Attractor dynamics and parallelism in a connectionist sequential machine

Michael I. Jordan
Department of Computer and Information Science
University of Massachusetts

ABSTRACT

Fluent human sequential behavior, such as that observed in speech production, is characterized by a high degree of parallelism, fuzzy boundaries, and insensitivity to perturbations. In this paper, I consider a theoretical treatment of sequential behavior which is based on data from speech production. A network is discussed which is essentially a sequential machine built out of connectionist components. The network relies on distributed representations and a high degree of parallelism at the level of the component processing units. These properties lead to parallelism at the level at which whole output vectors arise, and constraints must be imposed to make the performance of the network more sequential. The sequential trajectories that are realized by the network have dynamic properties that are analogous to those observed in networks with point attractors (Hopfield, 1982): learned trajectories generalize, and attractors such as limit cycles can arise.

INTRODUCTION

One of the arguments for "connectionist" or "parallel, distributed processing" networks has been that they have properties that seem to reflect processes at which humans are most naturally proficient (Hinton & Anderson, 1981; Rumelhart & McClelland, 1986). These properties include the ability to generalize from instances, the ability to deal with partial information, and insensitivity to noise. It has been suggested that it might be advisable to base theories on such primitives rather than on those associated with sequential, symbolic processing. These arguments have been made mostly in the context of models dealing with the interpretation of incoming data, or with mappings from one set of data to another. However, one need only consider the fluency of human speech to see that humans are also very good at certain kinds of sequential behavior. Furthermore, such behavior is often characterized by a high degree of parallelism, fuzzy boundaries, and insensitivity to perturbations — properties which are difficult to capture in a formalism in which the underlying primitive is a sequential processor, but which are more natural in a connectionist system. In this paper, I consider the problem of parallelism in speech production and suggest a connectionist architecture that can exhibit behavior similar to that shown in speech. My approach is based on the recent work on learning by Rumelhart, Hinton, and Williams (1986) and is related to previous work by Henke (1966), Kohonen, Lehtio, and Oja (1981), and Rumelhart and Norman (1982).

COARTICULATION

Much of the complexity of describing sequential processes in speech production comes from the fact that speech gestures associated with nearby phonemes can overlap in time. Such overlap, or *coarticulation*, is ubiquitous in utterances and can be quite complex, given the many degrees of freedom of the speech apparatus. It is possible to see gestures that anticipate future phonemes, referred to as *forward coarticulation*, as well as perseveratory gestures, or *backward coarticulation*. The overall effect of coarticulation is to make the utterance more smooth by merging nearby phonemes and to allow speech to proceed faster than would otherwise be possible by taking advantage of opportunities for the parallel execution of movements.

Several studies have investigated coarticulation by recording articulator trajectories during utterances. Moll and Daniloff (1971) showed that in an utterance such as "freon", the velar opening for the nasal /*n*/ can begin as early as the first vowel, thereby nasalizing the vowels. [1] Benguerel and Cowan (1974) studied phrases such as "une sinistre structure," in which there is a string of the six consonants /*strstr*/ followed by the rounded vowel /*y*/. [2] They showed that lip-rounding for the /*y*/ can begin as early as the first /*s*/, an example of forward coarticulation over six phonemes.

One way to characterize these examples is to say that if certain degrees of freedom are not being used in the production of a particular sound, then they may anticipate or perseverate aspects of other phonemes in the utterance so that performance becomes more parallel. However, such a conception of coarticulation ignores the constraints which must be imposed on the parallelism. Certain anticipatory gestures, for example, would inflict too much change on the sound currently being produced, and there must therefore be a way to prevent such coarticulations while allowing others. In the case of "freon", for example, the velum is allowed to open during the production of the vowels because the language being spoken is English. In a language such as French, in which nasal vowels

[1] The velum is a muscular tissue that opens to allow air to pass between the pharynx and the nasal cavities.

[2] The vowel /*y*/ is the "u" in "tu", and is somewhat like pronouncing the English sound "ee" with rounded lips.

are different phonemically from non-nasal vowels, the velum would not be allowed to coarticulate. Thus the articulatory control system cannot blindly anticipate articulations, but must be sensitive to phonemic distinctions in the language being spoken.

The implementation of constraints on parallelism is complicated by the fact that the constraints cannot be encoded as relations between whole phonemes, but must be specific to particular phonemic structure. For example, in the case of */strstry/*, the *rounding* of the */y/* can be anticipated during the preceding consonants, but the *voicing* of the */y/* cannot, because that would change the phonemic identities of the consonants (for example, the */s/* would become a */z/*). Other features of the */y/*, such as those specifying tongue position, may be more or less constrained, depending on the particular allowable variations of the consonants. Again, such knowledge cannot come from consideration of strategies of articulation, but must reflect higher-level phonemic constraints.

