Experiments With Sequential Associative Memories

Stephen I. Gallant* and Donna J. King

College of Computer Science
Northeastern University
Boston, Ma. 02115 USA

Abstract

Humans are very good at manipulating sequential information, but sequences present special problems for connectionist models.

As an approach to sequential problems we have examined totally connected subnetworks of cells called *sequential associative memories (SAM's)*. The coefficients for SAM cells are unmodifiable and are generated at random.

A subnetwork of SAM cells performs two tasks:

1. Their activations determine a state for the network that permits previous inputs and outputs to be recalled, and
2. They increase the dimensionality of input and output representations to make it possible for other (modifiable) cells in the network to learn difficult tasks.

The second function is similar to the *distributed method*, a way of generating intermediate cells for non-sequential problems.

Results from several experiments are presented. The first is a robotic control task that required a network to produce one of several sequences of outputs when input cells were set to a corresponding 'plan number'.

The second experiment was to learn a sequential version of the parity function that would generalize to arbitrarily long input strings.

Finally we attempted to teach a network how to add arbitrarily long pairs of binary numbers. Here we were successful if the network contained a cell dedicated to the notion of 'carry'; otherwise the network performed at less than 100% for unseen sequences longer than those used during training.

Each of these tasks required a representation of state, and hence a network with feedback. All were learned using subnetworks of SAM cells.

Keywords: Connectionist Models, Learning, Sequences, Distributed Representation, Robotic Control

1 Introduction

A supervised, discrete SEQUENCE LEARNING PROBLEM involves learning to produce a sequence of correct output vectors

$$C^1, C^2, \cdots, C^k \quad (C^i \in \{+1, -1\}^g)$$

*Partially supported by National Science Foundation grant IRI-8611596. Thanks to Emmanouil Kalfaoglu for help with experiments. The first portion of this paper is revised from [Gallant 1987].

in response to a corresponding sequence of example input vectors

$$E^1, E^2, \cdots, E^k \ (E^i \in \{+1, -1\}^p)$$

where *order counts.*

For example if E^i and C^i are one dimensional vectors so that $E^i, C^i = \pm 1$ then we might desire

$$C^i = \begin{cases} +1 & \text{if } \sum_{j=1}^{i} \{E^j | E^j = +1\} \text{ is odd} \\ -1 & \text{otherwise.} \end{cases}$$

This is clearly a sequential version of the parity problem for a network with one input cell and one output cell. Note that the previous context ($\{E^j | j < i\}$) is important; knowing the value of E^i alone is not enough information to determine C^i.

There are many interesting sequential learning problems from a variety of disciplines: speech understanding (even for a single word), language understanding, robotic control (i.e. coordinated muscular activities that take advantage of sensory input), sequential fault detection problems, dynamic vision tasks, end even static vision problems if a single scene is analyzed by a sequence of visual fixations on parts of that scene.

Sequences can be difficult for connectionist models, particularly if they are of indefinite length. For some tasks (e.g. pronoun disambiguation) we cannot specify an n where the n previous inputs are always sufficient to allow correct outputs. Moreover if we remember the "window" of n previous inputs and apply brute force methods to every possible n-input configuration then the combinatorics becomes devastating. Therefore some sort of processing is required to extract and remember relevant information from prior context.

Recently there has been increased attention to sequence processing by connectionist models. Jordan [Jordan 1986a,Jordan 1986b] has employed a simple subnetwork of cells to represent the state of a network by encoding previous activations from each output cell in the activation value of a corresponding state cell using low order binary decimal bits. Thus if $u_i(T)$ is the activation of output cell u_i at time T then a corresponding state cell, ${u_i}^S(T)$ would have activation

$${u_i}^S(T) = \sum_{t=0}^{T} (0.5)^{T-t} u_i(t).$$

This method preserves information on *all* previous outputs but at the (inevitable) cost of requiring manipulation of high precision activations.

In this paper we explore highly connected subnetworks with feedback called *sequential associative memories* or *SAM*'s. SAM's are composed of linear discriminant cells with fixed, randomly generated weights. These subnetworks serve two important functions:

1. Their activations define a state for the network that allows previous inputs and outputs to affect the current output.

