Incremental Learning of Complex Temporal Patterns

DeLiang Wang, *Member, IEEE,* and Budi Yuwono

Abstract—**A neural model for temporal pattern generation is used and analyzed for training with multiple complex sequences in a sequential manner. The network exhibits some degree of interference when new sequences are acquired. It is proven that the model is capable of incrementally learning a finite number of complex sequences. The model is then evaluated with a large set of highly correlated sequences. While the number of intact sequences increases linearly with the number of previously acquired sequences, the amount of retraining due to interference appears to be independent of the size of existing memory. The model is extended to include a chunking network which detects repeated subsequences between and within sequences. The chunking mechanism substantially reduces the amount of retraining in sequential training. Thus, the network investigated here constitutes an effective sequential memory. Various aspects of such a memory are discussed.**

I. INTRODUCTION

ONE of the fundamental aspects of natural intelligence is the ability to process temporal information [27]. Learning and recalling temporal patterns are closely associated with our ability to perceive and generate body movements, speech and language, music, etc. A considerable body of neural-network literature is devoted to studying temporal pattern generation (see [28] and [40] for reviews). These models generally treat a temporal pattern as a sequence of discrete patterns, called a temporal sequence. Most of the models are based on either multilayer perceptrons with backpropagation training or the Hopfield model of associative memory. The basic idea for the former class of models is to view a temporal sequence as a set of associations between consecutive components, and learn these associations as input–output pairs [22], [11], [31]. To deal with temporal dependencies beyond immediate predecessors, part of the input layer is used to keep a blended form of history, behaving as short-term memory (STM). Similarly, for temporal recall models based on the Hopfield associative memory, a temporal sequence is viewed as associations between consecutive components. These associations are stored in extended versions of the Hopfield model [39], [6], [21]. To deal with longer temporal dependencies, high-order networks have been proposed [19].

One of the main problems with these two classes of model lies in their difficulty in retrieving complex temporal sequences, where the same part may occur many times in the sequence. Though proposed remedies can alleviate the problem to a certain degree, the problem is not resolved. In multilayer perceptrons, a blended form of STM becomes increasingly ambiguous when temporal dependencies increase [3]. The use of high-order units in the Hopfield model entails a huge number of connections when long range temporal dependencies appear, or the model faces ambiguities [40].

More recently, Bradski *et al.* [5] proposed an STM model, and showed that both recency and primacy can be captured by such a model. In addition, their model creates new representations for repeated occurrences of the same symbol, thus capable of encoding complex sequences to a certain extent. Granger *et al.* [15] proposed a biologically motivated model for encoding temporal sequences. Their model uses a non-Hebbian competitive learning rule that eventually develops sequence detectors at the end of sequence presentation. Each detector encodes a sequence with the beginning component having the strongest weight, and the subsequent components having successively weaker weights. They claim that the network has an unusually high capacity. However, it is unclear how their network reads out the encoded sequences. Baram [1] presented a model for memorizing vector sequences using the Kanerva memory model [23]. The basic idea is similar to those models that are based on the Hopfield model. Baram's model uses second-order synapses to store the temporal associations between consecutive vectors in a sequence, and the model deals only with sequences that contain no repeating vectors. Rinkus [33] proposed a model of temporal associative memory, based on associations among random patterns. The associations are built using a scheme similar to the associative memory of Willshaw *et al.* [44]. However, it is not clear how to map a sequence component to a semirandom vector and remember the mapping later.

Based on the idea of using STM for resolving ambiguities, Wang and Arbib [41] proposed a model for learning to recognize and generate complex temporal sequences. With an STM model, a complex sequence is acquired by a learning rule that associates the activity distribution in STM with a context detector (for a rigorous definition see Section II-A). For sequence generation, each component of a sequence is associated with a context detector that learns to uniquely activate the component. After successful training, a beginning part of the sequence forms the context for activating the next component, and the newly activated component joins STM to form a context for activating yet a next component. This process continues until the entire sequence is generated. A later model [42] addressed the issues of time warping and chunking of subsequences. In particular, sequences can be recognized in a hierarchical manner and without being affected by presen-

Manuscript received May 25, 1995; revised January 9, 1996. This work was supported in part by NSF Grants IRI-9211419, IRI-9423312, and equipment Grant CDA-9413962, and the ONR Grant N00014-93-1-0335.

D. L. Wang is with the Laboratory for AI Research, Department of Computer and Information Science and Center for Cognitive Science, The Ohio State University, Columbus, OH 43210-1277 USA.

B. Yuwono is with the Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210-1277 USA.

Publisher Item Identifier S 1045-9227(96)07448-6.

tation speed. Hierarchical recognition enables the system to recognize sequences whose temporal dependencies are much longer than the STM capacity. In sequence generation, the system can maintain relative timing among the components while changing overall rate.

Recently we proposed an *anticipation* model for temporal pattern generation [43]. Similar to Wang and Arbib [41], [42], an STM model is used for maintaining a context which is stored by a context detector. In learning a temporal sequence, the model actively anticipates the next component. When the anticipation is correct, the model does nothing and continues to learn the rest of the sequence. When the anticipation is incorrect, namely a mismatch occurs, the model automatically expands the context for the component. A one-shot normalized Hebbian learning rule is used to learn contexts that exhibits the mechanism of temporal masking, where a sequence is preferred to its subsequences in winner-take-all competition. We proved that the anticipation model can learn to generate an arbitrary temporal sequence by self-organization, thus avoiding supervised teaching signals as required in Wang and Arbib.

In this paper, we focus on learning multiple temporal sequences. In particular, we study sequential training of multiple sequences, meaning that new sequences are learned after old sequences have been acquired. One way of learning multiple sequences is to use simultaneous training, where many sequences are learned at once. A model that can learn one sequence can be generally extended to learn multiple sequences with simultaneous training. A straightforward way is to concatenate these sequences into a single long sequence during training. Given that each sequence has a unique identifier, a model can learn all of the sequences if it can learn the concatenated sequence. However, sequential learning of multiple sequences is an entirely different matter. It is a more desirable form of training because it allows the model to acquire new knowledge on the basis of an existing memory—a form of *incremental learning*. Incremental learning not only conforms well with human learning experience, but also is important for many applications that do not have all the training data available at the beginning and where learning is a long-term ongoing process.

It turns out that incremental learning is particularly challenging to obtain. In multilayer perceptrons, it has been recognized that the network exhibits so-called catastrophic interference, whereby later training disrupts the traces of previous training. It was pointed out by Grossberg [18], and systematically revealed by McCloskey and Cohen [29] and Ratcliff [32]. Many subsequent studies attempt to address the problem. Most of the proposals amount to reducing overlapping in hidden layer representations by some techniques of orthogonalization, which were used long ago for reducing crosstalks in associative memories (see [25], [13], [14], [26], [38], and [36]). Most of these proposals are verified only by small-scale simulations, which, together with the lack of rigorous analysis, make it difficult to judge to what extent these proposed methods work. It remains to be seen whether a general remedy can emerge for multilayer perceptions at not too great a cost. Associative memories appear to be able to incorporate more patterns easily so long as the overall number of patterns does not exceed the memory capacity. However, the Hopfield model has an exceedingly low capacity when storing correlated patterns that have overlapping structures [20]. The Hopfield model has been extended to deal with correlated patterns [24], and Diederich and Opper [10] proposed a local learning rule to acquire the necessary weights iteratively. The local learning rule used by them is similar to the perceptron learning rule. Thus, it appears that such a scheme for dealing with correlated patterns should suffer from catastrophic interference.

The major cause of catastrophic interference is the *distributedness* of representations. The learning of new patterns needs to use those weights that participate in representing previously learned patterns. We can say that there is a trade-off between distributedness and interference. Models that use nonoverlapping representations, or local representations, do not exhibit the problem. For example, the adaptive resonance theory (ART) [7] does not have the problem because each pattern corresponds to the weight vector of a unique unit and no overlapping is allowed between any two weight vectors.

It is clear that during sequential learning, humans show some degree of interference. Retroactive interference has been well documented in psychology [9], which occurs when learning a later event interferes with the recall of earlier information. In general, the similarity between the current event and memory items is responsible for retroactive interference [2], [8], [4]. Retroactive interference exists in animals as well [34]. The existence of retroactive interference suggests that events are not independently stored in the brain, and related events are somehow associated in the memory. Also, though the recall performance with the interfered items decreases, it still is better than the chance level, and it is easier to regain these items than to learn them the first time. These considerations suggest that a memory model that stores every item independently does not provide an adequate basis for modeling human/animal memory. From the computational perspective, the models that contain a certain degree of sharing in storing different events have a better storage efficiency than those that do not. In sum, a desired memory model should exhibit a degree of retroactive interference when learning similar events, but not catastrophic interference.

In a preliminary simulation, the anticipation model seems to exhibit some retroactive interference when learning two sequences sequentially that have overlapping subsequences [43]. In this paper, the model will be analyzed in terms of its general performance on sequential training tasks. We will show that the anticipation model is capable of incremental learning with retroactive interference but without catastrophic interference. Extensive simulations reveal that the amount of retraining is relatively independent of the number of sequences stored in the model. Furthermore, a mechanism of chunking is proposed that creates chunks for recurring subsequences. This chunking mechanism significantly improves training and retraining performances.

The remaining part of the paper is organized as follows. In Section II, we provide necessary terminology and a brief description of the anticipation model. Section III gives a

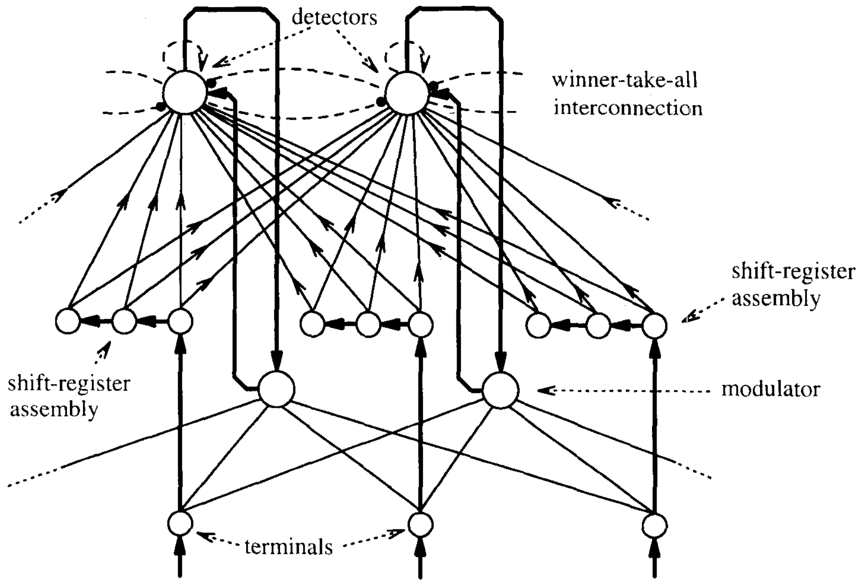


Fig. 1. Diagram of the anticipation model (from [43]). Thin solid lines indicate modifiable connections, and thick or dashed lines indicate fixed connections. The connections between terminals and modulators are bidirectional.

rigorous result of the model in sequential training, namely the model exhibits the ability of incremental learning. In Section IV, we provide the results of numerical simulations for learning many sequences incrementally, which suggest that incremental learning in the anticipation model is capacity independent. Section V describes the chunking mechanism and shows how chunking affects the learning performance of the same simulations as described in Section IV. Finally, Section VI provides some general discussions about the anticipation model.