Thus, speech presents an interesting control problem in which constraints of various kinds are imposed on the particular patternings of parallelism and sequentiality that can be obtained in an utterance. What I wish to discuss in the remainder of this paper is an approach to this problem based on connectionist mechanisms.

CONNECTIONIST NETWORKS

General discussions of connectionist networks can be found in Feldman and Ballard (1982) and Rumelhart and McClelland (1986). For present purposes, the main features of the networks that are relevant involve distributed representations, nonlinearities, and learning.

A one layer network with no recurrent connections computes a function from the vector of activation of its input units to the vector of activation of its output units. It is possible for such a network to learn to make associations between input vectors and output vectors. This can be done by an error-correcting learning rule that changes the weights coming in to each output unit in proportion to the difference between the actual output of that unit and the desired output (Widrow & Hoff, 1960).

An important property of such networks, which is due to the weighted sums that units form in determining their activations, is that similar input vectors tend to produce similar output vectors. Many connectionist approaches take advantage of this property by representing entities as distributed patterns of activation, so as to achieve a kind of automatic generalization between similar patterns (Hinton & Anderson, 1981).

A one layer network has only a single weight matrix and is restricted to linear mappings. By allowing more layers, with nonlinear activation functions on intermediate units, it is possible to implement arbitrary nonlinear mappings. Until recently, the learning rules in these networks were restricted to the single layer case. However, several rules have now been developed for multilayer networks that allow essentially arbitrary associations to be formed (Ackley, Hinton, & Sejnowski, 1985; Barto & Anandan, 1985; Rumelhart, Hinton, & Williams, 1986). The back-propagation rule of Rumelhart, Hinton, and Williams has been used in simulations of the network discussed in this paper. The back-propagation rule is an error-correction procedure that generalizes the Widrow-Hoff rule. As before, errors are generated at the output units by comparing the actual outputs to the desired outputs, and these errors are used to change the weights of the output units. The errors are then propagated back into the network to provide intermediate units with error signals so that they can change their weights.

A NETWORK ARCHITECTURE FOR SEQUENTIAL PERFORMANCE

Let there be some sequence of *actions* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_s$, which are to be produced in order in the presence of a *plan* $\mathbf{p}$. Each action is a vector in a feature or parameter space, and the plan can be treated as an action produced by a higher level of the system. The plan is assumed to remain constant during the production of the sequence, and serves primarily to designate the particular sequence which is to be performed.

We wish to construct a network that can perform arbitrary sequences by taking a plan as input and producing the corresponding sequence. One approach is to explicitly represent the state of a sequential machine as an activation vector and to produce actions by evaluating a function from states to actions. At each moment in time, an action is chosen based on the current state $\mathbf{s}$, and the state is then updated to allow the next action to be chosen. Thus, there is a function f which determines the output action $\mathbf{x}_n$ at time n,

$$\mathbf{x}_n = f(\mathbf{s}_n, \mathbf{p})$$

and a function g which determines the state $\mathbf{s}_{n+1}$,

$$\mathbf{s}_{n+1} = g(\mathbf{s}_n, \mathbf{p}), \tag{1}$$

where both functions depend on the constant plan vector as well as the current state vector. Following the terminology of automata theory (Booth, 1969), f will be referred to as the *output* function, and g will be referred to as the *next-state* function. [3]

The basic network architecture is shown in Figure 1. The entities in the state equations — plans, states, and outputs — are all assumed to be represented as distributed patterns

[3] From the definition, it can be seen that the plan $\mathbf{p}$ plays the role of the input symbol in a sequential machine. The use of the term "plan" is to emphasize the assumption that $\mathbf{p}$ remains constant during the production of the sequence. That is, we are not allowed to assume temporal order in the input to the system.

