reflect sequence A. These pseudo-sequences would be used to refresh the network memory when it learnt a new sequence B. We tested this procedure and found that, contrary to expectations, it does not work.

First, NET 1 (an SRN) completely learned sequence A. Then NET 1 generated 10^4 pseudo-sequences, PS, that were used to train an identical SRN, NET 2. To create



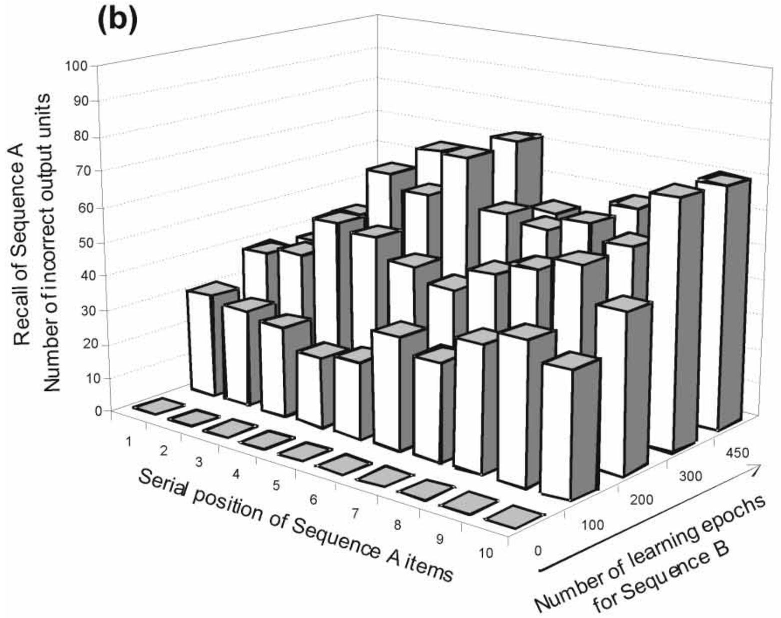


Figure 4. (a) Learning of sequence *B* (after having previously learned sequence *A*). By 450 epochs (an epoch corresponds to one pass through the entire sequence), sequence *B* has been completely learned. Note that it is more difficult to learn the two 'ambiguous' target items, *S*(2) and *S*(6). (b) The number of incorrect units for sequence *A* during learning of sequence *B*. After 450 epochs, the SRN has, for all intents and purposes, completely forgotten the previously learned sequence *A*. (Note that for the sake of readability of the graphs, the learning epochs increase from left to right in the second graph in the direction of the arrow.)

each pseudo-sequence we fed a random input $PS(0)$ through the network, collected the target output $PS(1)$, then input $PS(1)$ through the network to produce $PS(2)$ on output, then input $PS(2)$ to the network, etc. This produced a pseudo-sequence $PS = \{PS(0), PS(1), \ldots, PS(10)\}$. When the random $PS(0)$ was input into the network, the context units were initialized with the neutral context. For the subsequent pseudo-sequence inputs PS the context units contained, as usual, a copy of the hidden-unit vector $H(t-1)$ associated with the previous item in the sequence. Then, when sequence B was being learned in NET 1, after each sequence learning epoch, NET 1 did a learning epoch for a pseudo-sequence generated by NET 2. As B was being learned, we monitored NET 1 to see if its performance on A had improved compared with a situation where no pseudo-sequences had been used. It had not, as shown in figure 5. In other words, the pseudo-sequence technique did not reduce the catastrophic forgetting of the previously learned sequence. In fact, the performance of the network on recalling sequence A was considerably worse when pseudo-sequences were used.

In explaining the failure of the pseudo-sequence technique, it is important to note that any given random input vector has no reason to be close to the initial item of the previously learned sequence. Clearly, the pseudo-sequence mechanism would be effective were the network to be given the real initial item of the originally learned sequence. This would, in fact, amount to a true refreshing process. However, this would require a separate dedicated memory that would be able to supply a reasonable approximation of the initial item of the previously learned sequence (or sequences). However, even if one could design such a system, it would still have no way of knowing *a priori* the *length* of the originally learned sequence(s). Consequently, it could not know *how many* sequence pseudo-elements it would have to generate to create pseudo-sequences reflecting the previously learned sequences. This problem also arose in the present simulation for which there was no theoretical justification for setting the pseudo-sequence length to the length of the original sequence A.

In this paper we do not rely on *ordered* series of pseudopatterns and, as a consequence, there is no problem with initial items or sequence lengths. Most importantly, our method of generating and interleaving *individual* pseudopatterns (as opposed to sequences of pseudopatterns) allows the self-refreshing system to work with remarkable flexibility, as will be shown in section 4.8.

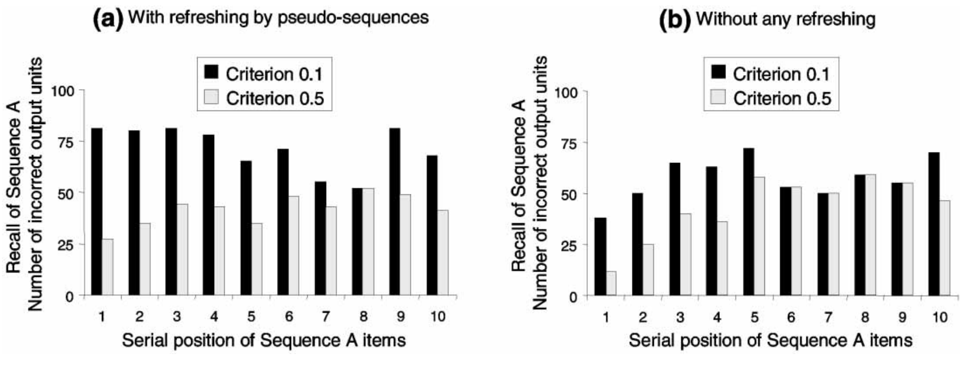


Figure 5. Recall performance for the first sequence A, once the second sequence B is completely learned in a SRN, with (a) and without (b) the pseudo-sequences refreshing. Whatever the learning criterion may be, it appears clearly that refreshing by pseudo-sequences in no way reduces catastrophic forgetting.

4.6. *Simulation 3: catastrophic forgetting is overcome with pseudopatterns*

A RSRN dual-network architecture was used with the parameters indicated in section 4.2. First, both networks are initialized to random-weight settings as described above. NET 1 then perfectly learns sequence *A*. This primary network (NET 1) then generates 10^4 pseudopatterns in order to transfer this learning to the secondary network (NET 2). (Given the size of input layer of each network (150 units), there are 2^{150} possible distinct states for the input layer and, hence, there is a very little probability that the random binary vectors used to produce the 'attractor pseudo-input' (see section 2.7) will be actual input patterns that the network has already seen.)

Now, NET 1 begins to learn sequence *B*. After each learning epoch (consisting of the entire sequence of items in *B*), NET 1 receives 10 pseudopatterns from NET 2 and does one feedforward-backpropagation pass for each of them. The use of 10 pseudopatterns allows a direct comparison with the results of simulation 2 in which self-refreshing was done with pseudo-sequences of length 10. Otherwise, the number of pseudopatterns is not related to the length of the previously learned sequences and can be varied.