II. Anticipation Model of Temporal Learning

A. Terminology

We follow the terminology introduced by Wang and Arbib [41]. Sequences are defined over a symbol set Γ, which contains all possible symbols, or spatial (static) patterns. Sequence S of length N over Γ is defined as $p_1 - p_2 - \cdots - p_N$, where $p_i (1 \leq i \leq N) \in \Gamma$ is a component of S. The sequence $p_j - p_{j+1} - \cdots - p_k$ where $1 \leq j \leq k \leq N$, is a *subsequence* of S, and the sequence $p_j - p_{j+1} - \cdots - p_N$ where $1 \leq j \leq N$, is a *right subsequence* of S. In general, in order to associate a component by its predecessors in a sequence, a prior subsequence is needed. For example, to produce the first "P" in the sequence M-I-S-S-I-S-S-I-P-P-I requires the prior subsequence S-I-S-S-I. This is because I-S-S-I is a recurring subsequence. Thus, the *context* of p_i is defined as the shortest prior subsequence of p_i which uniquely determines p_i in S, and the *degree* of p_i is the length of its context. The degree of S is defined as the maximum degree of all the components of S. According to these definitions, a *simple sequence*, where each component is unique, is a degree one sequence; a *complex sequence*, which contains recurring subsequences, is a sequence whose degree is greater than one.

B. Basic Network Description

We now describe the basic components of the anticipation model [43]. Fig. 1 shows the architecture of the network. The entire architecture consists of a layer of n input terminals, each associated with a shift-register (SR) assembly, a layer of m context detectors, each associated with a modulator. Each SR assembly contains r units, arranged so that an input signal sensed by an input terminal shifts to the next unit every time step. SR assemblies are introduced to serve as a STM for input signals. Each detector is projected by all SR units, and there are lateral connections, including self-excitation, within the detector layer to form winner-take-all dynamics. These connections and competitive dynamics amount to that the detector that receives the greatest ascending input from SR units will be the sole winner of the entire detector layer. In addition to the ascending and lateral connections, each detector also connects mutually with its corresponding modulator, which in turn connects directly with input terminals.

More specifically, let $I_j(t)$ represent the activity of terminal j at time t, and $V_{jk}(t)$ the activity of the kth SR unit of the jth assembly at time t. $I_j(t)$ is binary, indicating the presence/absence of a particular input symbol that the terminal represents. The activity of detector i is denoted by $E_i(t)$. The detector has another parameter d_i to indicate the degree of the sequence that the detector learns to recognize. Also, for detector $i, O_i(t)$ is used to indicate the result of competition in the detector layer, so that $O_i(t) = 1$ if detector i is the winner at time t and $O_i(t) = 0$ otherwise. The weight of the connection from the kth unit of assembly j to detector i is denoted by $W_{i,jk}$. Let $M_i(t)$ denote the activity of modulator i at time t, and R_{ij} the weight of the connection from terminal j to modulator i. These notations are consistent with the definitions introduced earlier [43]. The dynamics of these quantities has been defined in [43], and we provide in the Appendix relevant equations for completeness.

The learning process proceeds as follows. An input sequence S is presented to the network one component at a time. In each time step, only one detector unit can be activated because all of the detector units form a winner-take-all network [17]. The winning detector, denoted as z, performs one-shot learning to recognize a sequence of most active components held in STM by adapting its connection weights to match the activity levels of the corresponding SR units—much like template matching. The degree parameter d_z determines the length of the sequence that z can learn to recognize. The output of the detector, O_z, activates its corresponding modulator in the next step (assuming some delay) while the next component of S is input to a terminal. The simultaneous activation between the modulator and the terminal is used to form a unique connection between the two through one-shot learning. This is how the model associates a prior subsequence (context) with a sequence component. The modulator which receives top-down input from z and bottom-up input from terminals serves to detect the match between the anticipated next component made by the detector and the actual next component. If they match, no change is made to the model; if they do not, this mismatch signal is fed back to z to increment d_z so as to expand the context to be detected by z. At the same time, a new connection is formed between the modulator and the terminal. Training is completed when there is no mismatch during the presentation of the entire sequence.

We now summarize the major results of our previous study that are relevant to the present investigation.

1) It is proven that a normalized Hebbian rule (see the Appendix) leads to a property called *temporal masking*: a parameter in the learning rule can be chosen to guarantee that the detector of sequence S is preferred to the detectors of the right subsequences of S.
2) It is proven that the model with m detectors, and r SR units for each of n SR assemblies can learn to produce an arbitrary sequence S of length $\leq m$ and degree $\leq r$, where S is composed of symbols from Γ with $|\Gamma| \leq n$.

Once training is completed, the network can be used to produce the sequence it has been trained on. During sequence generation, the learned connections from input terminals to modulators are used in the reverse order for producing input components. Sequence generation is triggered by the presentation of the first component (or an identifier). This presentation will be able to trigger an appropriate detector which then, through its modulator, leads to the activation of the second component. In turn, the newly activated terminal adds to existing STM, which then forms an appropriate context to generate yet another component in the sequence. This process continues until the entire sequence is produced.

Aside from the theoretical result that the model can learn a sequence of arbitrary complexity by self-organization, learning is efficient—it generally takes just a few training sweeps to acquire a sequence. This is because the anticipation model employs the strategy of least commitment. The model assumes that the sequence to be learned is a simple one, and expands the contexts of sequence components only when necessary. Another feature of the model is that, depending on the nature of the sequence, the system can yield significant sharing in the use of context detectors. The same detector may be used for anticipating the same symbol that occurs many times in a sequence. As a result, the system needs fewer detectors to learn complex sequences than the model of Wang and Arbib [41], [42].

III. Incremental Learning

The previous study on the anticipation model is mainly concerned with the learning and storage of a single sequence [43]. A preliminary simulation presented in [43] indicates that the model has the ability of learning two sequences sequentially. We now analyze the network ability to learn multiple sequences sequentially. In particular, we are interested in what type of interference is exhibited by the model during sequential training.

As described in the previous section, a sequence is presented to the network one component at a time during training. Notice that each training sweep is associated with a single sequence. When dealing with multiple sequences, each sequence is viewed as unique, because learning a sequence that has been acquired corresponds to recalling the sequence. We assume that the first component of a sequence represents the unique identifier of the sequence. In order to make the following analysis possible, we give a precise definition of a *sequential learning procedure* as the following. The training process proceeds in rounds. In the first round, the first sequence is presented to the network in repeated sweeps. Once the network has learned the first sequence, the second round starts with the presentation of the second sequence. Once the second sequence is acquired by the network, the network is checked to see if it can generate the first sequence correctly when presented with the identifier of the sequence. If the network can produce the first sequence, then the second round ends. Otherwise, the first sequence is brought back for retraining; in this case, the first sequence is said to be *interfered* by the acquisition of the second sequence. If the first sequence needs to be retrained, the second sequence needs to be checked again after the retraining of the first sequence is completed, which may lead to the interference of the second sequence. The second round ends when both sequences can be produced by the network. In the third round, the third sequence is presented to the network repeatedly until the sequence has been learned. The network is then checked if it can generate the first two sequences; if yes, the third round is completed; if not, retraining is conducted. In the latter case, retraining is always conducted on the sequences that are interfered. The system sequentially checks and retrains each sequence until every one of the three sequences can be generated by the network—that ends the third round. Later sequences are sequentially trained in a similar manner. It is possible that a sequence that is not interfered as a result of acquiring the latest sequence gets interfered as a result of the retraining of some other interfered sequences. Because of this, retraining is conducted in a systematic fashion as the following. All of the previous sequences plus the current sequence are checked sequentially and retrained if interfered. This entire checking/retraining process is conducted repeatedly until no more interference occurs for every sequence learned so far. Each such process is called a *retraining cycle*. Thus a round in general consists of repeated retraining cycles.

It is clear that if a system exhibits catastrophic interference, it cannot successfully complete a sequential learning procedure with multiple sequences. The system instead will show endless *oscillations* between learning and relearning different sequences. In the case of two sequences, for example, the system can only acquire one sequence—the latest one used in the sequential training procedure. Thus, the system will be stuck in the second round. We are now ready to prove the main conclusion of this paper.

Theorem 1: Given sufficient numbers of detectors and SR units for each shift-register assembly, the anticipation model can learn to produce a finite number of sequences sequentially.

Proof: We prove the theorem by mathematical induction on the number of sequences to be trained. For only one sequence, the theorem is equivalent to Theorem 1 in Wang and Yuwono [[43], see 2) above]. Thus, the conclusion is true.

Let us assume that for k sequences, the theorem is true. That is, the model can learn to produce k sequences sequentially. Now, we examine the case with $k+1$ sequences.

According to the induction hypothesis, the first k sequences can be trained sequentially. Thus, we can assume that, when the model is trained on the $(k+1)$th sequence, denoted as S_{k+1}, the first k sequences have all been acquired by the model. Let $|S_{k+1}| = L$. During the training with S_{k+1}, each component of the sequence is presented to the model sequentially. When a component p_i is presented, one of the following three cases occurs. The first case is that right before p_i is presented, a committed detector is activated and this detector anticipates p_i. In this case, the system makes correct anticipation and nothing needs to be done. The second case is that right before p_i is presented, a committed detector is activated, but the detector anticipates a component different from p_i. In this case, a mismatch occurs and the detector will increase its degree by one and makes a link to p_i. The last case is that right before p_i is presented, no committed detector is activated. In this case, the system automatically selects an uncommitted detector, and makes a link from the detector to p_i. For the sequence S_{k+1} with length L, the system needs to use at most $L-1$ detectors. In addition, each detector needs to have at most a degree of $L-1$ because the first component of S_{k+1} is a unique component. Thus, learning S_{k+1} will succeed after a finite number of training sweeps if the model has enough detectors and SR units for each SR assembly.