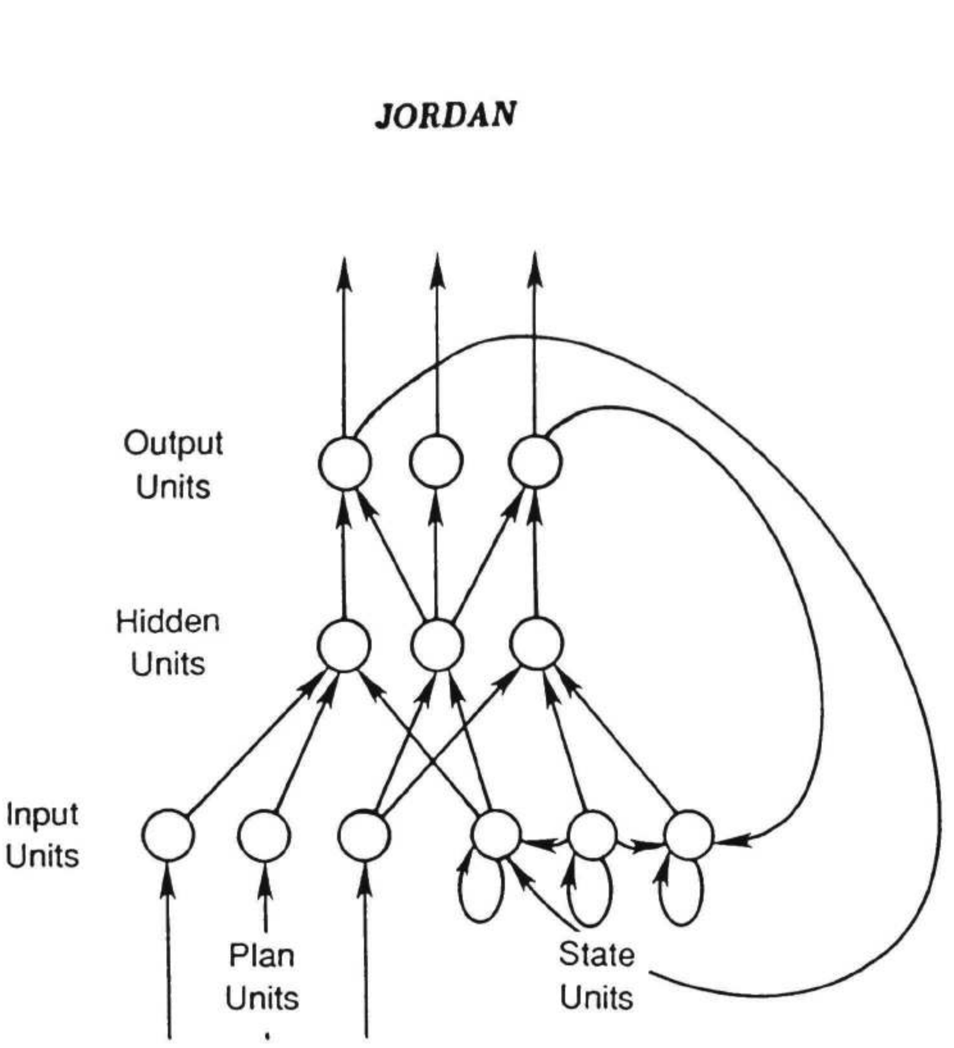

Figure 1: Basic network architecture. (Not all connections are shown).

of activation on separate pools of processing units. The plan units and the state units together serve as the input units for a multilayer network. This network implements the output function through weighted connections from the plan and state units to the output units. The output function is generally nonlinear, as will be discussed below, therefore it is also necessary to have hidden units in the path from the plan and state units to the output units. Finally, the next-state function is implemented with recurrent connections from the state units to themselves, and from the output units to the state units. This allows the current state to depend on the previous state and on the previous output (which is itself a function of the previous state and the plan).

The network can learn to produce sequences of actions by changing the weights in the network. Assume that the recurrent connections implementing the next-state function are given fixed values (particular choices for these values are discussed below). At each time step, an activation vector composed of the plan and the state is present on the input units, and an association can be learned from this input vector to a desired output vector. Clearly, one requirement for the network to be able to learn arbitrary sequences is that the next-state function produce distinguishable state vectors at each time step. It is not necessary that these vectors be different between sequences, because the plan serves to distinguish the sequences. A second requirement is that there be no restrictions on the form of the associations that can be learned (such as a linearity restriction). This requirement is met by using the back-propagation learning rule. Note that the ability to learn arbitrary sequences does not imply that all sequences are equally easy to learn; indeed, the network will have more difficulty in learning and performing sequences when distinctions must be made similar states and plans.

A further requirement must be imposed on the next-state function so that the results on parallelism will hold: *State vectors at nearby points in time must be similar.* There are many ways to choose the recurrent connections so as to achieve this continuity property. One particular choice, which has been used in many of the simulations of the network, is based on a conception of the state as representing the *temporal context* of actions. Consider the case of a sequence with a repeated subsequence or a pair of sequences with a common subsequence. It seems appropriate, given the positive transfer which can occur in such situations as well as the phenomena of capture errors (Norman, 1981), that the state should be similar during the performance of similar subsequences. One way to achieve this is to define the state in terms of the actions being produced. However, the representation must provide an extensive enough temporal context so that there are no ambiguities in cases involving repeated subsequences. If the state were to be defined as a function of the last n outputs, for example, then the system would be unable to perform sequences with repeated subsequences of length n, or to distinguish between pairs of sequences with a common subsequence of length n. To avoid such problems, the state can be defined as an exponentially weighted average of past outputs, so that the arbitrarily distant past has some representation in the state, albeit with ever-diminishing strength. This representation of the state can be obtained if each output unit feeds back to a state unit with a weight of one, if each state unit feeds back to itself with a weight μ, and if the state units are linear. [4] In this case, the state at time n is given by

$$\begin{aligned}\mathbf{s}_n &= \mu\mathbf{s}_{n-1} + \mathbf{x}_{n-1}\\ &= \sum_{\tau=1}^{n-1} \mu^{\tau-1}\mathbf{x}_{n-\tau}.\end{aligned}$$

Since this representation of the state is an average, it tends to have the desired property that states nearby in time are similar. The similarity depends on the particular actions that are added in at each time step and on the value of μ. In general, however, with sufficiently large values of μ, the similarity extends forward and backward in time, growing weaker with increasing distance.