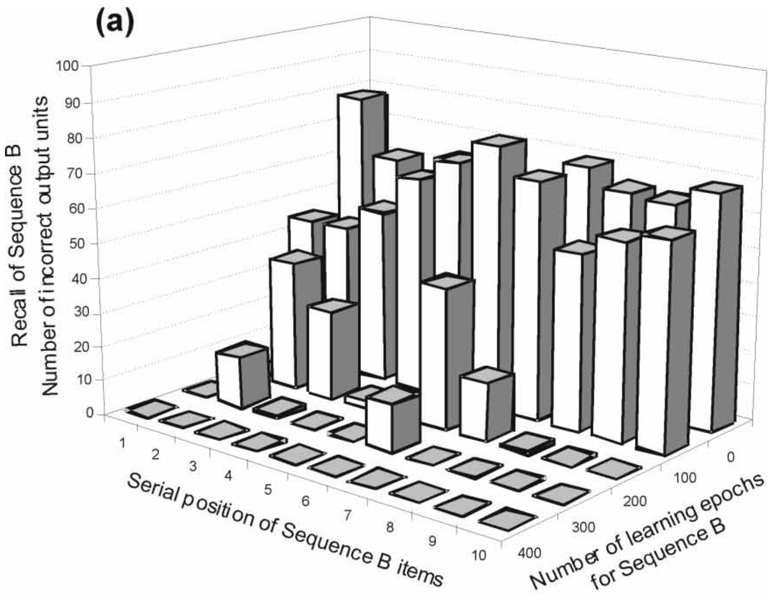
Figure 6(a) shows that the NET 1 does, in fact, learn sequence *B* completely by 400 epochs. In other words, for all items in sequence *B*, all of the units in the network output are within 0.1 of the desired target output. Notice that the sequence items *B*(2) and *B*(6) are learned more slowly by the network. This was, of course, expected, since these two items are preceded by identical items, hence creating ambiguity, having to be solved by the temporal context. During learning of sequence *B*, we tested the performance of the network on sequence *A*. Figure 6(b) shows that *there is virtually no forgetting of sequence A as the network learns sequence B*. In short, catastrophic forgetting has been overcome.

4.7. *Simulation 4: learning a sequence by learning sub-sequences*

The method introduced in the previous section can now be applied to learning long sequences (poems, songs, lists of instructions, etc.) by breaking them into shorter, more manageable sub-sequences. In this simulation we shall show how the dual-network RSRN model solves the problem of learning a whole sequence by learning its successive component parts.

We begin with a sequence of 21 ordered patterns, denoted by *S*(0), *S*(1), *S*(2), . . . , *S*(20). These sequence items were constructed as in the previous simulations. First, 21 distinct vectors with 100 random binary components (0 or 1 with probability 0.5) were created. We made two pairs of items identical, $S(3) = S(6)$, and $S(15) = S(19)$. The sequence *S* was next divided into two sub-sequences: *C*, consisting of the first consecutive items *S*(0) to *S*(10) (with two repeated items *S*(3) and *S*(6)); and *D*, consisting of the remaining consecutive items *S*(10) to *S*(20) (with two repeated items *S*(15) and *S*(19)). Note that the two sub-sequences also have item *S*(10) in common. Sub-sequence *C* ends with this item, while sub-sequence *D* starts with this same item. The role of this 'link' item is to attach the two sub-sequences when the whole sequence has to be recalled after sequential learning of the two sub-sequences separately (that is, learning *C* and then learning *D*).

Using the same network parameters as in simulation 3, we start from an initial state where the primary network, NET 1, has already perfectly learned sub-sequence *C*. If sub-sequence *D* were now learned without the self-refreshing mechanism, it would catastrophically interfere with sub-sequence *C*. However, with self-refreshing in a dual-network RSRN setting using attractor pseudopatterns, catastrophic interference of sub-sequence *C* by sub-sequence *D* is eliminated (as in simulation 3).



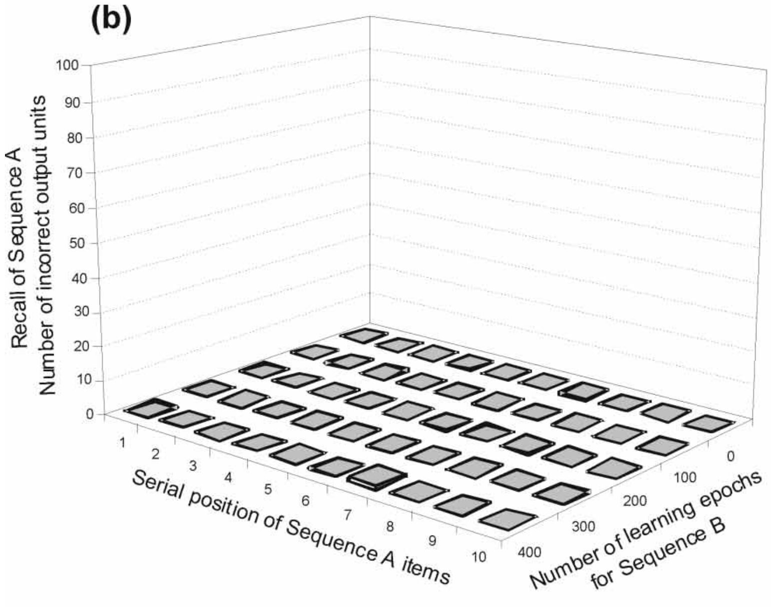


Figure 6. Recall performance for sequences *B* and *A* during learning of sequence *B* by a dual-network RSRN. (a) By 400 epochs, the second sequence *B* has been completely learned. Note that it is more difficult to learn the two 'ambiguous' target items. (b) The previously learned sequence *A* shows virtually no forgetting. Catastrophic forgetting of the previously learned sequence *A* has been completely overcome.

Given the results of simulation 3, it is not particularly surprising that sub-sequence concatenation as described here would work. What is interesting to note, however, is that when sub-sequence *D* is learned by the network, the context units do not contain the context (i.e. the hidden layer representation) from the last item of sequence *C* that was previously learned by the network. Rather, the context units are simply reinitialized to the 0.5 neutral values before the network begins to learn sub-sequence *D*. The network, *in spite of the change of the temporal context for the linking item*, is able to concatenate correctly the two sub-sequences to self-produce the full sequence of 20 ordered items, starting only from the initializing item *S*(0) and the neutral context, without any other information provided from the environment (see figure 7).

4.8. *Simulation 5: random pseudopattern interleaving*

Here we discuss a number of possible ways of interleaving the pseudopatterns with the new sequences to be learned so as to generalize the self-refreshing mechanism and to show its robustness and, in so doing, extend its neural and cognitive validity. Up to this point, pseudopatterns were inserted only *after* each learning epoch of the new sequence. Now we shall explore two other ways in which pseudopatterns reflecting previously learned sequences can be inserted randomly *within* the new sequence as it is being learned. The motivation for this was that, in most cases, there is little or no *a priori* reason that the system would 'know' that a sequence-learning epoch had just been completed and that it was time to generate some pseudopatterns. We therefore felt that it was necessary to investigate whether this constraint can be removed. It turns out that it can.