Since learning S_{k+1} may interfere with the memory of some of the first k sequences, some retraining may be needed to retain the first k sequences. According to the induction hypothesis, retraining will succeed after a finite number of training sweeps.[1] The same reasoning reveals that the retraining process can interfere with the memory of S_{k+1}. Therefore, S_{k+1} may need to be retrained. The following observation ensures that the retraining of S_{k+1} and that of the first k sequences will not enter an oscillating (infinite) loop. During the retraining of any interfered sequence, one of the three cases outlined in the previous paragraph occurs. As a result, some committed detectors may increase their degrees and some new detectors may be committed. Let the length of the longest sequence of the $k+1$ sequences be M. Since each sequence starts with a unique component, the degree of any detector is at most $M-1$. In addition, the model needs to use at most $\Sigma_{i=1}^{k+1}|S_i| - (k+1)$ detectors. Thus, the retraining process will end after a finite number of sweeps. Q.E.D.

Remark 1: In the theorem, the number of units in each SR assembly, or the STM capacity, must be sufficient to handle long temporal dependencies in the context of multiple sequences. It should be clear that the complexity of a sequence may increase when it is trained with other sequences. For example, X-A-B-C and Y-A-B-D are both simple sequences when taken independently. But when the system needs to memorize both sequences, A-B becomes a repeating sequence, and as a result none of them is simple any longer. We define the *degree of a set of sequences* as the length of the shortest prior sequence that uniquely determines every component of every sequence in the set. Because each sequence has its identifier as the first component, the definition of the set degree does not depend on how this set of sequences is ordered. Moreover, the degree of a set of sequences must be smaller than the length of the longest sequence in the set. With this definition, it is sufficient to satisfy the condition of Theorem 1 regarding SR units if the number of SR units in each assembly is greater than or equal to the set degree.

Remark 2: The sequential learning procedure is not necessary for the validity of Theorem 1. A more relaxed procedure of sequential training is to postpone retraining until interfered sequences need to be recalled in a specific application task. This procedure is more consistent with human learning process. One often does not notice memory interference until being tested in a psychological experiment or daily life. This learning procedure blurs the difference between learning a new sequence for the first time and relearning an interfered sequence. The proof of Theorem 1 essentially implies that more and more sequences will be acquired by the system as the learning experience of the model expands. This is an important point. As a result, the model can be viewed as an open system of learning. No rigid procedure for sequential training is needed for the system to increase its long-term memory capacity. The model automatically increases its capacity by just focusing on learning the current sequence.

Remark 3: The number of detectors needed to satisfy Theorem 1 can be significantly smaller than the upper limit of $\Sigma_{i=1}^{k}|S_i| - k$ for k sequences. This is because detectors can be shared within the same sequence as well as across different sequences. This point will be further discussed in Section IV.

IV. Sequential Training Examples

Theorem 1 guarantees that the anticipation model does not suffer from catastrophic interference. Interference exists nonetheless in sequential training, because committed detectors may be seized by later training or retraining to make

[1] We recognize possible differences between sequentially learning k sequences from the scratch or a blank model and retraining k sequences. But these differences do not matter because of the following observation. We can strengthen the conclusion for the case of one sequence to include the situation that contains committed detectors to start with. The strengthened conclusion can be shown to be true following the same argument that S_{k+1} can be learned after the first k sequences are acquired.

TABLE I
SEQUENCE BASE FOR PHASE I TRAINING

no.	Sequence	no.	Sequence
1	Learning and Memory II	7	Time Series Prediction
2	Intelligent Control II	8	Neural Systems Hardware
3	Pattern Recognition II	9	Image Processing
4	Hybrid Systems III	10	Applications of Neural Networks to Power Systems
5	Probabilistic Neural Networks and Radial Basis Functions	11	Supervised Learning
6	Artificially Intelligent Neural Networks II		

TABLE II
SEQUENCE BASE FOR PHASE II TRAINING

no.	Sequence	no.	Sequence
1	Social and Philosophical Implications of Computational Intelligence	50	Image Recognition
2	Neurocontrol Research: Real-World Perspectives	51	Medical Applications
3	Fuzzy Neural Systems	52	Parallel Architectures
4	Advanced Analog Neural Networks and Applications	53	Associative Memory I
5	Neural Networks for Control	54	Pattern Recognition IV
6	Neural Networks Implementations	55	Supervised Learning III
7	Hybrid Systems I.D.	56	Learning and Memory IV
8	Artificial Life	57	Intelligent Control IV
9	Learning and Recognition for Intelligent Control	58	Economic/Finance/Business Applications
10	Artificially Intelligent Neural Networks	59	Machine Vision I
11	Hybrid Systems II	60	Machine Vision
12	Supervised Learning X	61	Architecture I
13	Intelligent Neural Controllers: Algorithms and Applications	62	Supervised Learning V
14	Who Makes the Rules?	63	Speech I
15	Pulsed Neural Networks	64	Robotics
16	Fuzzy Neural Systems II	65	Associative Memory II
17	Neural Networks Applications to Estimation and Identification	66	Medical Applications II
18	Adaptive Resonance Theory Neural Networks	67	Modular/Digital Implementations
19	Analog Neural Chips and Machines	68	Pattern Recognition VI
20	Learning and Memory I	69	Robotics II
21	Pattern Recognition I	70	Unsupervised Learning I
22	Supervised Learning I	71	Optimization I
23	Intelligent Control I	72	Applications in Image Recognition
24	Neurobiology	73	Architecture III
25	Cognitive Science	74	Optimization II
26	Image Processing III	75	Supervised Learning VII
27	Neural Network Implementation II	76	Associative Memory IV
28	Applications of Neural Networks to Power Systems	77	Robotics III
29	Neural System Hardware I	78	Speech III
30	Time Series Prediction and Analysis	79	Unsupervised Learning II
31	Probabilistic Neural Networks and Radial Basis Function Networks	80	Neurodynamics I
32	Pattern Recognition II	81	Applications I
33	Supervised Learning II	82	Applied Industrial Manufacturing
34	Image Processing I	83	Applications II
35	Learning and Memory II	84	Architecture IV
36	Hybrid Systems III	85	Optimization III
37	Artificially Intelligent Networks II	86	Applications in Image Recognition II
38	Fast Learning for Neural Networks	87	Unsupervised Learning III
39	Industry Application of Neural Networks	88	Supervised Learning VIII
40	Neural Systems Hardware II	89	Neurodynamics II
41	Image Processing II	90	Computational Intelligence
42	Nonlinear PCA Neural Networks	91	Optimization Using Hopfield Networks
43	Intelligent Control III	92	Supervised Learning IX
44	Pattern Recognition III	93	Applications to Communications
45	Supervised Learning III	94	Applications III
46	Applications in Power	95	Unsupervised Learning IV
47	Time Series Prediction and Analysis II	96	Optimization IV
48	Learning and Memory III	97	Applications
49	Intelligent Robotics		

different anticipation. For example, assume that the system is sequentially trained with two simple sequences: S_a: C-A-T and Sb: E-A-R. Once S_a has been learned, the training with S_b will lead to the following situation. The previously established link from A to T will be replaced by a link from E-A to R. As a result, S_a is interfered and cannot be generated after S_b is acquired. The question now is what kind of interference is exhibited by the model, and how severely does it affect learning performance? The extent of interference depends on the amount of overlap between the sequence to be learned and the sequences already stored in the memory. Clearly, if a new sequence has no component in common with the stored sequences, the sequence can be trained as if nothing had been learned by the model. In this sense, interference is caused by the similarity between the sequence and the memory. This is consistent with psychological studies on retroactive interference.

Knowing that the amount of interference, and thus retraining, depends on the overlap of the sequences to be learned, we arbitrarily select a domain that contains a high degree of overlap among the sequences. The database of the sequences used consists of the titles of all sessions that were held during the 1994 IEEE International Conference on Neural Networks (ICNN-94). The model is evaluated in two phases, both of which use the sequential training procedure. In Phase I, we test the model using 11 sequences; in Phase II, we test the model using the whole set of 97 sequences. Phase I with fewer sequences is included for illustrating the process of the sequential training procedure. The set of sequences used in Phase I is composed of the titles of a group of 11 parallel sessions, arbitrarily selected from many such groups. The two test sets are listed in Tables I and II, respectively. These titles are listed without any change and in exactly the same order as they appear in the brochure of the final conference program, even retaining the obvious mistakes printed in the program. As is evident from the two tables, there are many overlapping subsequences within each set. Thus, we believe that these sequences provide a good testbed for evaluating sequential training and retroactive interference.

A. *Phase I Training*

For Phase I training, the network has 45 input terminals, 34 of which are for the symbol set (26 English letters plus "#," " " (space), ".," "&," "?," "-" (hyphen), ":," and "/"), and 11 for the identifiers of the 11 sequences. There are 20 SR units for each SR assembly. In addition, the network needs a minimum of 250 detectors and 250 modulators. Thus, the network has a total of 1445 units. The values of the system parameters (see the Appendix for their definitions): $\alpha = 0.2, \delta = 1/20$, and $C = 134$. To measure the extent of interference, we keep track of the number of retraining sweeps required to eliminate all interference for every round of sequential training. This number is a reliable indicator of how much retraining is needed to store all of the sequences that have been sequentially presented to the system. Also, we keep track of the number of intact (noninterfered) sequences right after the acquisition of the latest sequence.

Fig. 2 shows the number of intact sequences and the number of retraining sweeps for 11 training rounds. As is expected, the number of intact sequences grows, albeit unsteadily, as the number of stored sequences increases. On the other hand, the number of retraining sweeps varies greatly from round to round. To see why this variation occurs, let us compare the training episodes of sequence 10 which requires 38 retraining sweeps, the highest of all rounds, and sequence 11 which requires 15 retraining sweeps. Sequence 10, denoted as S_{10}, is "*Applications of Neural Networks to Power Systems*." This sequence contains long subsequences that occur in previously acquired sequences, such as "*Neural Networks*" which occurs both in S_5 and S_6, and "*Systems*" which occurs in S_4 and S_8. On the other hand, S_{11} contains only one such sequence, "*Learning*," and that occurs only once in S_1. This reinforces the earlier point that the overlap between the stored sequences causes interference. The same explanation can apply to S_6 in Table I, which requires 31 retraining sweeps (see Fig. 2). For long overlapping subsequences, many sweeps may be needed to resolve the interference caused by overlaps.