Other possible representations of the state are discussed in Jordan (1985). The major differences between different representations is in the particular metrics they induce on the

[4] The linearity assumption gives the state a simple interpretation and also gives the state units a more extended dynamic range, but is not essential for the operation of the network.

difficulty of learning and performing particular sequences and also the kinds of generalizations that can be made between sequences. It is also possible to consider learning of the next-state function. Indeed, the back-propagation algorithm applies to the case of recurrent networks, although in a more complex form, requiring units to store histories of their activations (Rumelhart, Hinton, & Williams, 1986). However, in the current framework, there is little to be gained by learning the next-state function; all the hard work can be done in learning the output function.

LEARNING AND PARALLELISM

The network as described thus far would appear to be strictly sequential: there is no overlap between neighboring actions. This is indeed the case and it is necessary to modify the form in which desired output vectors are specified to see that the network is in fact capable of highly parallel performance.

The form that desired output vectors are assumed to take is a generalization of the approach used in traditional error-correction schemes (Duda & Hart, 1973; Rosenblatt, 1961; Rumelhart, Hinton, & Williams, 1986; Widrow & Hoff, 1960). Rather than assuming that a value is specified for each output unit, it is assumed that in general there are *constraints* specified on the values of the output units. Constraints may specify a range of values which an output unit may have, a particular value, or no value at all. This latter case is referred to as a "don't-care condition." It is also possible to consider constraints which are defined among output units. For example, the sum of the activations of a set of units may be required to have a particular value.

Constraints enter into the learning process in the following way: If the activation of an output unit fits the constraints on that unit, then no error corrections are instigated from that unit. If, however, a constraint is not met, then the error is defined as a proportion of the degree to which that constraint is not met, and this error is used in the normal way to change weights towards a configuration in which the constraint is met. An example of this process is shown in Figure 2 for a desired output vector with three specified values and two don't-care conditions (represented by stars). As shown in the figure, errors are propagated from only those units where constraints are imposed. In the case of constraints among units, it is possible to impose constraints on units having fixed connections from the output units. Errors generated at these units are propagated back to the output units. This process is sketched in Figure 3, where the constraints $x_1 + x_2 = .6$ and $x_2 + x_3 = .4$ are imposed. Note that if many constraints are imposed on the same unit, the errors are simply added together, and the network will eventually find an activation value for the unit that satisfies all of the constraints (given that such a value exists).

Consider now the case in which desired output vectors specify values for only a single

$$\mathbf{x}_i = \begin{bmatrix} * \\ .7 \\ * \\ .3 \\ * \end{bmatrix}$$

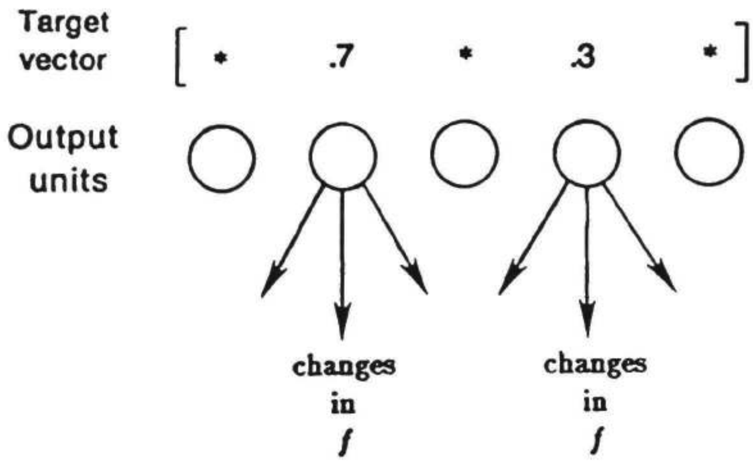

Figure 2: Learning with don't-care conditions.

output unit. This is essentially the case of local representations for actions, in which the network is essentially being instructed to activate its output units in a particular order. Suppose, for example, that a network with three output units is learning the sequence

$$\begin{bmatrix} 1 \\ * \\ * \end{bmatrix}, \begin{bmatrix} * \\ 1 \\ * \end{bmatrix}, \begin{bmatrix} * \\ * \\ 1 \end{bmatrix}$$