Normally, NET 1 learns sequence *A*, as defined in simulation 3, and this information is transferred to NET 2 by means of pseudopatterns emanating from NET 1 and learned by NET 2. NET 1 then begins to learn the new sequence *B*. Then NET 1 does one feed-forward-backpropagation pass for each of the items of sequence *B* and, only then, does

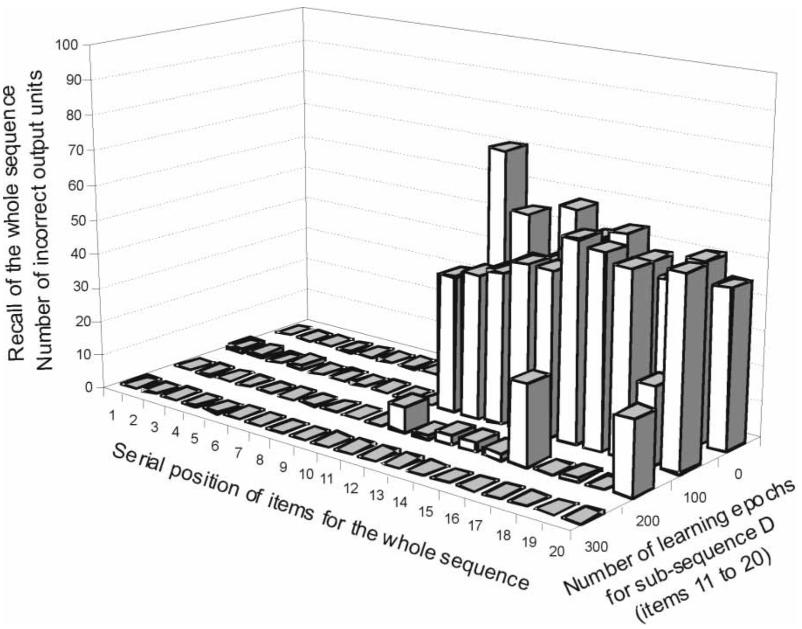


Figure 7. Recall performance for the whole sequence in the course of learning of its second sub-sequence *D* within a RSRN dual-architecture. By 300 epochs, the second sub-sequence *D* has been completely learned. The previously learned sub-sequence *C* shows no forgetting and the whole sequence of 20 ordered items can be perfectly reproduced when starting only from the initializing item *S*(0) and the neutral context. The two separately learned sub-sequences *C* and *D* were correctly linked.

a feedforward-backpropagation pass for each of the pseudopatterns generated by NET 2. This interleaving method is called the 'between-sequences' procedure. But as we said above, it could be argued that this strict alternation of real sequence learning and pseudopattern learning lacks flexibility. Thus, for the following simulations, we decided to interleave the pseudopatterns *within* sequence B, that is, between items forming the sequence being learned, rather than wait until a learning epoch for sequence B was completed. We call this method the 'within-sequence' procedure.

One learning epoch for sequence B in NET 1 normally consists of one feedforward-backpropagation pass for each of the 10 successive input-target patterns, $[B(0) \to B(1)]$ $[B(1) \to B(2)]$ $[B(2) \to B(3)] \cdots [B(9) \to B(10)]$. (Note: with this notation the current temporal context, designated in figures 2 and 3 by $H(t-1)$, that is systematically included in each input is omitted.) Now, we assume that a pseudopattern generated by NET 2, Ψ: $i_\psi \to o_\psi$, may be interleaved within a learning epoch of sequence B in one of two ways. In the first interleaving mode of the within-sequence procedure, called the 'insert mode', a fixed number of pseudopatterns, denoted M, may be *inserted* between two successive input-target patterns of the sequence, for example, taking $M = 3$:

$$\cdots[B(2) \longrightarrow B(3)][i_{\psi 1} \longrightarrow o_{\psi 1}][i_{\psi 2} \longrightarrow o_{\psi 2}][i_{\psi 3} \longrightarrow o_{\psi 3}][B(3) \longrightarrow B(4)]\cdots$$

Note that in this mode the links of the sequence are not 'broken' ($B(3)$ in the example above), even though the current temporal context is. It is, indeed, the hidden layer activations induced by the last occurring pseudopattern that will constitute the contextual input for the first real pattern learned after pseudopattern insertion. In the second mode, called the 'replace mode', the M pseudopatterns may *replace* a number of input-target patterns that are therefore lost for the current epoch, for example, taking $M = 2$:

$$\cdots[B(2) \longrightarrow B(3)][i_{\psi 1} \longrightarrow o_{\psi 1}][i_{\psi 2} \longrightarrow o_{\psi 2}][B(5) \longrightarrow B(6)]\cdots$$

Here, the two input-target patterns $[B(3) \to B(4)]$ and $[B(4) \to B(5)]$ are replaced by two pseudopatterns and are, therefore, lost for the current learning epoch. Both the sequence and the temporal context are broken.

For these two within-sequence modes, pseudopatterns are generated stochastically. This means that in the course of one learning epoch of sequence B, with a given probability p, a pseudopattern 'gate' may be opened after each association $[B(i) \to B(i+1)]$. When the gate is opened, a 'burst' of M pseudopatterns is interleaved at that point in sequence B. For each simulated interleaving mode, we varied p between 0.1 and 1, in two separate conditions: pseudopattern bursts of size $M = 1$ and bursts of size $M = 5$. Simulations were replicated five times. Table 1 shows the performance of the different self-refreshing procedures for the two within-sequence methods (insert and replace modes) and for additional simulations using the original between-sequences procedure (as used in simulation 3). To allow the various procedures to be compared, the network performance was given as a function of the mean number N of interleaved pseudopatterns per learning epoch of sequence B. For the within-sequence cases, this number is a consequence of the burst probability p, which was explicitly chosen in order to maintain the same mean number N for each column. For the original between-sequences method, N corresponds, of course, to a fixed number, not to a mean across learning epochs and replications.

These results demonstrate that all self-refreshing procedures continue to work with substantial variations in the mean number N of pseudopatterns that are interleaved

Table 1. Forgetting of sequence *A* after complete learning of sequence *B* using different self-refreshing procedures.