As an illustration of what happens during retraining, Fig. 3 shows the detailed process during round four after S_4 has been acquired. Each row represents the activity of an input terminal with respect to time. The symbol "#" at the bottom row represents the end-of-sequence marker. Retroactive interference can cause two problem scenarios. In the first scenario, the interfered sequence is switched to a part of a different sequence during generation. For example, the shaded boxes in retraining S_1 (see Fig. 3) represent terminal activities which do not belong to S_1, but to S_4 which is the latest sequence stored in the model. This effect resembles the recency factor, which describes a phenomenon in human learning where the recall of more recent items is facilitated. This is because a context detector that has been used for storing S_1 is also used in training S_4, and this detector can be triggered by a subsequence of S_1 even after S_4 is acquired successfully. In the second scenario, the interfered sequence is aborted before the entire sequence is generated. This scenario happens frequently in Fig. 3. When a detector that has been used before participates in storing another sequence, its degree increases, and therefore the interfered sequence may not be able to trigger the detector.

One of the most critical questions left unanswered in Phase I learning is *whether* the number of retraining sweeps increases with the size of the stored sequences, or the round of sequential training. The database used in Fig. 2 is too small, and variations in retraining sweeps are too large to permit a reliable conclusion. It is also interesting to see how the number of intact sequences changes with respect to the number of stored sequences. Phase II training is included to answer these questions.

B. *Phase II Training*

For Phase II training, the network has 131 input terminals—34 for the symbol set as in Phase I and 97 for the identifiers of 97 sequences. Each SR assembly contains 40 units. Also, the network needs at least 1088 detectors and 1088 modulators. Hence, the network has a total of 7567 units. The parameters of the network take the following values: $\alpha = 0.2, \delta = 1/40$, and $C = 535$. As in Fig. 2, Fig. 4 displays the number of intact sequences and the number of retraining sweeps with respect to training rounds.

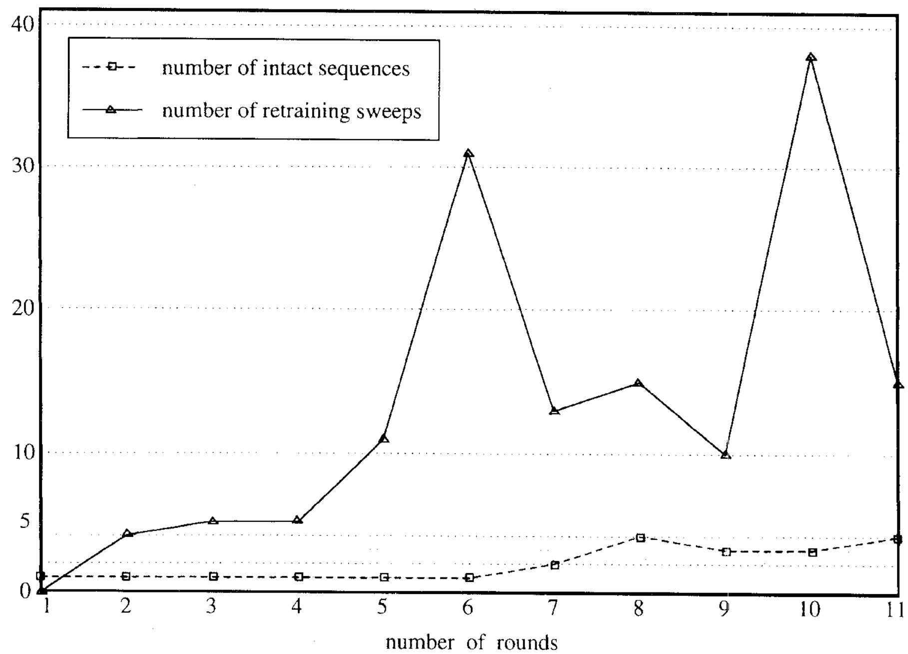


Fig. 2. The number of intact sequences and the number of retraining sweeps plotted against training rounds during Phase I learning.

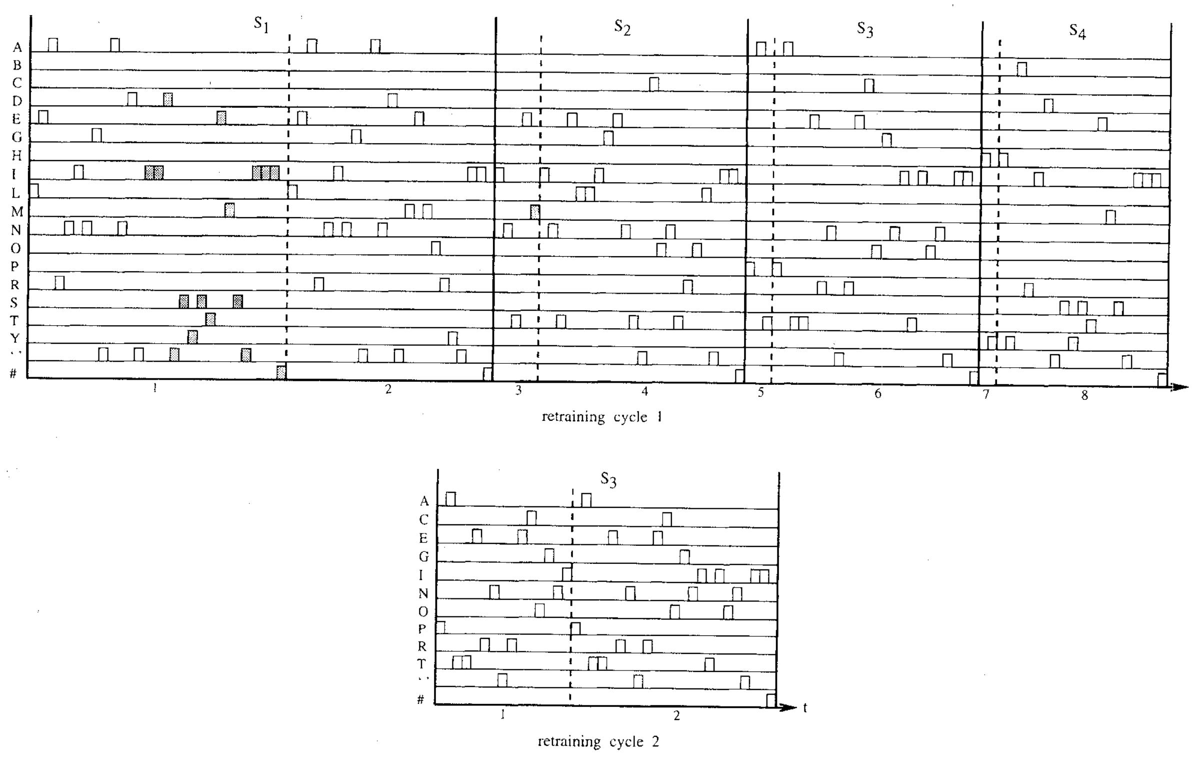


Fig. 3. Retraining process during round 4 of Phase I training. Each row represents the activity trace of an input terminal, as indicated by the corresponding input symbol. White boxes represent correct terminal activities in sequence generation, whereas shaded boxes represent terminal activities which do not belong to the desired sequence. Solid vertical lines separate training sweeps of different sequences, and dashed vertical lines separate training sweeps of the same sequence. Cycle numbers are indicated under each panel. Time runs from left to right.

Several conclusions can be drawn from this simulation. The first and the most important conclusion is that the number of retraining sweeps seems independent of the size of the previous memory. The overall curve for retraining sweeps remains flat, even though there is a large amount of variation across different training rounds. We view this result as particularly

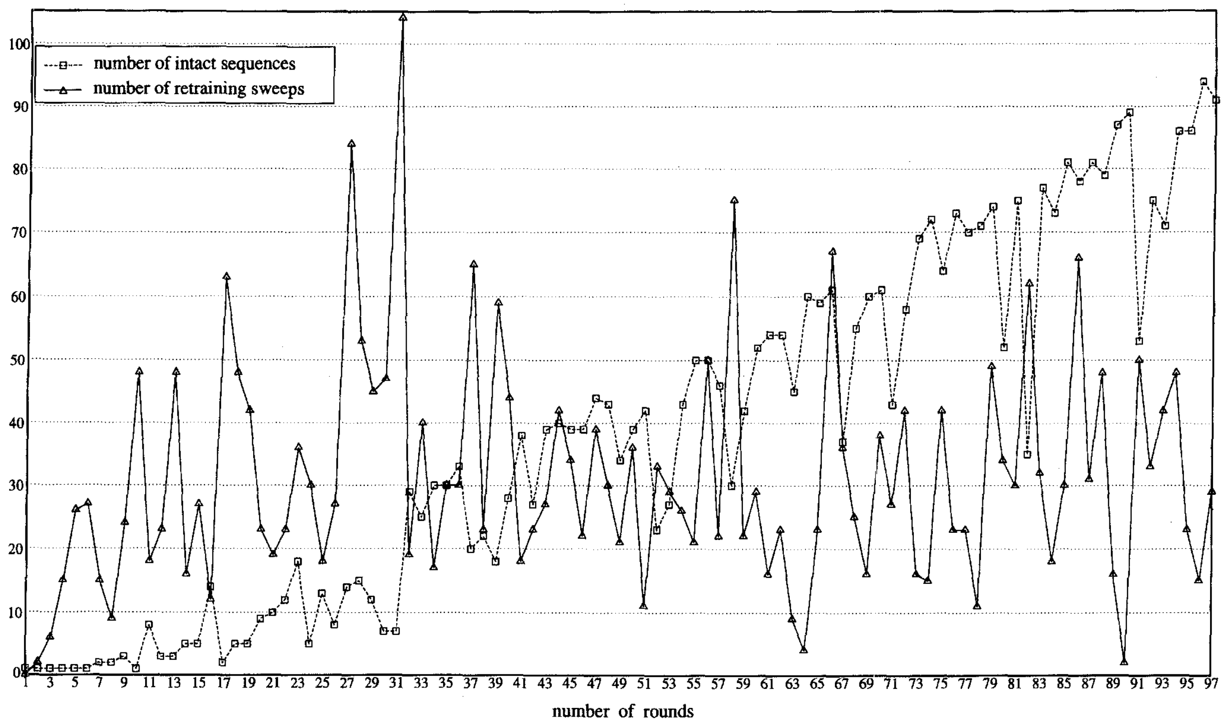


Fig. 4. The number of intact sequences and the number of retraining sweeps plotted against training rounds during Phase II learning.

significant because it suggests that in the anticipation model the amount of interference caused by learning a new sequence does not increase with the number of previously memorized sequences. This conclusion not only conforms intuitively with human performance of long-term learning, but also makes the model feasible to provide a reliable sequential memory that can be incremented or updated later on. In this case, new items can be incorporated into the memory readily without being limited by those items already in the memory. We refer to this property of sequential learning/memory as *capacity-independent incremental learning/memory*. The anticipation model exhibits this property because a learned sequence spreads its traces across the network, involving a set of distributed associations between subsequences and context detectors (see Fig. 1). On the other hand, each context detector stores its context locally. When a new sequence is learned, it employs a group of context detectors, some of which may have been committed, thus causing interference. But as the sequential memory becomes large, so is the number of context detectors. Out of these detectors, only a certain number of them can get interfered because of learning a new sequence. The number of interfered detectors tends to relate to the new sequence itself, not the size of the sequential memory. Contrasting capacity independent incremental learning, catastrophic interference would require simultaneous retraining of the entire memory when a new item is to be learned. The cost of retraining when catastrophic interference occurs can be prohibitive if the size of memory is not so small. Ruiz de Angulo and Torras [35] presented a study on sequential learning of multilayer perceptrons, and reported that their model can learn sequentially several most recent patterns. Although it is a better result than the original multilayer perceptrons, their model appears unable to support a sizable memory. The high variations in the number of retraining sweeps shown in Fig. 4 can be explained similarly as those shown in Fig. 2.