2. They help trainable cells elsewhere in the network to solve difficult (i.e. nonseparable) computational tasks requiring intermediate (or "hidden") cells.

SAM cell activations are discrete, so there is no problem with high precision activation values. However since a finite number of bits can only encode a finite amount of information, the SAM

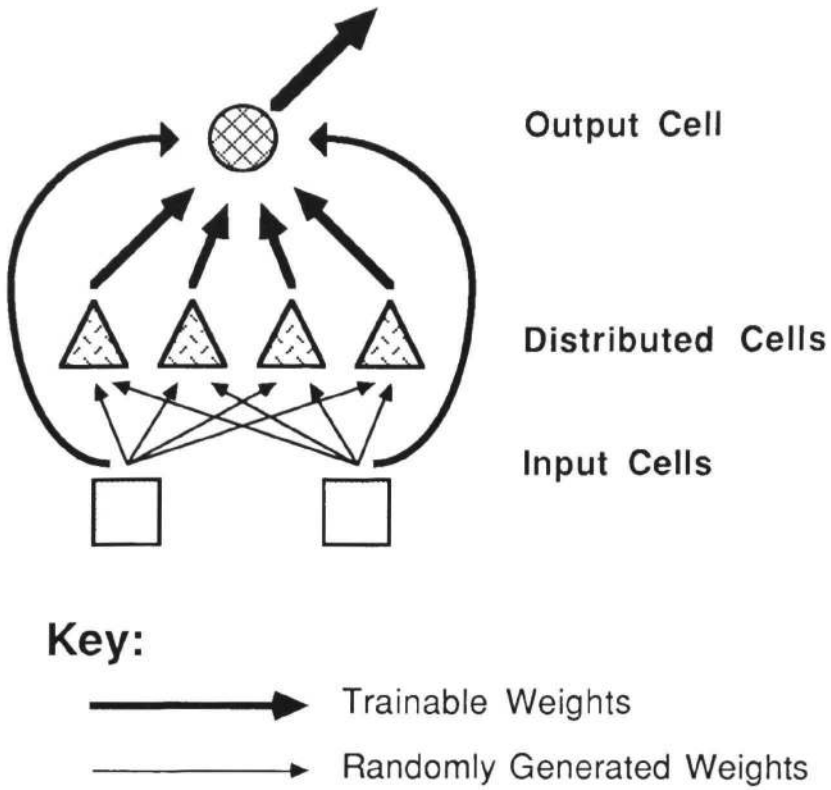

Figure 1: The Distributed Method.

cells cannot have perfect memory; they must 'forget' some information when their capacity is exceeded.

The following sections give a motivation for, and a more precise definition of, sequential associative memories and describe their role in several learning experiments.

2 The Distributed Method

We have previously described a technique called the *distributed method* for creating intermediate cells so that a network could learn nonseparable functions [Gallant 1986a, Gallant & Smith 1987]. See figure 1.

In this method all of the network cells (except input cells) are linear discriminants [Fisher 1936] (also known as threshold logic units [McCulloch & Pitts 1943] or perceptrons [Rosenblatt 1961]). All cells compute discrete activation values of $+1$, -1, or 0 according to whether the weighted sum of their inputs (plus a constant bias term) is > 0, < 0, or $= 0$ respectively. The weights for distributed cells are chosen at random[1] and remain fixed. Output cell weights are generated using the pocket algorithm [Gallant 1986b], a modification of perceptron learning.

Assume there are more distributed cells than input cells. For every setting of the input cells the activations of the distributed cells will form a distributed representation [Rumelhart & McClelland 1986, chap. 3] of the input that lies in a higher dimensional space than the original input. This higher dimensional representation eases the learning task for the output cells and, in

[1] We use integers between -5 and $+5$, but random real values between -1 and $+1$ are arguably better.

particular, makes it more likely that the transformed problem will be separable.