	Mean number (*N*) of interleaved pseudopatterns per learning epoch of *B*						
Self-refreshing procedures	0 (i.e. AARN)	1	3	5	7	9	10
PP within sequence							
Insert mode							
M = *1 PP per burst*							
Probability (p) of the burst		0.10	0.30	0.50	0.70	0.90	1
Number of learning epochs on *B*	450	447	718	1178	2188	UnL	UnL
Number of incorrect units on *A*	575.0	28.5	9.0	3.8	3.3	—	—
M = *5 PP per burst*							
Probability (p) of the burst		0.02	0.06	0.10	0.14	0.18	0.20
Number of learning epochs on *B*	450	391	460	522	562	617	696
Number of incorrect units on *A*	575.0	38.8	4.2	2.2	0.6	1.8	0.0
Replace mode							
M = 1 *PP per burst*							
Probability (*p*) *of the burst*		0.10	0.30	0.50	0.70	0.90	1
Number of learning epochs on *B*	450	539	910	2088	26 945	UnL	UnL
Number of incorrect units on *A*	575.0	26.6	6.4	0.6	8.0	—	—
M = 5 *PP per burst*							
Probability (*p*) *of the burst*		0.02	0.06	0.10	0.14	0.18	0.20
Number of learning epochs on *B*	450	427	427	533	645	758	837
Number of incorrect units on *A*	575.0	29.6	6.6	1.6	0.4	0.0	0.8
PP between sequences (original method)							
Number of learning epochs on *B*	450	380	360	404	380	401	391
Number of incorrect units on *A*	575.0	29.0	2.6	1.2	2.0	0.2	0.4

UnL: at least in one of the five replications the network fails to reach the 0.1 learning criterion by 100 000 epochs.

per learning epoch. Note that the procedure fails only when the interleaving probability p is close to one, i.e. when the context is systematically broken in the insert mode or when the sequence to be learned is almost never presented in the replace mode. For all the other cases, catastrophic forgetting is prevented. Even when using a single pseudopattern per epoch, forgetting is still acceptable since only about 30 units are incorrect out of the 1000 units involved in coding a whole sequence (i.e. 10 items in a sequence and 100 units to code each item). The difference among the procedures derives mainly from the number of epochs required to learn the second sequence *B*. In this respect the between-sequence procedure (used in simulations 3 and 4) seems to be the most effective.

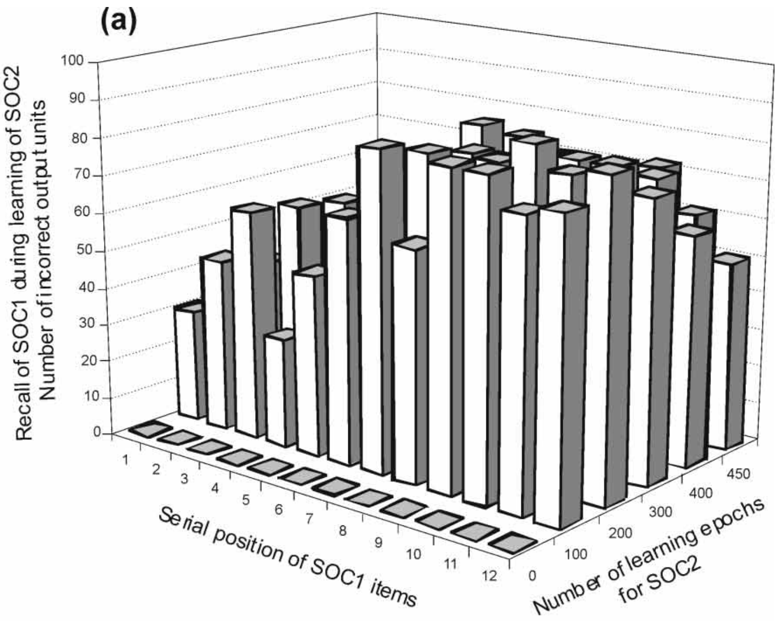
In sum, regardless of the interleaving procedure, refreshing by pseudopatterns is effective. At a neural or cognitive level, the robustness of the self-refreshing mechanism allows us to choose from any of the proposed procedures, or even from a mixture of them, depending on the constraints set by the domain/task.

4.9. *Simulation 6: challenging the RSRN dual-network with complex sequences*

To check at a deeper level the power of the RSRN self-refreshing mechanism, the dual-network was faced with a class of complex sequences, second-order conditional sequences (or SOCs, Cohen *et al.* 1990, Reed and Johnson 1994), in which no single element in the sequence can predict its successor. The complexity of these sequences derives from their structure consisting of four distinct items appearing in an order such that every item is immediately followed by one of the three other possible items with equal frequency. A single item cannot by itself predict the following item better than chance. In other words, two consecutive items are always required to predict the next one. These 'tailor-made' ambiguous sequences, used in a specific paradigm (Reed and Johnson 1994, Shanks and Johnstone 1999, Destrebecqz and Cleeremans 2001) of the implicit learning literature, were used here only as a practical means to challenge the proposed self-refreshing mechanism.

We created two SOC sequences, $SOC1 = abcadcdbacbda$ and $SOC2 = efgehghfegfhe$, in which component items were, as in previous simulations, distinct vectors with 100 random binary components (0 or 1, with 0.5 probability). In agreement with the definition of this type of sequence, every item is followed by one of the three other possible items. For example, in *SOC*1, *c* is followed by *a* or *b* or *d* (ambiguity), and two successive items are required to determine the subsequent item, in this case: $(b)c \rightarrow a$, $(a)c \rightarrow b$, $(d)c \rightarrow d$ (the ambiguity is resolved by the predecessor of item *c*). Learning this temporal structure in a RSRN (or a SRN) therefore gives a predominant and permanent role to the current temporal context.

A RSRN dual-network with the same parameters and procedures as in simulations 3 and 4 was used to check the effectiveness of the self-refreshing mechanism in suppressing catastrophic forgetting in the consecutive learning of two SOC sequences, in this case, *SOC*1 and *SOC*2. As before, we start from an initial state where the primary network, NET 1, has already perfectly learned *SOC*1. Then NET 1 learns *SOC*2 either with or without the self-refreshing mechanism at work. In this latter case, as expected, severe catastrophic forgetting occurs for the previously learned *SOC*1. This occurs almost as soon as the network begins to learn *SOC*2 (see figure 8(a)). In contrast, when *SOC*2 is jointly learned with pseudopatterns from NET 2 (using the same between-sequence interleaving procedure as in simulations 3 and 4), there is *virtually no forgetting of SOC*1 (see figure 8(b)). Hence, the ability of the self-refreshing mechanism to overcome catastrophic forgetting in learning complex temporal structures appears to be as good as in previous simulations on simpler temporal structures.



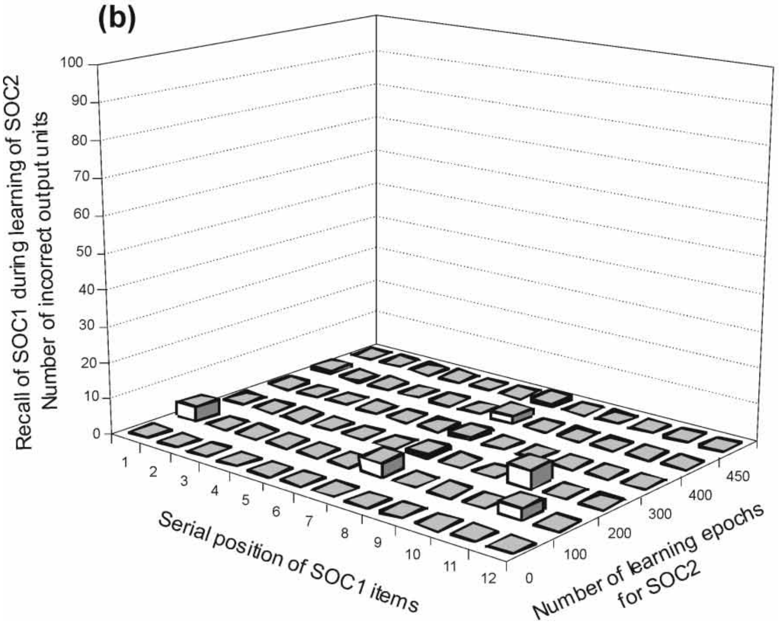


Figure 8. Recall performance for the previously learned *SOC*1 sequence during learning of a second *SOC*2 sequence (completely learned by 450 epochs). The two SOCs are made up of 13 items and, as in previous simulations, the item in position 0 is not shown because it is used only to initialize sequence learning and recall. (a) *Without self-refreshing*, catastrophic forgetting is severe. (b) *With self-refreshing*, the previously learned *SOC*1 sequence does not show any catastrophic forgetting during *SOC*2 learning.