The second conclusion is that the number of intact sequences increases with the round of sequential training approximately linearly. This is to be expected given the result on the amount of retraining. Again, there are considerable variations from one round to another. Since interference is caused by the overlap between a new sequence and the stored sequences, another way of looking at this result is the following: As the memory expands, relatively fewer items in the memory will overlap with the new sequence.

Fig. 5 illustrates the detailed retraining process during round 96. Right after the model learned S_{96} of Table II, the only interfered sequence was S_{14}. After S_{14} was retrained, S_{96} was interfered and had to be retrained. This finished the first retraining cycle for round 96. During the second cycle, S_{71} was found to be interfered. After S_{71} was retrained, S_{96} was interfered again and had to be retrained. The retraining with S_{71} alternated with that of S_{96} for three more cycles. Notice the large overlap between S_{71}: "*Optimization* I" and S_{96}: "*Optimization* IV." In cycle 6, several more sequences were interfered. After retraining them sequentially, all of the first 96 sequences had been learned successfully.

To examine the amount of detector sharing, let us compare the detector use in the anticipation model with one in which no detector is shared by different components (see, for example, [42]). Without detector sharing, the number of detectors needed to acquire all of the 97 sequences in Table II is $\Sigma_{i=1}^{97} |S_i| - 97$, which equals 2520. As stated earlier, the

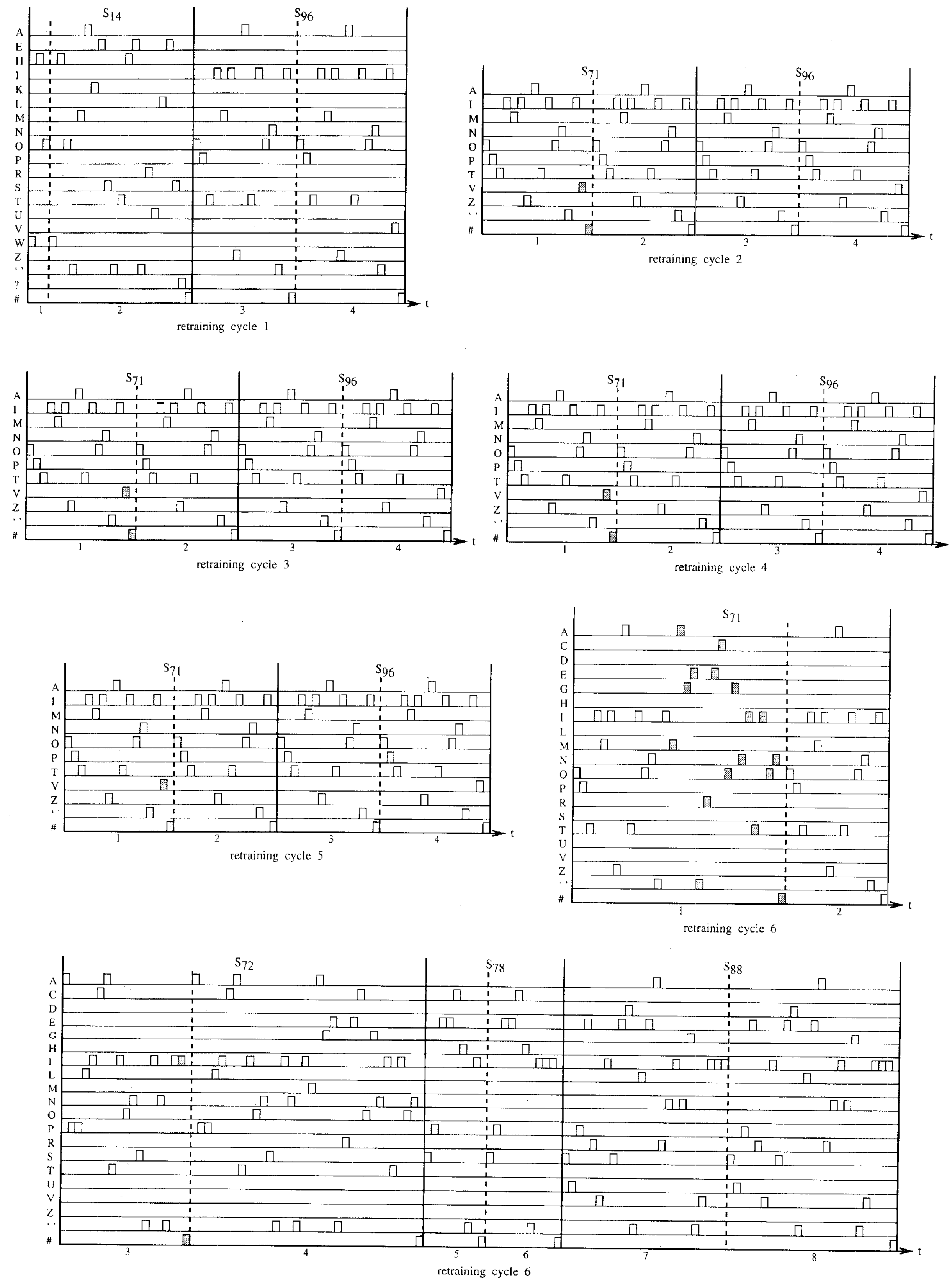


Fig. 5. Retraining process during round 96 of Phase II training. See the caption of Fig. 3 for notations of this figure.

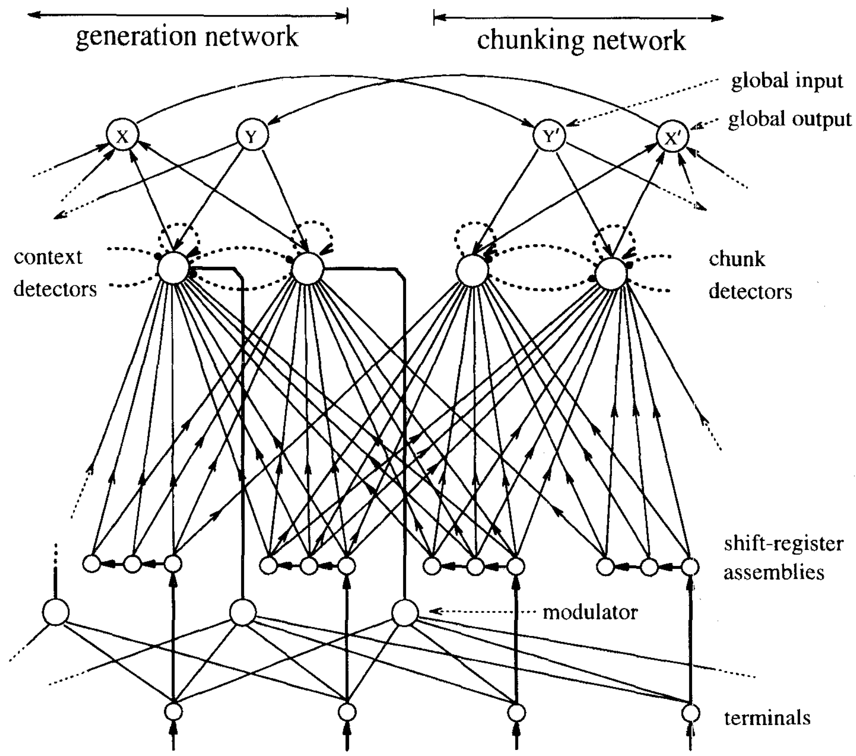


Fig. 6. Diagram of the extended anticipation model with a dual architecture. The entire architecture consists of two mutually connected networks: the generation network and the chunking network. The connections to and from global input and output units have fixed weights. See the caption of Fig. 1 for other notations of this figure.

anticipation model needed 1088 detectors to learn all of the sequences. Thus, the detector sharing in the anticipation model cuts required context detectors by nearly a factor of 2.5.

V. Context Learning by Chunking

Even though the amount of retraining does not depend on the number of stored sequences in the memory, retraining can be expensive. As shown in Fig. 4, it usually takes dozens of retraining sweeps to fully incorporate a new sequence. Our analysis of the model indicates that repeated sweeps are usually caused by the need of committing a new detector and gradually expanding the context of the detector to resolve ambiguities resulting from long recurring subsequences. For example, consider the situation that the system has stored the two sequences

$$S_c\text{:}\quad \text{“}JOE\ LIKES\ 0\text{”}$$
$$S_d\text{:}\quad \text{“}JAN\ LIKES\ 1\text{”}$$

and the network is learning the sequence

$$S_e\text{:}\quad \text{“}DEB\ LIKES\ 2.\text{”}$$

There is an overlapping subsequence between S_c and S_d: "*LIKES*" (ignore the spaces here). A common situation before S_e is learned is that there are two detectors, say u_1 and u_2, tuned to the contexts of "*E LIKES*" and "*N LIKES*," respectively. While S_e is being trained, neither u_1 nor u_2 can be activated and a new detector, say u_3, will be committed to anticipate "2" (u_3 needs to recognize only "S" since "S" cannot activate either u_1 or u_2). Suppose now the network is trained with yet another sequence

$$S_f\text{:}\quad \text{“}DIK\ LIKES\ 3,\text{”}$$

S_e and S_f will take turns to capture u_3 and gradually increase its degree until u_3 can detect either "*B LIKES*" or "*K LIKES*." Eventually, another detector, say u_4, will be committed for the remaining sequence. This gradual process of degree increment is a major factor causing numerous retraining sweeps.