At each time step, errors are propagated from only a single output unit, so that activation of that unit becomes associated to the current state. Associations are learned from $\mathbf{s}_1$ to activation of the first output unit, from $\mathbf{s}_2$ to activation of the second output unit, and from $\mathbf{s}_3$ to activation of the third output unit. [5]

[5] where $\mathbf{s}_i$ denotes the activation of the state units at time i. I am ignoring the plan vector to simplify the exposition.

$$\mathbf{x}_i = \begin{bmatrix} x_1 + x_2 = .6 \\ x_2 + x_3 = .4 \end{bmatrix}$$

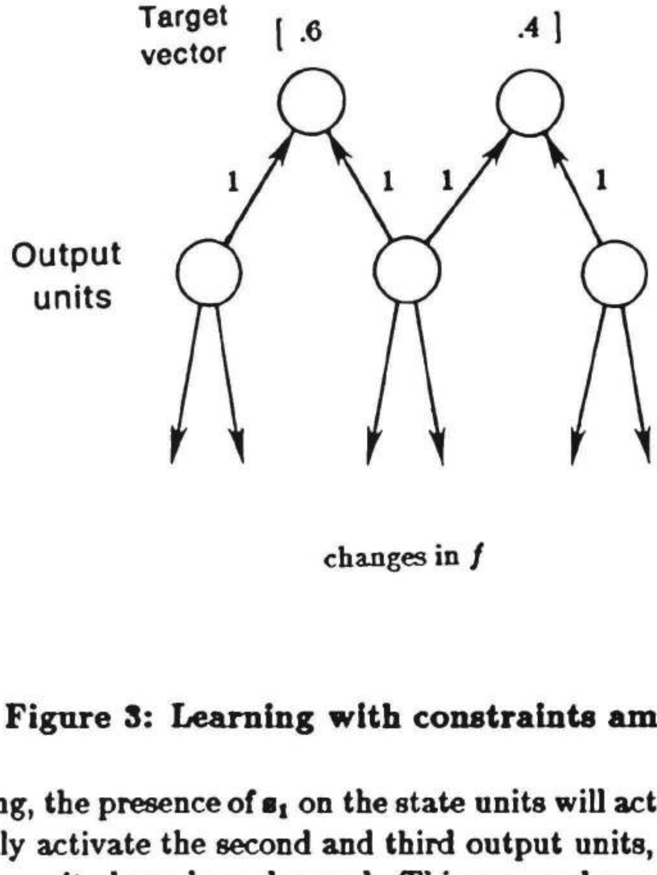

Figure 3: Learning with constraints among units.

After learning, the presence of $\mathbf{s}_1$ on the state units will activate the first output unit. It will also partially activate the second and third output units, even though no associations from $\mathbf{s}_1$ to these units have been learned. This occurs because $\mathbf{s}_1$ is similar to $\mathbf{s}_2$ and $\mathbf{s}_3$ (given the requirement made of the next-state function) and similar inputs tend to produce similar outputs in these networks. The associations made to $\mathbf{s}_2$ and $\mathbf{s}_3$ also generalize, so that after learning, the network will likely produce a sequence such as

$$\begin{bmatrix} 1 \\ .8 \\ .6 \end{bmatrix}, \begin{bmatrix} .8 \\ 1 \\ .8 \end{bmatrix}, \begin{bmatrix} .6 \\ .8 \\ 1 \end{bmatrix},$$

where at each time step, there are parallel activations of all output units. If the network is driving a set of articulators that must travel a certain distance, or have a certain inertia, then it will be possible to go faster with these parallel control signals than with signals where only one output unit can be active at a time.

The foregoing example is simply the least constrained case and further constraints can be added. Suppose, for example, that the second output unit is not allowed to be active during the first action. This can be encoded in the constraint vector for the first action so that the network is instructed to learn the sequence

$$\begin{bmatrix} 1 \\ 0 \\ * \end{bmatrix}, \begin{bmatrix} * \\ 1 \\ * \end{bmatrix}, \begin{bmatrix} * \\ * \\ 1 \end{bmatrix}.$$

After learning, the output sequence will likely be as follows:

$$\begin{bmatrix} 1 \\ 0 \\ .6 \end{bmatrix}, \begin{bmatrix} .8 \\ 1 \\ .8 \end{bmatrix}, \begin{bmatrix} .6 \\ .7 \\ 1 \end{bmatrix},$$

where the added constraint is now met. In this example, the network must block the generalization that is made from $\mathbf{s}_2$ to $\mathbf{s}_1$. In general, the ability to block generalizations in this manner implies the need for a nonlinear output function.