However, it should be noted that for the transfer phase from NET 1 to NET 2, five times more pseudopatterns from NET 1 were used than in previous simulations on simpler sequences. This seems to be the price to pay for higher complexity.[1]

5. Experimental data and simulations

Up to this point, our main objective has been to demonstrate, by means of simulations, the power of the self-refreshing mechanism in maintaining the stability of previously learned sequences. As the initial goal of the dual-network model was to support the simple fact that humans do not forget catastrophically, we then conducted a more detailed comparison of human sequence retention and our RSRN model. The experiment conducted is reminiscent of the classic forgetting experiments carried out by Barnes and Underwood (1959), with the crucial difference being that, instead of two lists of non-word–word paired associates, we used ordered series of characters.

5.1. *Experiment*

5.1.1. *Method.* *Participants.* Thirty-five undergraduate students, aged 18–39 years, participated in the experiment for extra course credit. They were randomly assigned to one of four experimental groups. Three of them were not retained for the final part of the experiment since they failed to complete the first learning phase (see 'Design and procedure') in less than 50 min. *Stimuli and apparatus.* The items were characters presented sequentially in upper case. Four sequences were used to *measure* retroactive interference and were learned during a first phase of the experiment. Each of these sequences, [$W \rightarrow 5 \rightarrow N \rightarrow G \rightarrow P$], [$Z \rightarrow H \rightarrow 5 \rightarrow R \rightarrow B$], [$X \rightarrow G \rightarrow 8 \rightarrow C \rightarrow J$] and [$Q \rightarrow L \rightarrow R \rightarrow K \rightarrow 8$], shared at least one character with one of the others. Another sequence, [$S \rightarrow T \rightarrow D \rightarrow B \rightarrow 8 \rightarrow L \rightarrow J \rightarrow F \rightarrow V$], was used to *produce* retroactive interference in the final phase of the experiment. This sequence was longer in order to study finely the time course of interference effects. The experiment was run on a PC computer using EPrime V1.1.4.1. Each character was presented in the centre of a 17-inch monitor (800×600 resolution) in Courier New-48 points font.

Design and procedure. Pilot experiments indicated that learning four short sequences of characters was in fact quite difficult for most participants when they have to produce delayed recalls. On the basis of these pilot experiments, we designed a learning procedure that ensured, for most participants, appropriate learning of the four sequences in a reasonable amount of time (i.e. 35–50 min). The resulting design is quite complex, because of the necessity, during learning, both to discourage participants from simply relying on short-term memory and to encourage them by giving appropriate feedback.

The learning phase of the four sequences was organized in five sub-phases. In the first sub-phase each sequence was presented starting with a display indicating the first character for 1s followed, after a 250 ms interval, by the next character for 1 s, and so on until the end of the sequence. After each presentation of a whole sequence an immediate recall was required: Its first character was presented followed by a question mark and the participant had to type the next character on the keyboard and validate it by pressing the Enter key. Feedback was given and the correct character appeared on the screen before the next question mark. During this first sub-phase there were two blocks of sequence learning, a block consisting of the presentation of all sequences in a random order. The second sub-phase was aimed at giving the participants some insight into the actual difficulty of the task by asking simply for a recall of each sequence, but without a prior presentation immediately before the test presentation. The third sub-phase was a replication of the first one. In the fourth sub-phase the feedback was modified so that the correct response did not appear on the screen, errors being indicated only by a red exclamation mark. Moreover, before each sequence recall, participants had to solve an addition problem involving two whole numbers lying between three and 49. This calculation was required to prevent participants from relying mainly on their short-term phonological loop (Baddeley and Hitch 1974) during learning. After a display indicating a forthcoming calculation, each term of the addition appeared for 1 s, separated by a 250 ms interval, and the participants had to type the response on the keyboard. This latter sub-phase involved four blocks of sequence learning. The fifth sub-phase consisted of recalls of the four sequences from their initial characters, each recall being preceded by a calculation. If a participant made an error during this test, the four sequences were presented again, and the whole recall test was repeated until complete success over all four sequences was achieved.

After a 5-min rest period participants underwent 20 learning trials of the new sequence of length nine. At various times during this learning, participants were asked to recall one of the four previously learned sequences followed by a recall trial of the new sequence. Each recall trial was preceded by a mental arithmetic problem as above. These recall trials occurred after one, five, 10 and 20 learning presentations of the new sequence. The measure of interest was the evolution of performance on the previously learned sequences as learning of the new sequence progressed. Performance was measured not by the number of successive correct characters produced, but simply by counting the number of correct characters recalled in their correct position within a sequence. This indicator corresponds to a general recall performance even for participants who, for example, make an error in the middle of a sequence while being correct at the end of it. Via a Latin square design, we ensured that, across participants, each of the four previously learned sequences was tested the same number of times at each of the four recall moments (i.e. after one, five, 10 and 20 learning presentations of the new sequence). Our experimental design produced four groups of participants according to the test order of the four previously learned sequences in this final phase. This ensured that the results could not be attributed to any intrinsic characteristics of a particular previously learned sequence but, rather, would be dependent on the moment during learning of the new sequence at which the initially learned sequence was tested.

5.1.2. *Results*. For both new and previously learned sequences, the per cent of sequence items correctly recalled is shown in figure 9, both being a function of the number of presentations of the new sequence. The first thing to note is that a high degree of learning of the new sequence (i.e. 99% of correct items after 20 learning trials [$F(3, 84) = 69.34$; $MSe = 0.02013$; $p < 0.000001$]) does not produce catastrophic forgetting, only retroactive interference. In other words, mean performance on the previously learned sequences is invariably above 60%. While this is obviously below the perfect learning level exhibited immediately after the first learning phase, participants' performance is, nevertheless, quite good if we consider that participants had to perform serial recalls. We can also note that there is a significant global effect of the moment of testing of the previously learned sequences in amount of retroactive interference [$F(3, 84) = 7.91$; $MSe = 0.06$; $p < 0.0001$]. More importantly, there is an *initial drop* in performance until the second test of the previously learned sequences [$F(1, 28) = 11.57$; $MSe = 0.06$; $p < 0.002$] (recall that the first and second tests correspond respectively to the first and fifth learning presentations of the new sequence) *followed by an improvement* in recall performance between the second test and the third test of the previously learned sequences [$F(1, 28) = 17.57$; $MSe = 0.05$; $p < 0.0001$] (i.e. between the fifth and the tenth learning trial of the new sequence). These effects are in no way linked to intrinsic