The above observation has led to the following extended model for reducing the amount of retraining. The basic idea is to incorporate a chunking mechanism so that newly committed detectors may expand their contexts from chunks formed previously, instead of from the scratch.

A. Network Architecture

The extended model consists of a dual architecture, as shown in Fig. 6. The dual architecture contains a *generation network* (on the left of Fig. 6), much of which is the same as the original architecture (see Fig. 1), and another similar network, called the *chunking network* (on the right of Fig. 6). The two networks are mutually connected at the top. The chucking network does not produce anticipation, and thus it does not need a layer of modulators. Because of this, the detectors in this network do not increment their degrees by mismatches. Besides, the chunking network mirrors every process occurring in the generation network.

In any time step during training, there is a pair of winning detectors in the dual architecture, each corresponding to one network. The dynamics is designed so that the winning detector of the chunking network has a degree that is always one less than the degree of the winning detector of the generation network. In addition, a newly committed detector of the generation network may take a degree which is one plus the degree of the activated chunk detector. We refer to the detectors of the chunking network as *chunk detectors*. The interaction between the two networks takes place via the two-way connections between the two networks (Fig. 6). The introduction of the chunk detectors can speed up learning when a subsequence (a chunk) occurs multiple times in the input flow. More specifically, assume that a context detector u_i of degree d has learned to recognize a context, and a corresponding chunk detector $u_{i'}$ of degree $d-1$ has learned a chunk which is a right subsequence of the context learned by u_i. If the chunk occurs at least twice, then there will be a time when $u_{i'}$ is activated but u_i is not. Through regular learning an uncommitted context detector, say u_j, is activated (thus committed). Instead of starting from degree one, u_j starts its degree at the value of d, leading to a significant reduction of training/retraining sweeps.

B. Formal Description

The degree parameters of chunk detectors are modified on the basis of the input from the generation network. Since the two networks are very similar, we use a prime symbol to indicate the corresponding symbols of the chunking network. The cross-network interaction is carried out by two pairs of global units. One pair is X and Y', where X denotes the output unit of the generation network, and Y' the input unit of the chunking network. The other pair involves X' and Y, where X' denotes the output unit of the chunking network and Y the input unit of the generation network. We also use these symbols to denote their activities.

In addition to the description for the generation network (Section II and the Appendix), we need to specify the chunking network, the two pairs of global units and the update of the degree parameters of the context detectors.

The size of the chunking network is set to be the same as for the generation network. The activity, the learning rule, and the threshold adjustment for a chunk detector i' are defined just as for a context detector (see the Appendix), including using the same parameters such as C and α [see (A4)]. Likewise, the degree parameter of i', $d_{i'}$, is initialized to zero. Once winner-take-all dynamics in both networks reaches the equilibrium, the winning detector of each network sends its output to its respective global output unit. The activity $X(t)$

$$X(t) = \sum_i O_i(t)(d_i(t) - 1) \tag{1}$$

and $X'(t)$ of the chunking network

$$X'(t) = \sum_{i'} O_{i'}(t)(d_{i'}(t) + 1). \tag{2}$$

Each of the above units sends its output to the global input unit of the other network. For the generation network

$$Y(t) = X'(t) \tag{3}$$

and for the chunking network

$$Y'(t) = g(X(t), \theta') \tag{4}$$

$$g(x, y) = \begin{cases} x & \text{if } x \geq y \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The parameter θ' is an integer threshold which defines the minimum size of a chunk. No chunking takes place unless the size of a potential chunk is greater than or equal to θ'. As θ' increases, fewer chunk detectors are required and less chunking goes on. This threshold is introduced to control the balance among these elements.

The degree of detector i' is updated according to

$$d_{i'}(t+1) = d_{i'}(t) + O_{i'}(t) g(Y'(t) - d_{i'}(t), 0). \tag{6}$$

This, together with (4) and (1), implies that $d_{i'}$ gets increased at $t+1$ if i' is a winner at t and the winning context detector at t has a degree which is greater than $d_{i'}(t) + 1$. Otherwise, $d_{i'}$ remains the same. Notice that the update takes place in the next time step.

The update rule for d_i is a little more complicated, since it is affected both by a mismatch within the generation network during anticipation and by the projection from the chunking network. Combining these elements together, we define

$$d_i(t+1) = \begin{cases} O_i(t) Y(t) & \text{if } Y(t) > d_i(t) \\ d_i(t) + O_i(t) & \text{if } Y(t) \leq d_i(t) \text{ and } M_i(t) = 0 \\ d_i(t) & \text{otherwise.} \end{cases} \tag{7}$$

See the Appendix for the definition of $M_i(t)$ and (A8) for comparisons. This update rule assures that if there is a nonzero degree increase as a result of input from the chunking network, d_i will be so increased; at the same time, the mismatch within the generation network will not yield degree increment. Apart from that, d_i will be subject to the same internal modification as is defined for the generation network only. It should be mentioned that in the same time step the update to the degree parameter d_i precedes the weight update to the detector i (see [43]).

Before presenting simulation results with chunking in the next section, we explain using the example at the beginning of this section how the dual architecture helps speed up training. After training with S_c and S_d, there will be a context detector which is tuned to either "E *LIKES*" or "N *LIKES*." This, in turn, will lead to the formation of the chunk "*LIKES*;" that is, a chunk detector will be tuned to "*LIKES*" since the detector has a degree that is one less than that of the corresponding context detector. After the chunk is formed, S_e can be acquired easily—the detector which is trained to associate with "2" can obtain its appropriate degree in just one sweep, thanks to the formation of the chunk "*LIKES*." Similarly, S_f can be acquired quickly.

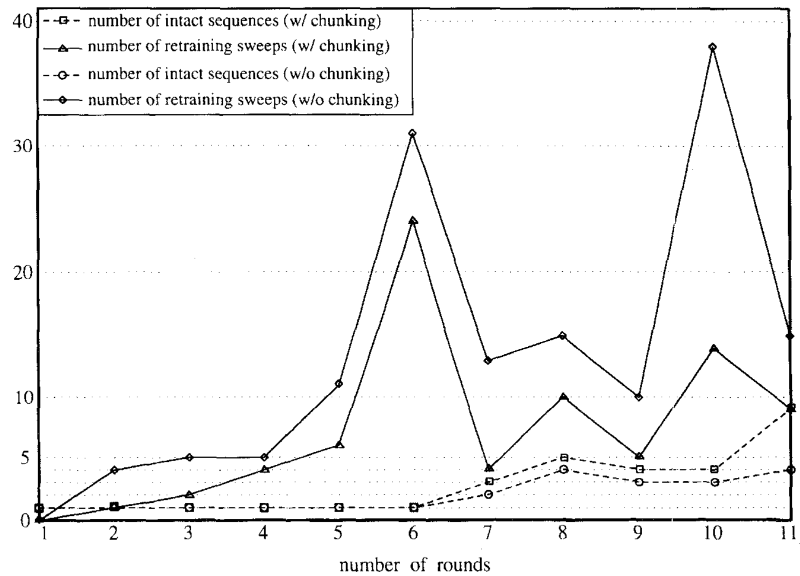


Fig. 7. The number of intact sequences and the number of retraining sweeps plotted against training rounds during Phase I learning with the dual architecture. For comparison purposes, the results of Fig. 2 are also included.

The speedup in sequential training does not come without a cost. Apparently, the addition of another network—the chunking network—adds both to the size of the overall network and to additional computing time. The latter, however, would not be a concern if the overall network is implemented on a fully parallel architecture.

C. *Performance*

To evaluate the effectiveness of the chunking mechanism, we conducted simulations using the same sequences and the same procedure as in Section IV. The model was again tested in two phases, where Phase I involves 11 sequences and Phase II involves 97 sequences.

For Phase I training, the parameters of the network were set to be the same as those used in the previous simulation. The only extra parameter introduced in defining the chunking network is θ', which was set to zero to allow the maximum amount of chunking. With $\theta' = 0$, the minimum chunk has a single component. The size of the overall network is a little different. To complete Phase I training, the dual architecture needed a minimum of 255 context detectors and 95 chunk detectors. The reason for a few more context detectors in the current simulation is the following. With chunking, context detectors may sometimes end up with degrees a little larger than absolutely required for disambiguation. The situation that detectors take larger-than-required degrees may lead to less context sharing among different sequence components across different sequences or within the same sequence, and the demand for more context detectors. The setup for other parts of the dual architecture is the same as in Section IV-A. The results are shown in Fig. 7 with the same format as in Fig. 2. To facilitate comparison, the results obtained by the dual architecture are plotted together in Fig. 7 with the previous results of Fig. 2. Compared to the previous results, the number of intact sequences with chunking is greater after round six. Meanwhile, the number of retraining sweeps is reduced considerably. The dual architecture never performed worse in any round.

For Phase II training, θ' was again set to zero. To complete the Phase II training, the dual architecture required 1234 context detectors as compared to 1088 required for the situation without chunking, and 436 chunk detectors. Other parts of the network, including parameter values, are the same as used in Section IV-B. Fig. 8 presents the results. A comparison between the results in Fig. 8 and in Fig. 4 shows that the former gives rise to a little more intact sequences. This indicates that, in the dual architecture, later training causes almost the same degree of interference. The dual architecture requires 54% more detectors than the original model. On the other hand, when the chunking is incorporated, the number of retraining sweeps on the whole reduces dramatically. The total number of retraining sweeps during entire Phase II training is 1104 when the chunking network is included. This is compared to 3029 without the chunking network. This represents a reduction of overall amount of sequential training almost by threefold.