As further constraints are added, there are fewer generalizations across nearby states that are allowed, and performance becomes less parallel. Minimal parallelism will arise when neighboring actions specify conflicting values on all output units, in which case the performance will be strictly sequential. Maximal parallelism should be expected when neighboring actions specify values on non-overlapping sets of output units. Note that there is no need to invoke a special process to program in the parallelism. Essentially, the system generalizes naturally across similar state vectors, and given that state vectors nearby in time are similar, the generalizations act so as to spread actions in time. In most cases, it will be more difficult for the system to learn in the more sequential case when there are more constraints imposed on the system which block the generalizations. These observations are summarized in Figure 4, which shows the relationships between constraint vectors and parallelism.

ATTRACTOR DYNAMICS

The properties of the system that lead to parallel performance also make the system relatively insensitive to perturbations. Suppose that the system has learned a particular sequence and that during performance of the sequence the state is perturbed somewhat. Given that similar states tend to produce similar outputs, the output of the system will not be greatly different from the unperturbed case. This would suggest that the network will perform a sequence which is a "shifted" version of the learned sequence. However, a stronger property appears to hold: The learned sequences become attractors for nearby

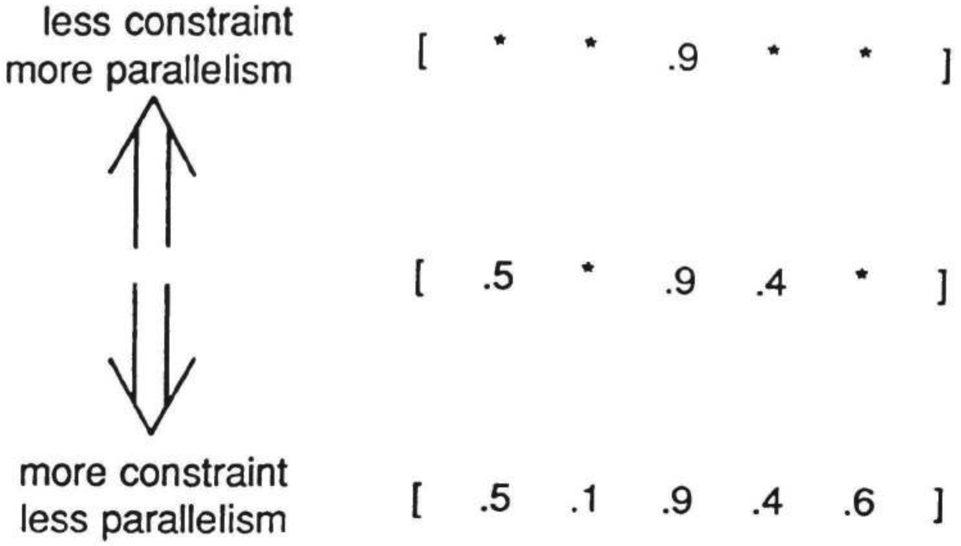

Figure 4: Relationships between constraint and parallelism.

regions of the state space and perturbed trajectories return to the learned trajectories. This property is demonstrated in Figures 5 and 6. A network with two output units learned to follow a square in the two-dimensional space which corresponds to the activations of the output units. As shown in the figures, when the network was started at other points in the space, the trajectories moved toward the square. This occurred whether the trajectories began inside or outside the square, showing that the square is a *limit cycle* for the system. For a dynamical system to have limit cycles, it is necessary that the system be nonlinear (Hirsch & Smale, 1974), which further demonstrates the need for the output function to be nonlinear.

More globally, a network which has learned to produce several different cyclical sequences may have several regions of the state space which are attractor basins for the learned cycles. If the network is started in one of these basins, then the performed trajectory will approach the learned cycle, with the part of the cycle which first appears depending on where in the basin the network is started relative to the configuration of the cycle. The network can be regarded as a generalization of a content-addressable memory (cf. Hopfield, 1982) in which the memories correspond to cycles or other dynamic trajectories rather than static points.

Constraints on the output units in general define regions through which trajectories can pass. The network is free to choose a particular trajectory within the region, and this tends to be done in a way so as to avoid sharp changes in the trajectory. Whatever trajectory is chosen by the network, it will tend to generalize so as to become an attractor