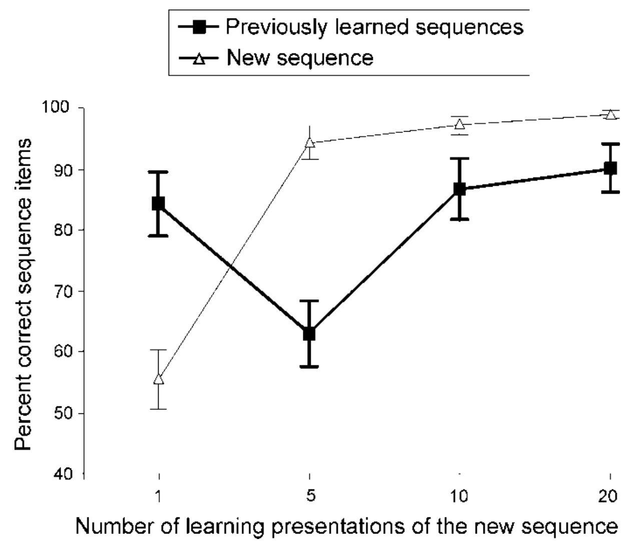


Figure 9. Recall performance of the new sequence and of the previously learned sequences during learning of the new sequence. Vertical bars denote standard errors.

characteristics of a particular previously learned sequence since there is neither group effect [$F(1, 28) = 0.35$; $MSe = 0.12$; $p = 0.78$] nor interaction between this group factor and the moment of test [$F(9, 84) = 1.15$; $MSe = 0.07$; $p = 0.16$].

5.2. *Simulated experiment*

The previous experiment was simulated using a dual-network RSRN with the same architecture (i.e. input items and hidden layer of length 100 and 50, respectively), parameters and procedures as in the basic simulations 3, 4 and 6. We created five sequences (four of length 5 and one of length 9) in which component items were, as in previous simulations, vectors with 100 random binary components (0 or 1, with 0.5 probability). These sequences are similar to those used in the experiment in so far as the serial order of the constituent items is the same. The primary network (NET 1) completely learned the first four sequences (of length 5) in an interleaved way as in the experiment. Next NET 1 generated 10^4 pseudopatterns, the same number as in previous basic simulations (except in simulation 6 where this number was greater) in order to transfer this learning to the secondary network (NET 2). NET 1 then learned the fifth sequence (of length 9) either with, or without, the self-refreshing mechanism at work. During learning of this new sequence, the performance of the network was tested on the four previously learned sequences in a way that corresponds to the experimental tests. When we tested the amount of forgetting of the previously learned sequences, for each item making up a given old sequence, we calculated the root mean squared error (RMS) between the expected item and the output pattern computed by the RSRN network. Only when this RMS was less than 0.1 was the network output considered to be correct for the corresponding sequence item. As an indicator of the global performance of the network, we computed the percentage of correct items over the four previously learned sequences.

As in the experiment above, these tests were performed after different numbers of presentations of the new sequence until learning of it was completed (150 presentations). The entire learning and test procedure was replicated 10 times, each time starting from a new set of random initial weights for the two RSRN networks, with and without the self-refreshing mechanism at work. For these two cases, the network performance on the four previously sequences (per cent correct sequence items, averaged over 10 replications) is shown in figure 10 as a function of the number of presentations during the learning of the new sequence.

5.3. *Discussion*

As expected, severe catastrophic interference occurs in the simulation without the self-refreshing mechanism. Further, with refreshing, there is virtually no interference with the previously learned sequences after learning of the new sequence is completed (after 150 presentations), as was the case with humans (see figure 9). It is, indeed, particularly interesting to compare the patterns of retroactive interference obtained in the experiment (see figure 9) and in the simulation (see figure 10, with self-refreshing). In particular, during learning of the new sequence, there is the same initial drop in recall of the previously learned sequences, followed by the same subsequent rise in performance. From the point of view of the dual-network RSRN architecture, this ‘early drop and subsequent rise’ in recall of old information when learning new information is due to the restructuring of the connection weights. These weights, initially shaped by the previously learned sequences, have to adapt suddenly to a new sequence, which induces retroactive interference, the refreshing process not yet having had enough time to ‘compete’ with the new information coming from the environment. Then, as learning of the new sequence proceeds, NET 1 is exposed to more and more internally generated pseudopatterns from NET 2 (reflecting the previously learned sequences). This thus explains why there is then a rise in the recall performance of the previously learned sequences. There is a striking similarity of the patterns of results obtained experimentally and in the connectionist architecture made of basic RSRN modules for which we are arguing.

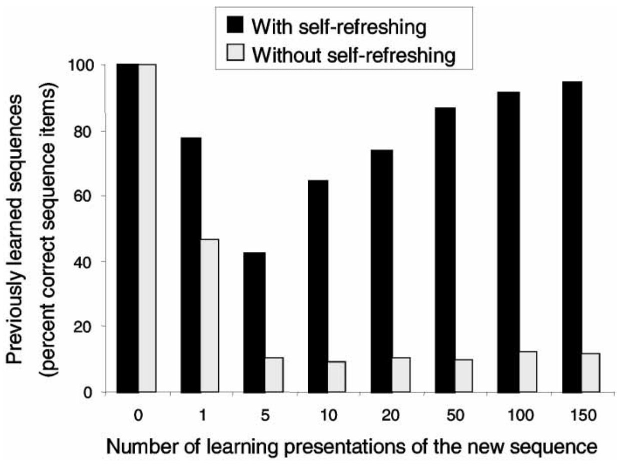


Figure 10. Recall performance, with and without the self-refreshing mechanism at work, of the previously learned sequences during learning of the new sequence (which is completed after 150 presentations). Without refreshing, there is clearly catastrophic forgetting of the previously learned sequences. With refreshing, however, the learning curve exhibits, as for humans, an initial drop and subsequent rise in recall performance.

This type of coupled recurrent-network architecture might seem to be related to the general class of 'chaining' models that generally provoke some scepticism (see Page 2000). There are, however, crucial differences. In particular, chaining predicts that once an error is made the rest of the sequence will be lost, which is clearly not how humans process sequences. Indeed, in the behavioural experiment, this phenomenon was not observed: during recall tests, one wrong character produced by the participants in, for instance, the middle of a sequence, did not systematically cause the loss of all subsequent items. It must be emphasized, however, that in our simulation of the multiple sequence-learning experiment (and in other previous simulations), the RSRN architecture, like people, was also able to make an error in some serial position without losing the rest of a sequence. In other words, criticisms based on the claim that recurrent networks are equivalent to simple chaining models are probably misguided, at least in the present dual-network RSRN framework (see also Botvinick and Plaut 2004, who demonstrate, in comparing recurrent and hierarchical models of sequence learning, inadequacies with this particular criticism of recurrent models).

6. Concluding remarks

We have shown that the reverberating dual-network architecture, originally proposed earlier to overcome catastrophic forgetting in learning non-temporally ordered patterns (Ans and Rousset 1997, 2000), could be generalized to the learning of successive temporal sequences. The basic principle of interleaving internally generated pseudopatterns from a long-term storage network with patterns from the environment being learned by a second network has been developed elsewhere (Ans and Rousset 1997, 2000, French 1997).