For a comparison with Fig. 5 which illustrates the retraining process of round 96, Fig. 9 shows the detailed retraining process during round 96 using the dual architecture. Right after S_{96} was learned, three sequences were interfered: S_{59}, S_{60}, and S_{86}. After S_{59}: "*Machine Vision* I" was retrained, S_{60}: "*Machine Vision* II" could be correctly generated without further retraining. This interesting situation arises because the two sequences have a large overlap and it is the overlapping part that was interfered during training S_{96}. Thus, when the overlapping part was regained during retraining with S_{59}, both S_{59} and S_{60} could be retrieved correctly. The system took

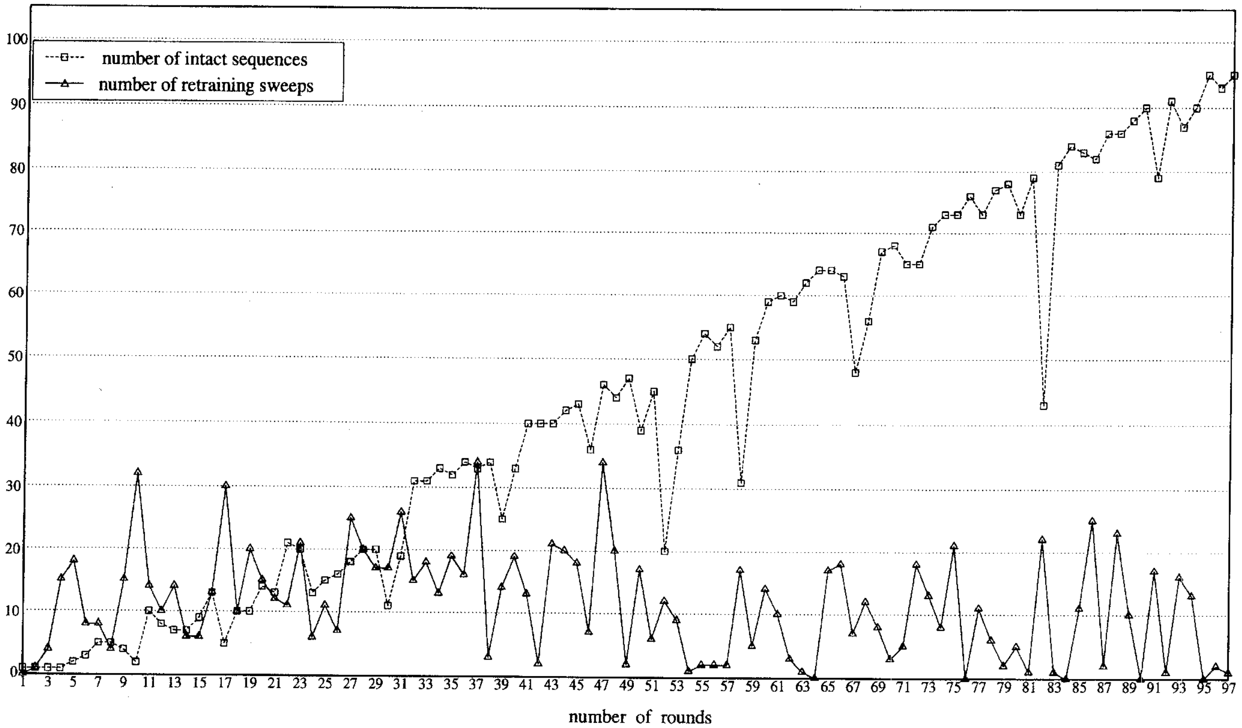


Fig. 8. The number of intact sequences and the number of retraining sweeps plotted against training rounds during Phase II training with the dual architecture.

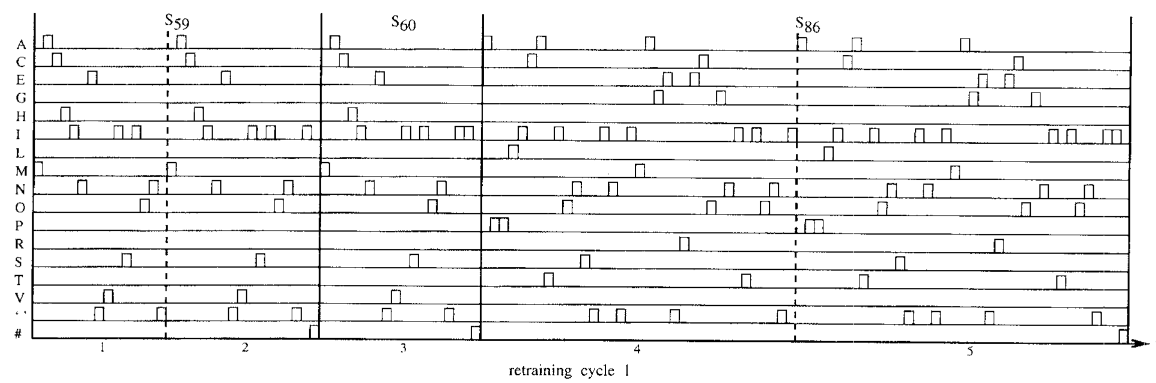


Fig. 9. Retraining process during round 96 of Phase II training with the dual architecture. The interference with S_{60} was eliminated as a result of retraining S_{59}. See the caption of Fig. 3 for notations of this figure.

another sweep to regain S_{86}. After the first retraining cycle, all of the first 96 sequences had been acquired.

VI. Discussion

The analysis clearly shows that the anticipation model does not suffer catastrophic interference. When multiple sequences are acquired by the model sequentially, some degree of interference occurs. But this kind of interference can be overcome by retraining the interfered sequences. Extensive computer simulations indicate that the amount of retraining does not increase as the number of sequences stored in the model increases. The anticipation model is characteristic of capacity independent incremental learning during sequential training. These results, plus the fact that interference is caused by the overlap between a new sequence and stored sequences, suggest that the behavior of the model in sequential learning resembles some elements of retroactive interference.

Once a sequence S is learned, it can be generated by its beginning component—the sequence identifier. Partial sequence generation can be elicited by a subsequence of S. If a sufficient subsequence is provided, the rest of S can be generated entirely. Partial generation can stop before the rest of the sequence is completed. For example, after the sequence X-A-B-C-D-E-A-B-C-D-F is learned, the presentation of A activates the subsequence B-C-D, but not the rest. The anticipation model exhibits partial generation because a sequence is stored as a chain of associations, each of which is triggered by a context, or a subsequence. This property of the model is consistent with our experience that we are able to pick up a familiar song, a melody, or an action sequence (like *Tai Chi*) from the middle. Partial generation gives the model a dimension of flexibility in generating traces of stored sequences.

As shown in Section V, the capability of chunking repeated subsequences within a sequence and between sequences substantially reduces the amount of retraining and improves the overall efficiency of learning in the present model. Without the chunking scheme, recurring subsequences must be learned separately as they occur in training process. The basic idea behind current chunking is to learn a recurring subsequence just once and store it as a chunk, so that the next time the subsequence occurs the model can simply use the chunk as a basic component. This is implemented by introducing a layer of chunk detectors which operates in a way similar to context detection. The dual architecture self-organizes to perform chunking. It is interesting to compare the present chunking method with the hierarchical model of temporal sequence recognition of Wang and Arbib [42]. In their model, a sequence in a layer becomes a basic component in the next layer, and using this idea of chunking they show that detectors in a higher layer can recognize sequences much longer than the basic capacity of STM. Unlike the anticipation model, their model does chunking in a supervised manner. It is still an open issue how hierarchical sequence recognition can be achieved by self-organization. On the other hand, the present model cannot learn a set of sequences if the set degree is greater than r, the STM capacity. It requires further investigation to incorporate the chunking idea of Wang and Arbib with the anticipation model so that the latter can acquire sequences whose set degree is much greater than r.

Chunking is one of the fundamental characteristics of human information processing [30], [37]. Though the present model and the model of Wang and Arbib [42] have addressed some aspects of chunking, the general issue of automatic chunking is very challenging and remains unsolved. It is not even clear what constitutes a chunk in general. In this paper, a chunk corresponds to a repeated subsequence. This is a reasonable definition in the present context. The anticipation model, through its mechanism of context learning, provides a neural network basis for forming such chunks. On the other hand, this definition of a chunk does not capture the richness of general chunking. We realize that chunking depends critically on the STM capacity [30]. Different people, however, may have different ways of chunking the same sequence in order to overcome STM limitations and memorize the sequence. Chunking also depends on general knowledge. For example, we tend to chunk a 10-digit telephone number in the U.S. into three chunks: The first three digits that correspond to an area code, then the next three digits to a district code, and the last four digits. But the same 10-digit number may well be chunked in a different way in a different country.

Theorem 1, Remark 2, and the property of capacity-independent incremental learning together enable the anticipation model to perform long-term automatic learning of temporal patterns. The system is both adaptive and stable, and its long-term memory increases gradually as learning episodes extend. Thus, the anticipation model provides a *sequential memory*, which can store and recall a large number of complex sequences. While sequences can be acquired from time to time, they can also be forgotten as a result of interference. But retraining with a particular sequence takes less effort than learning the sequence from the scratch.

How does the anticipation model as a sequential memory compare with computer storage of symbol sequences (strings)? A typical computational scheme is to store each sequence independently, and recall it by its identifier. This is similar to Grossberg's outstar avalanche model [16]. In doing so, this method treats both simple sequences and complex sequences in the same way, and a complex sequence can be recalled just as easily as a simple one. Also, there is no interference when a new sequence is acquired. Provided that an identifier can access its corresponding sequence by a very efficient search method, this method can support immediate recall. This method of storing sequences is used for many applications, such as storing book titles for library search and various pieces of music, and it is characteristic of the computer database approach [12]. The anticipation model differs from this method in several ways. First, our model supports not only sequence recall by its identifier but also partial recall by a context. Second, the computer method stores recurring subsequences as different copies, and the anticipation model stores them as a unique copy. As a result, our model yields a significantly more efficient use of memory space. Finally, though the computer method does not produce interference, the anticipation model is always adaptive and its long-term memory increases gradually with learning and recall experience. As a result of long-term adaptation and overlaps in storage, the anticipation model permits a dimension of recall flexibility that is missing from the computer method. For example, the anticipation model is more reliable because being self-organized it exhibits graceful degradation if detectors or their connections are damaged. Also, the anticipation model has the potential to support cognitive investigation of temporal learning.

APPENDIX

Here we provide a complete definition for the original anticipation model [43]. See Section II-B for notations. The activity of detector i is defined as[2]

$$E_i(t) = g\left(\sum_{jk} W_{i,jk}\, g(V_{jk}(t), A_i), \theta_i\right) \tag{A1}$$

$$g(x, y) = \begin{cases} x & \text{if } x \geq y \\ 0 & \text{otherwise} \end{cases} \tag{A2}$$

where θ_i is an adjustable threshold for detector i, which may be increased when the detector wins winner-take-all competition in the detector layer, to be discussed later. See (A5) for the definition of A_i. The activity of the kth SR unit of the jth

[2] Notice the difference between (A1) here and the corresponding equation in [43]—the gating of V_{jk} by the inner g function is included in (A1). The gating makes the model robust in the presence of initial variations. We thank C. Rosenberg for pointing this out.

assembly is

$$V_{jk}(t) = \begin{cases} I_j(t) & \text{if } k = 1 \text{ (head unit)} \\ \max[0, V_{j,k-1}(t-1) - \delta] & \text{otherwise} \end{cases} \tag{A3}$$

where δ is a decay parameter.