For n training examples our experiments have shown that adding between $\frac{1}{4}n$ and n distributed cells is usually sufficient to make (non-contradictory) data separable for the most difficult classical problems such as parity. Fewer suffice for more 'natural' problems. This bound is roughly consistent with a theoretical result by Cover [Cover 1965] for random functions. By restricting distributed cells to linear discriminants, however, we are also able to preserve robustness because a new input that is close to an example used in training the network is likely to produce the same output as that example. Had we used arbitrary random functions for distributed cells rather than restricting ourselves to the subclass of random *linear discriminants*, the increased probability of separability would have remained but the robustness would have been lost. For a fuller description of the distributed method see [Gallant & Smith 1987].

3 Sequential Associative Memories

The distributed method increases learning ability while preserving robustness (in the sense described), but this technique gives no help with sequences since the network is unaffected by previous inputs and outputs. What is missing is feedback and the notion of state to capture previous behavior.

In figure 2 we see a generalization of the distributed method where the added cells also receive inputs from previous network outputs and from each other. We call such a subnetwork a *sequential associative memory (SAM)*. The state of the network is the pattern of activations in the SAM cells, and it is influenced by previous inputs, outputs, and states. Moreover the SAM cells aid learning by the trainable output cells while still preserving robustness, just as with the distributed method.

3.1 Network Dynamics

The dynamic operation of the network for sequences is as follows:

1. Initialize SAM cell activations and output cell activations to 0.
2. Set the activations of input cells to the {first / next} set of inputs in a sequence.
3. For each SAM cell, compute its new activation value but do not change that activation until all other SAM cells have computed their new values. After computing the new activation values for all SAM cells, modify all the activations accordingly.
4. For each output cell, compute its new activation value and immediately change its activation to that value. These activations are the {first / next} network outputs for the sequence.
5. Go to 2.

Note that all SAM cells can recompute and update their activations in parallel, making this architecture attractive for parallel hardware implementations.