The distinction between successively learning individual patterns of input–output associations and learning multiple sequences of patterns is an important one. It turns out that there is no need to interleave pseudo-sequences with the new sequences to be learned—interleaving large numbers of simple input–output pseudopatterns, which collectively reflect *the entire previously learned sequence(s)*, is what overcomes catastrophic forgetting of the previously learned sequence(s). In fact, pseudo-sequences produced by NET 2 generated from random input to that network bear no resemblance whatsoever to originally learned sequence(s) and are of no use in overcoming catastrophic forgetting.

In particular, we wish to emphasize the power of a network architecture that integrates an autoassociative part with a standard heteroassociative architecture. This type of network design allows us to produce 'reverberated' pseudopatterns that are attractors of the entire network and that collectively reflect the previously learned sequences. As for the brain correlates of the self-generated pseudopatterns, Robins and colleagues (Robins 1996, Robins and McCallum 1999) have suggested that they could be related to spontaneous cortical reactivations observed during different sleep phases, in particular rapid eye movement (REM) sleep. This hypothesis is strengthened by human experiments using position emission tomography (PET) imaging techniques that show reactivations during REM sleep of the cortical regions that were previously activated during the execution of a serial reaction time task during the wake state (Maquet *et al.* 2000). However, to account for our results involving multiple sequence learning, we suggest that the pseudopattern information exchange mechanism is also active during awake phases, but to a lesser extent because of the greater levels of activity generated by input from the external environment.

Further research will study the self-refreshing mechanism applied to other types of temporal structure, such as Reber's artificial grammars (e.g. Reber 1967, Cleeremans 1993). In particular, it would be particularly interesting to study how knowledge of an artificial grammar in one perceptual domain can be transferred to a different perceptual domain. This problem was addressed by Dienes *et al.* (1999) using a modified version of the SRN network. Once a given artificial grammar was learned in one domain (e.g. sequences of tones differing in pitch), the weights of one of the two hidden layers of their architecture (the 'core' hidden layer) were then 'frozen' and the performance of the same grammar when trained in a different domain (e.g. sequences of letters) was observed. This model displayed performance similar to that observed in humans, and the freezing mechanism, which made knowledge transfer possible while avoiding catastrophic forgetting, was interpreted by the authors as simulating a dynamic learning algorithm in which the learning rate was reduced as learning improved. However, using the same architecture in the context of previous simulations (Altmann and Dienes 1999) on infant data, Altmann (2002) has recently proposed a very interesting way (based on pre-training on prior experience) in which the freezing process can be avoided (see also the recent work of Hanson and Negishi 2002, where it has been shown that transfer in a recurrent network does not require weight freezing). It is likely that a self-refreshing mechanism similar to the one described here would allow us directly to avoid the weight freezing process and to obtain transfer between domains.

This is but one of numerous paths that could be explored with self-refreshing reverberating dual-network models. These models may well be applied to areas where sequential learning of input, whether simple patterns or sequences, long or short, musical or visual, whether learned explicitly or implicitly, is important.

To demonstrate how the catastrophic forgetting problem can be solved during multiple sequence learning, we chose the SRN (Elman 1990) as our starting point simply because this is the most commonly used network in the sequence-learning literature. This model relies on the backpropagation learning algorithm, widely considered not to be biologically realistic. However, it has been shown that the deterministic version of the contrastive Hebbian learning algorithm, or CHL (Peterson and Anderson 1987, Hinton 1989b), is formally equivalent to backpropagation (Xie and Seung 2003) and that CHL could be directly derived from backpropagation (O'Reilly 1996). A crucial feature of CHL is its biological plausibility as it is a Hebbian-type learning algorithm, relying on only local pre- and post-synaptic activities locally available. The equivalence between backpropagation and CHL is fundamental because it implies at least the potential biological validation of an extensive body of research using feedforward-backpropagation models to explore and explain certain cognitive phenomena. In any way, the plasticity-stability problem in connectionist models of learning is a drawback inherent to the entire class of gradient descent learning algorithms in fully distributed systems and the present study proposes an effective and rather general solution to this major problem.

Acknowledgements
This research was supported in part by a research grant from the European Commission (HPRN-CT-1999-00065) and by the French government (CNRS UMR 5105).

Note
1. When the same number of pseudopatterns as in previous simulations (i.e. 10^4) is used during transfer from NET 1 to NET 2, the retroactive interference on the *SOC*1 sequence remains acceptable (root mean squared error, calculated over the whole sequence, equal to about 0.05). However, a higher number of pseudopatterns was used in order to show that, for more complex sequences, the system is able to reach a performance level comparable to that obtained in previous simulations involving a simpler sequence structure.

References

Altmann, G. T. M., 2002, Learning and development in neural networks—the importance of prior experience. *Cognition*, **85**: 43–50.

Altmann, G. T. M., and Dienes, Z., 1999, Rule learning by seven-month-old infants and neural networks. *Science*, **284**: 875.

Ans, B., 1990, Neuromimetic model for storage and recall of temporal sequences. *CR Académie des Sciences Paris, Life Sciences*, **311**: 7–12.

Ans, B., Carbonnel, S., and Valdois, S., 1998, A connectionist multiple-trace memory model for polysyllabic word reading. *Psychological Review*, **105**: 678–723.

Ans, B., Coiton, Y. Gilhodes, J. C., and Velay, J. L., 1994, A neural network model for temporal sequence learning and motor programming. *Neural Networks*, **7**: 1461–1476.

Ans, B., and Rousset, S., 1997, Avoiding catastrophic forgetting by coupling two reverberating neural networks. *CR Académie des Sciences Paris, Life Sciences*, **320**: 989–997.

Ans, B., and Rousset, S., 2000, Neural networks with a self-refreshing memory: knowledge transfer in sequential learning tasks without catastrophic forgetting. *Connection Science*, **12**: 1–19.

Ans, B. Rousset, S. French, R. M., and Musca, S., 2002, Preventing catastrophic interference in multiple-sequence learning using coupled reverberating Elman networks. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society* (Hillsdale NJ: Lawrence Erlbaum), pp. 71–76.

Baddeley, A. D., and Hitch, G., 1974, Working memory. In G. A. Bower (ed.) *Recent Advances in Learning and Motivation*, Vol. 8 (New York: Academic Press).

Barnes, J., and Underwood, B., 1959, 'Fate' of first-list associations in transfer theory. *Journal of Experimental Psychology*, **58**: 97–105.

Botvinick, M., and Plaut, D. C., 2004, Doing without schema hierarchies: a recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review*, **111**: 395–429.

Carpenter, G. A., and Grossberg, S., 1988, The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, **21**: 77–88.

Cleeremans, A., 1993, *Mechanisms of Implicit Learning: Connectionist Models of Sequence Processing* (Cambridge MA: MIT Press).

Cleeremans, A., and Destrebecqz, A., 1997, Incremental sequence learning. In *Proceedings of the 19th Annual Conference of the Cognitive Science Society* (Hillsdale NJ: Lawrence Erlbaum), pp. 119–124.

Cohen, A. Ivry, R., and Keele, S. W., 1990, Attention and structure in sequence learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **16**: 17–30.