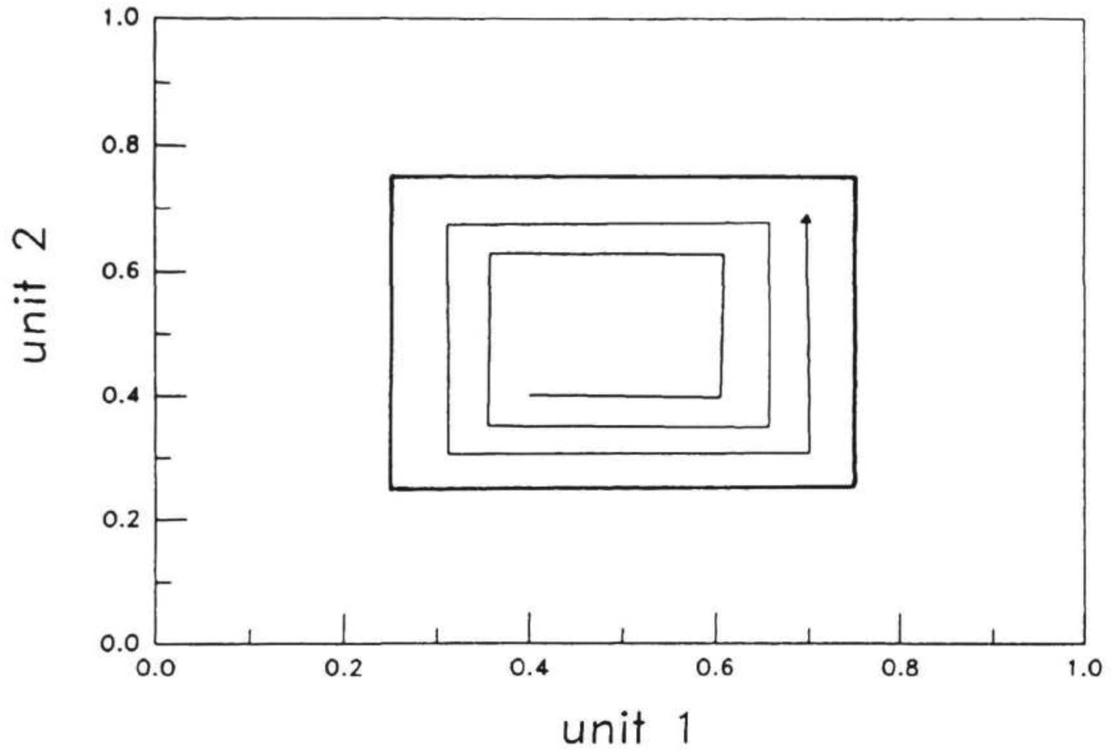

Figure 5: The activations of the two output units plotted with time as a parameter. The square is the trajectory that the network learned, and the spiral trajectory is the path that the network followed when started at the point (.4, .4).

for the surrounding space.

APPLICATIONS TO SPEECH PRODUCTION

In the case of speech, the constraint lists used by the learning process can be taken to encode knowledge about the phonetic structure of the language, and it is natural to identify these constraint lists with phonemes. Thus, in the current framework, the role of phonemes is to constrain the dynamical process that produces utterances by changing parameters of the process until the constraints are met. The constraints that define phonemes are themselves independent of context: They specify in what ways a phoneme can be altered by its context, without specifying values for particular contexts. During the learning process, parallel interactions between nearby phonemes can arise as long as they do not

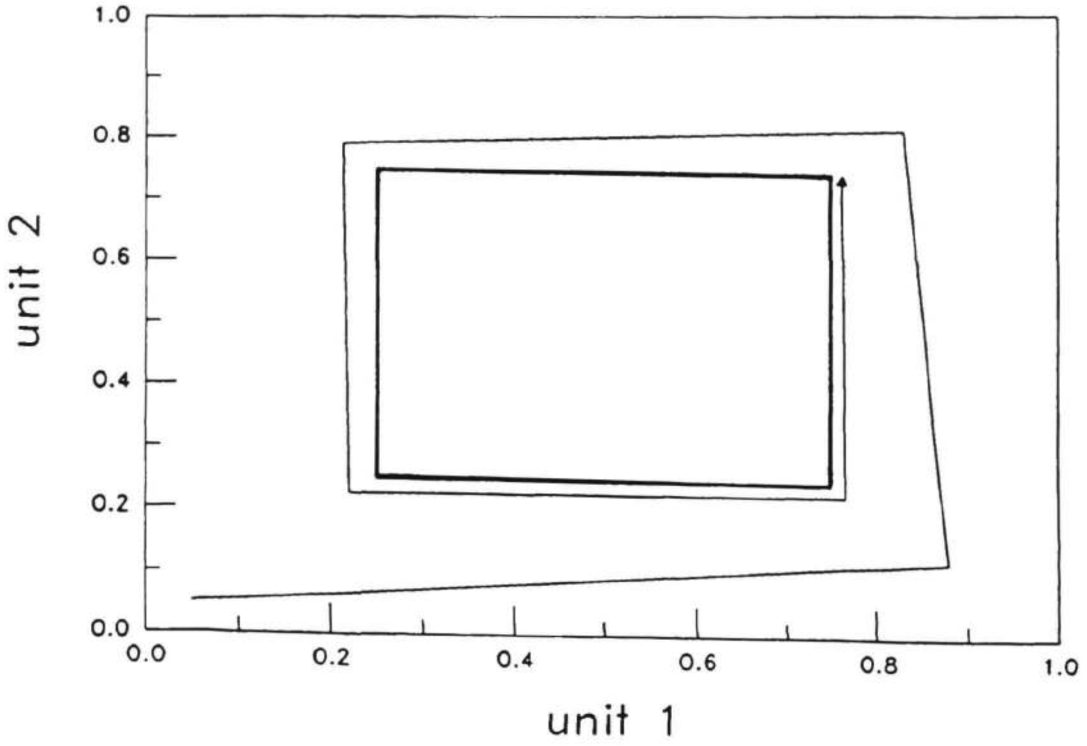

Figure 6: The activations of the two output units plotted with time as a parameter. The square is the trajectory that the network learned, and the spiral trajectory is the path that the network followed when started at the point (.05, .05).

violate the constraints.