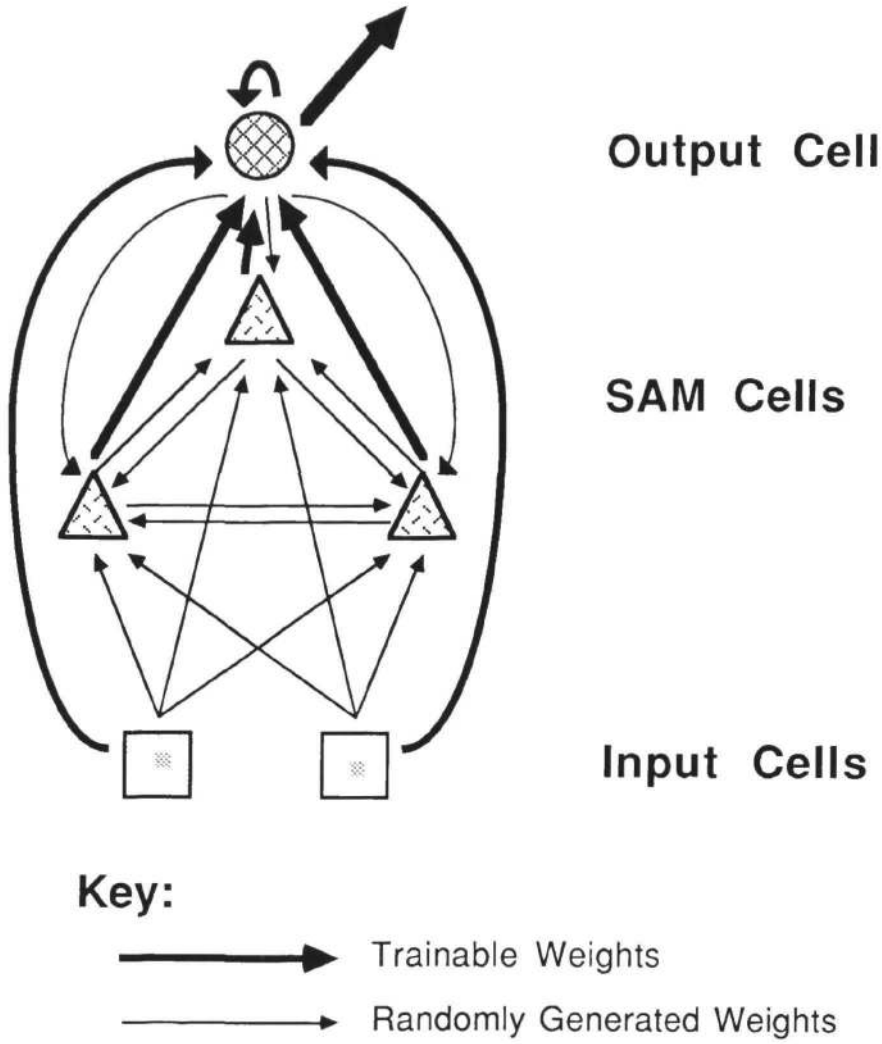

Figure 2: Network with 3 SAM Cells.

3.2 Input Gain

For some tasks it is necessary to multiply activations for input and output cells by a *gain factor* to increase their effect on the SAM cells. If this is not done the SAM cells are not sensitive enough to input and output values and performance can be poor. The higher the gain the more the emphasis is placed upon *recent* inputs and outputs; thus gain might be viewed as a parameter that determines an inductive bias for the learning algorithm. A good setting for the gain factor varies with the number of SAM cells and the problem type.

The easiest time to implement gain is at the time when the (random) SAM connection weights are generated; we simply multiply the weights from input or output cells to SAM cells by the gain factor.

4 Experiments

4.1 Robotic Control

We tried SAM networks with a robotic control problem similar to the one studied by Jordan [Jordan 1986b]. The network had 4 input cells that were set to one of 16 possible 'plan numbers'. Every plan number corresponded to a sequence of outputs from a single output cell over 4 iterations; for example plan $<+1,-1,-1,+1>$ might correspond to the sequence $\{-1,-1,+1,-1\}$. We set the input cells to the same plan number at each time step, and the correspondence between plan numbers and correct output sequences was chosen at random.

Notice that the network cannot determine its correct output based solely upon its (current) input (even if its last output were available). Therefore a 'state' must be maintained by the system and a feedback network is required.

For a network with 4 input cells and 1 output cell we found that 16 sequences of length 4 could be learned reliably by adding 40 SAM cells (using gain 1) in about 5000 iterations (consuming several minutes of CPU time).

We then tried a harder problem involving 5 input cells and 32 sequences of length 5. Here about 150 SAM cells with gain 1 sufficed for reliable 100% performance after about 8000 iterations.

Finally we tried problems with 6 input cells and 64 sequences of length 6. Here 350 SAM cells with gain 1 were required to reliably give 100% performance after about 4000 iterations.

Our conclusion was that SAM cells allow networks to learn simple, sequential robotic tasks in reasonable time.

It is interesting to analyze this task from an information theory viewpoint. If we consider the random coefficients of SAM cells as carrying no information (since they are not specific to the particular set of plans and output sequences being represented) then all information is carried in the trainable weights leading to output cells. The number of such weights equals the number of cells in the network. In our experiments the number of weights was approximately the same as the number of output bits that were to be learned, i.e. the number of plans multiplied by the length of each sequence. For example with 5 input cell sequences and 150 SAM cells there are $5 + 150 + 1$ trainable weights and these can correctly produce 32×5 total output bits. The magnitudes of the weight values did not appear to grow with the difficulty of the problem.

The total number of output bits gives a lower bound for the encoding size of this task; therefore the output cell weights appear to give reasonably efficient encodings because they seem to be growing linearly with the lower bound.

4.2 Sequential Parity

For our second experiment we set out to teach a network the concept of parity as a sequential task. Our goal was for the network to compute parity *for arbitrarily long input sequences*, an impossible task for a feed-forward network.

We used a network with one input cell, one output cell and a collection of SAM cells. The problem was made somewhat harder by allowing the output to see only the current input and the SAM activations but not the previous output. We found that only 10 SAM cells (with gain of 10) were sufficient to quickly and reliably learn parity for arbitrary length sequences.[2] Five SAM

[2]We tested the resulting networks on 50 unseen sequences of length 100 to verify that they had learned parity with high, but not 100%, certainty.

cells were usually too few for this task. To our knowledge this is the first connectionist network to learn parity for arbitrary length sequences.