Destrebecqz, A., and Cleeremans, A., 2001, Can sequence learning be implicit? New evidence with the process dissociation procedure. *Psychonomic Bulletin and Review*, **8**: 343–350.

Dienes, Z. Altmann, G. T. M., and Gao, S. J., 1999, Mapping across domains without feedback: a neural network model of transfer of implicit knowledge. *Cognitive Science*, **23**: 53–82.

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

French, R. M., 1992, Semi-distributed representations and catastrophic forgetting in connectionist networks. *Connection Science*, **4**: 365–377.

French, R. M., 1994, Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference. In *Proceedings of the 16th Annual Conference of the Cognitive Science Society* (Hillsdale NJ: Lawrence Erlbaum), pp. 335–340.

French, R. M., 1997, Pseudo-recurrent connectionist networks: an approach to the 'sensitivity–stability' dilemma. *Connection Science*, **9**: 353–379.

French, R. M., 1999, Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, **3**: 128–135.

French, R. M. Ans, B., and Rousset, S., 2001, Pseudopatterns and dual-network memory models: advantages and shortcomings. In R. French and J. Sougné (eds) *Connectionist Models of Learning, Development and Evolution* (London: Springer), pp. 1–10.

Grossberg, S., 1987, Competitive learning: from interactive activation to adaptive resonance. *Cognitive Science*, **11**: 23–63.

Hanson, S. J., and Negishi, M., 2002, On the emergence of rules in neural networks. *Neural Computation*, **14**: 2245–2268.

Hebb, D. O., 1949, *The Organization of Behavior* (New York: Wiley).

Hetherington, P., and Seidenberg, M., 1989, Is there ‘catastrophic interference’ in connectionist networks? In *Proceedings of the 11th Annual Conference of the Cognitive Science Society* (Hillsdale NJ: Lawrence Erlbaum), pp. 26–33.

Hinton, G. E., 1989a, Connectionist learning procedures. *Artificial Intelligence*, **40**: 185–234.

Hinton, G. E., 1989b, Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation*, **1**: 143–150.

Hintzman, D. L., 1986, ‘Schema abstraction’ in a multiple-trace memory model. *Psychological Review*, **93**: 411–428.

Jordan, M. I., 1986, *Serial Order: A Parallel Distributed Processing Approach.* Technical Report ICS-8604, University of California at San Diego, CA.

Kanerva, P., 1988, *Sparse Distributed Memory* (Cambridge MA: MIT Press).

Kortge, C. A., 1990, Episodic memory in connectionist networks. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society* (Hillsdale NJ: Lawrence Erlbaum), pp. 764–771.

Kruschke, J. K., 1992, ALCOVE: an exemplar-based connectionist model of category learning. *Psychological Review*, **99**: 22–44.

Kruschke, J. K., 1993, Human category learning: implications for backpropagation models. *Connection Science*, **5**: 3–36.

Lewandowsky, S., 1991, Gradual unlearning and catastrophic interference: a comparison of distributed architectures. In W. E. Hockley and S. Lewandowsky (eds) *Relating Theory and Data: Essays on Human Memory in Honor of Bennet B. Murdock* (Hillsdale NJ: Lawrence Erlbaum), pp. 445–476.

Lewandowsky, S., 1994, On the relation between catastrophic interference and generalization in connectionist networks. *Journal of Biological Systems*, **2**: 307–333.

Lewandowsky, S., and Li, S. C., 1995, Catastrophic interference in neural networks. Causes, solutions, and data. In F. N. Dempster and C. Brainerd (eds) *New Perspectives on Interference and Inhibition in Cognition* (New York: Academic Press), pp. 329–361.

Maquet, P. Laureys, S. Peigneux, P. Fuchs, S. Petiau, C. Phillips, C. Aerts, J. Del Fiore, G., Degueldre, C. Meulemans, T. Luxen, A. Franck, G. Van Der Linden, M., Smith, C., and Cleeremans, A., 2000, Experience-dependent changes in cerebral activation during human REM sleep. *Nature Neuroscience*, **3**: 831–836.

Maskara, A., and Noetzel, A., 1992, Forced simple recurrent neural network and grammatical inference. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society* (Hillsdale NJ: Lawrence Erlbaum), pp. 420–425.

Maskara, A., and Noetzel, A., 1993, Sequence learning with recurrent neural networks. *Connection Science*, **5**: 139–152.

McClelland, J. L. McNaughton, B. L., and O’Reilly, R. C., 1995, Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, **102**: 419–457.

McCloskey, M., and Cohen, N. J., 1989, Catastrophic interference in connectionist networks: the sequential learning problem. In G. H. Bower (ed.) *The Psychology of Learning and Motivation*, Vol. 24 (New York: Academic Press), pp. 109–165.

McRae, K., and Hetherington, P. A., 1993, Catastrophic interference is eliminated in pretrained networks. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society* (Hillsdale NJ: Lawrence Erlbaum), pp. 723–728.

Murre, J. M. J., 1992, The effects of pattern presentation on interference in backpropagation networks. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society* (Hillsdale NJ: Lawrence Erlbaum), pp. 54–59.

O’Reilly, R. C., 1996, Biologically plausible error-driven learning using local activation differences: the generalized recirculation algorithm. *Neural Computation*, **8**: 895–938.

Page, M., 2000, Connectionist modelling in psychology: a localist manifesto. *Behavioral and Brain Sciences*, **23**: 443–467.

Peterson, C., and Anderson, J., 1987, A mean field theory learning algorithm for neural networks. *Complex Systems*, **1**: 995–1019.

Plaut, D. C. McClelland, J. L. Seidenberg, M. S., and Patterson, K., 1996, Understanding normal and impaired word reading: computational principles in quasi-regular domains. *Psychological Review*, **103**: 56–115.

Ratcliff, R., 1990, Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological Review*, **97**: 285–308.

Reber, A. S., 1967, Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behaviour*, **6**: 855–863.

Reed, J., and Johnson, P., 1994,, Assessing implicit learning with indirect tests: determining what is learned about sequence structure. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **20**: 585–594.

Reiss, M., and Taylor, J. G., 1991, Storing temporal sequences. *Neural Networks*, **4**: 773–787.

Robins, A. V., 1995, Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, **7**: 123–146.

Robins, A. V., 1996, Consolidation in neural networks and in the sleeping brain. *Connection Science*, **8**: 259–275.

Robins, A. V., and McCallum, S., 1998, Catastrophic forgetting and the pseudorehearsal solution in Hopfield-type networks. *Connection Science*, **10**: 121–135.

Robins, A. V., and McCallum, S., 1999, The consolidation of learning during sleep: comparing the pseudorehearsal and unlearning accounts. *Neural Networks*, **12**: 1191–1206.

Shanks, D. R., and Johnstone, T., 1999, Evaluating the relationship between explicit and implicit knowledge in a sequential reaction time task. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **25**: 1435–1451.

Sharkey, N. E., and Sharkey, A. J. C. 1995, An analysis of catastrophic interference. *Connection Science*, **7**: 301–329.

Xie, X., and Seung, H. S., 2003, Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural Computation*, **15**: 441–454.