The detailed dynamics of winner-take-all competition in the detector layer can be found in [17]. The learning rule for each detector i is a Hebbian rule plus normalization to keep the overall weight constant (the normalized Hebbian rule)

$$\hat{W}_{i,jk}(t+1) = W_{i,jk}(t) + \alpha O_i(t) g(V_{jk}(t), A_i) \tag{A4a}$$

$$W_{i,jk}(t+1) = \frac{\hat{W}_{i,jk}(t+1)}{\alpha C + \sum_{j,k} \hat{W}_{i,jk}(t+1)} \tag{A4b}$$

where α is a learning rate. A_i is the sensitivity parameter of detector i, which is adapted by the following rule:

$$A_i = \begin{cases} 1 & \text{if } d_i = 0 \\ \max[0, 1 - \delta(d_i - 1)] & \text{if } d_i > 0. \end{cases} \tag{A5}$$

The threshold of the winning unit z in the detector layer is updated according to

$$\theta_i(t+1) = \theta_i(t) + O_i(E_i^*(t+1) - \theta_i(t)) \tag{A6}$$

where $E_i^*(t+1)$ is the activity of i based on the new weights, i.e., $E_i^*(t+1) = \Sigma_{jk} W_{i,jk}(t+1) g(V_{jk}(t), A_i)$. The activity of modulator i is defined as

$$M_i(t) = O_i(t-1) \sum_{j=1}^{n} R_{ij} I_j(t) \tag{A7}$$

where R_{ij} is a binary-valued weight of the connection from terminal j to modulator i. All R_{ij}'s are initialized to zero. The degree d_i of detector i is adjusted at time t as follows:

$$d_i = \begin{cases} d_i + O_i(t-1) & \text{if } M_i(t) = 0 \\ d_i & \text{otherwise.} \end{cases} \tag{A8}$$

Finally, one-shot learning is performed on the modulator of the winning detector z

$$R_{zj} = I_j(t). \tag{A9}$$

REFERENCES

[1] Y. Baram, "Memorizing binary vector sequences by a sparsely encoded network," *IEEE Trans. Neural Networks*, vol. 5, pp. 974–981, 1994.

[2] J. M. Barnes and B. J. Underwood, "Fate of first-list associations in transfer theory," *J. Exp. Psych.*, vol. 58, pp. 97–105, 1959.

[3] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, pp. 157–166, 1994.

[4] G. H. Bower, S. Thompson-Schill, and E. Tulving, "Reducing retroactive interference: An interference analysis," *J. Exp. Psych.: Learning, Memory, Cognition*, vol. 20, pp. 51–66, 1994.

[5] G. Bradski, G. A. Carpenter, and S. Grossberg, "STORE working memory networks for storage and recall of arbitrary temporal sequences," *Biol. Cybern.*, vol. 71, pp. 469–480, 1994.

[6] J. Buhmann and K. Schulten, "Noise-driven temporal association in neural networks," *Europhys. Lett.*, vol. 4, pp. 1205–1209, 1987.

[7] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Comput. Vision, Graphs, Image Processing*, vol. 37, pp. 54–115, 1987.

[8] C. C. Chandler, "Accessing related events increases retroactive interference in a matching test," *J. Exp. Psych.: Learning, Memory, Cognition*, vol. 19, pp. 967–974, 1993.

[9] R. L. Crooks and J. Stein, *Psychology: Science, Behavior, and Life*. Fort Worth, TX: Holt, Rinehart, and Winston, 1991.

[10] S. Diederich and M. Opper, "Learning of correlated patterns in spin-like glass networks by local learning rules," *Phys. Rev. Lett.*, vol. 58, pp. 949–952, 1987.

[11] J. L. Elman, "Finding structure in time," *Cognitive Sci.*, vol. 14, pp. 179–211, 1990.

[12] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*. Redwood City, CA: Benjamin/Cummings, 1994.

[13] R. M. French, "Using semidistributed representations to overcome catastrophic forgetting in connectionist networks," in *Proc. 13th Annu. Conf. Cognitive Sci. Soc.*, 1991, pp. 173–178.

[14] ——, "Dynamically contraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference," in *Proc. 16th Annu. Conf. Cognitive Sci. Soc.*, 1994, pp. 335–340.

[15] R. Granger, J. Whitson, J. Larson, and G. Lynch, "Non-Hebbian properties of long-term potentiation enable high-capacity encoding of temporal sequences," in *Proc. Nat. Academy Sci. USA*, vol. 91, 1994, pp. 10 104–10 108.

[16] S. Grossberg, "Some networks that can learn, remember, and reproduce any number of complicated space-time patterns, 1," *J. Math. Mechan.*, vol. 19, pp. 53–91, 1969.

[17] ——, "Adaptive pattern classification and universal recoding I: Parallel development and coding of neural feature detectors," *Biol. Cybern.*, vol. 23, pp. 121–134, 1976.

[18] ——, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Sci.*, vol. 11, pp. 23–63, 1987.

[19] I. Guyon, L. Personnaz, J. P. Nadal, and G. Dreyfus, "Storage and retrieval of complex sequences in neural networks," *Phys. Rev. A*, vol. 38, pp. 6365–6372, 1988.

[20] H. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley, 1991.

[21] T. M. Heskes and S. Gielen, "Retrieval of pattern sequences at variable speeds in a neural network with delays," *Neural Networks*, vol. 5, pp. 145–152, 1992.

[22] M. I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Proc. 8th Annu. Conf. Cognitive Sci. Soc.*, 1986, pp. 531–546.

[23] P. Kanerva, *Sparse Distributed Memory*. Cambridge, MA: MIT Press, 1988.

[24] I. Kantor and H. Sompolinsky, "Associative recall of memory without errors," *Phys. Rev. A*, vol. 35, pp. 380–392, 1987.

[25] C. A. Kortge, "Episodic memory in connectionist networks," in *Proc. 12th Annu. Conf. Cognitive Sci. Soc.*, 1990, pp. 764–771.

[26] J. K. Kruschke, "ALCOVE: An exemplar-based model of category learning," *Psych. Rev.*, vol. 99, pp. 22–44, 1992.

[27] K. S. Lashley, "The problem of serial order in behavior," in *Cerebral Mechanisms in Behavior*, L. A. Jeffress, Ed. New York: Wiley, 1951, pp. 112–146.

[28] R. P. Lippmann, "Review of neural networks for speech recognition," *Neural Computa.*, vol. 1, pp. 1–38, 1989.

[29] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psych. Learning Motivat.*, vol. 24, pp. 109–165, 1989.

[30] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *Psych. Rev.*, vol. 63, pp. 81–97, 1956.

[31] M. C. Mozer, "Neural-net architectures for temporal sequence processing," in *Predicting the Future and Understanding the Past*, A. Weigend and N. Gershenfeld, Eds. Redwood City, CA: Addison-Wesley, 1993, pp. 243–264.

[32] R. Ratcliff, "Connectionist models of recognition memory: Constraints imposed by learning and forgetting function," *Psych. Rev.*, vol. 97, pp. 285–308, 1990.

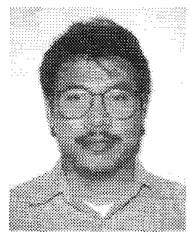
[33] G. J. Rinkus, "TEMECOR: An associative, spatio-temporal pattern memory for complex state sequences," in *Proc. World Congr. Neural Networks*, Washington, D.C., 1995, pp. I.442–I.448.

[34] W. A. Rodriguez, L. S. Borbely, and R. S. Garcia, "Attenuation by contextual cues of retroactive interference of a conditional discrimination in rats," *Animal Learning Behavior*, vol. 21, pp. 101–105, 1993.
[35] V. Ruiz de Angulo and C. Torras, "On-line learning with minimal degradation in feedforward networks," *IEEE Trans. Neural Networks*, vol. 6, pp. 657–668, 1995.
[36] N. E. Sharkey and A. J. C. Sharkey, "Understanding catastrophic interference in neural nets," Dep. Comput. Sci., Univ. Sheffield, U.K., Tech. Rep. CS-94-4, 1994.
[37] H. A. Simon, "How big is a chunk?" *Sci.*, vol. 183, pp. 482–488, 1974.
[38] S. A. Sloman and D. E. Rumelhart, "Reducing interference in distributed memories through episodic gating," in *From Learning Theory to Connectionist Theory: Essays in Honor of William K. Estes*, A. F. Healy, S. M. Kosslyn, and R. M. Shiffrin, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1992, pp. 227–248.
[39] H. Sompolinsky and I. Kanter, "Temporal association in asymmetric neural networks," *Phys. Rev. Lett.*, vol. 57, pp. 2861–2864, 1986.
[40] D. L. Wang, "Temporal pattern processing," in *Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA: MIT Press, 1995, pp. 967–971.
[41] D. L. Wang and M. A. Arbib, "Complex temporal sequence learning based on short-term memory," *Proc. IEEE*, vol. 78, pp. 1536–1543, 1990.
[42] ——, "Timing and chunking in processing temporal order," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 993–1009, 1993.
[43] D. L. Wang and B. Yuwono, "Anticipation-based temporal pattern generation," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 615–628, 1995.
[44] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Nonholographic associative memory," *Nature*, vol. 222, pp. 960–962, 1969.

DeLiang Wang (M'94) was born in Anhui, the People's Republic of China, in 1963. He received the B.Sc. degree in 1983 and the M.Sc degree in 1986 from Beijing University, China, and the Ph.D. degree in 1991 from the University of Southern California, Los Angeles.

From July 1986 to December 1987 he was with the Institute of Computing Technology, Academia Sniica, Beijing. He is currently an Assistant Professor in the Department of Computer and Information Science and the Center for Cognitive Science at the Ohio State University, Columbus. His present research interests include temporal pattern processing, auditory and visual perception, neural-network theories, and computational neuroscience.

Dr. Wang is a member of the IEEE Computer Society, the IEEE Systems, Man, and Cybernetics Society, the International Neural Network Society, and AAAS.



Budi Yuwono received the B.Eng. degree in industrial engineering and management from the Bandung Institute of Technology, Bandung, Indonesia, in 1986 and the M.Sc. degree in computer science from the Ohio State University, Columbus, in 1991. Currently he is a Ph.D. candidate at the Department of Computer and Information Science, the Ohio State University.

He has been a Research Associate since 1995 at the Department of Computer Science, the Hong Kong University of Science and Technology. His research interests include neural networks and information retrieval.