4.3 Sequential Addition

We also attempting to teach a network sequential (right-to-left) addition of pairs of arbitrarily long numbers using networks composed of 2 input cells, 1 output cell and a collection of SAM cells as in figure 2. This is an easy task if we dedicate an extra output cell to the computation of 'carry'; 10 SAM cells usually suffice to quickly learn addition of arbitrary length numbers with perfect generalization [Gallant 1987].

Without such a dedicated carry cell the problem is much harder because the notion of carry must be represented in the collective activations of the SAM cells.

For this problem a network with 300 SAM cells (and gain 300) learned to add a training set consisting of 100 pairs of 40 bit numbers using 2500 iterations.[3] Testing on groups of 20 (unseen) pairs of numbers of various lengths the network generalized as follows:

Length of numbers added:	20	30	40	50	75	100
% Sequences correct:	100%	100%	95%	100%	95%	95%

For a sequence to be counted as correct in the testing we required that *every* output bit be correct. Although the network failed to precisely learn the concept of addition, we considered it encouraging that it was able to generalize from 40-bit pairs of numbers to 75-bit and 100-bit pairs with 95 per cent of the sequences totally correct.

The improvement in generalization over those tests reported in [Gallant 1987] we attribute to two factors: higher gain settings and the use of longer sequences for training. Experiments are continuing on this problem.

5 Concluding Remarks

We have described a connectionist model with feedback and state called sequential associative memories and several experiments involving sequences. The most important current research involving SAM networks is to determine how well they train and generalize for difficult sequential pattern recognition problems such as speech understanding, fault detection, natural language, and possibly vision. While we hold little hope that SAM networks *alone* could tackle these difficult problems, we are more optimistic about their *combination* with other techniques (such as "windows" and unsupervised learning of input features during a pre-processing step).

We believe that the problem of manipulating sequences represents an important challenge for connectionist research and for machine learning in general.

References

[Cover 1965]
Cover, T. M. Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. IEEE Trans. Electronic Computers, Vol. 14, 326-334, 1965.

[3] We count an iteration as one presentation of a single training sequence, as contrasted to a 'sweep' through all 100 sequences.

[Fisher 1936]
Fisher, R. A. The use of multiple measurements in taxonomic problems. Ann. Eugenics, 7, Part II, 179-188. Also in *Contributions to Mathematical Statistics* (1950) John Wiley, New York.

[Gallant 1986a]
Gallant, S. I. Three Constructive Algorithms for Network Learning. Proc. Eighth Annual Conference of the Cognitive Science Society, Amherst, Ma., Aug. 15-17, 1986.

[Gallant 1986b]
Gallant, S. I. Optimal Linear Discriminants. Proc. Eighth International Conference on Pattern Recognition, Paris, France, Oct. 28-31, 1986.

[Gallant & Smith 1987]
Gallant, S. I., and Smith, D. Random Cells: An Idea Whose Time Has Come and Gone...And Come Again? IEEE International Conference on Neural Networks, San Diego, Ca., Vol. II, 671-678, June 1987.

[Gallant 1987]
Gallant, S. I. Sequential Associative Memories. Technical Report NU-CCS-87-20, Northeastern University College of Computer Science.

[Jordan 1986a]
Jordan, M. I. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. Proceedings of the Eighth Annual Conference of the Cognitive Science Society, Amherst, Ma., 1986

[Jordan 1986b]
Jordan, M. I. Serial Order: A Parallel Distributed Processing Approach. Institute for Cognitive Science Report 8604, University of California, San Diego, May 1986.

[McCulloch & Pitts 1943]
McCulloch, W. S. & Pitts, W. H. A logical calculus of the ideas imminent in nervous activity. Bulletin of Math. Biophysics, 5, 115-133 (1943).

[Rosenblatt 1961]
Rosenblatt, F. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms.* Spartan Press, Washington, DC.

[Rumelhart & McClelland 1986]
D. E. Rumelhart & J. L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the Microstructures of Cognition.* MIT Press.