I have elsewhere presented simulations that show that the network can mimic coarticulation data such as those presented earlier (Jordan, 1986). Several predictions were also made on the basis of these simulations. The simulations show that there can be non-adjacent interactions, so that, for example, the degree of anticipation of a feature can depend on what follows the feature. It is also the case that there is more coarticulation in the simulation over strings with homogeneous phonemic structure than over strings with heterogeneous phonemic structure.

Finally, it should be noted that it is consistent with the current approach to treat the state equations as discrete versions of a continuous process. In this case, the constraint vectors can still be applied at discrete epochs during learning. Thus, the approach would seem to have some promise for resolving some of the theoretical problems that arise at the interface between discrete phonemic representations and continuous articulatory processes

(Fowler, 1980).

DISCUSSION

One of the important problems that arises in the temporal domain is that there can be interactions both forward and backward in time. One approach to this problem is to represent actions explicitly in a spatial buffer, use relaxation techniques to allow interactions between buffer positions, and then map space into time by gating connections between actions (Feldman & Ballard, 1982). The present paper demonstrates a second approach. In the proposed network, there is no explicit representation of temporal order and no explicit representation of action sequences. There is only one set of output units for the network, therefore output vectors must arise as a dynamic process. Representing actions as distributed patterns on a common set of processing units has the virtue that partial activations can blend together in a simple way to produce the output of the system. Likewise, the representation of states as distributed patterns on a single set of units has the advantage that similarity between states has a natural functional representation in terms of the overlap of patterns. It is the similarity between nearby states that is responsible for interactions in time and this similarity has no time arrow associated with it, so that forward and backward interactions are both possible.

REFERENCES

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science, 9*, 147-169.

Barto, A. G. & Anandan, P. (1985). Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, 15*, 360-375.

Benguerel, A.- P. & Cowan, H. A. (1974). Coarticulation of upper lip protusion in French. *Phonetica, 30*, 41-55.

Booth, T. L. (1967). *Sequential machines and automata theory*. New York: Wiley.

Duda, R. O. & Hart, P. E. (1973). *Pattern classification and scene analysis.* New York: Wiley.

Feldman, J. A. & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science, 6,* 205-254.

Fowler, C. A. (1980). Coarticulation and theories of extrinsic timing. *Journal of Phonetics, 8,* 113-133.

Henke, W. L. (1966). *Dynamic articulatory model of speech production using computer simulation.* Massachusetts Institute of Technology.

Hinton, G. E. & Anderson, J. A. (Eds.) (1981). *Parallel models of associative memory.* Hillsdale, NJ: Erlbaum.

Hirsch, M. W. & Smale, S. (1974). *Differential equations, dynamical systems and linear algebra.* New York: Academic Press.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science, 79,* 2554-2558

Jordan, M. I. (1986). *Serial order: A parallel, distributed processing approach.* (Technical Report 8604). La Jolla, CA: University of California, San Diego, Institute for Cognitive Science.

Kohonen, T., Lehtio, P. & Oja, E. (1981). Storage and processing of information in distributed associative memory systems. In: G. E. Hinton and J. A. Anderson (Eds), *Parallel models of associative memory.* Hillsdale, NJ: Erlbaum.

Moll, K. L. & Daniloff, R. G. (1971). Investigation of the timing of velar movements during speech. *Journal of the Acoustical Society of America, 50,* 678-684.

Norman, D. A. (1981). A psychologist views human processing: Human errors and other phenomena suggest processing mechanisms. In: *Proceedings of the Seventh IJCAI.* Vancouver, BC.

Rosenblatt, F. (1961). *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms.* Washington, D.C.: Spartan Books.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In: D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Vol.1: Foundations.* Cambridge, MA: Bradford Books/MIT Press.

Rumelhart, D. E. & McClelland, J. L. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition, Vol.1: Foundations.* Cambridge, MA: Bradford Books/MIT Press.

Rumelhart, D. E. & Norman, D. A. (1982). Simulating a skilled typist: A study of skilled cognitive-motor performance. *Cognitive Science, 6*, 1-36.

Widrow, B. & Hoff, M. E. (1960). Adaptive switching circuits. *WESCON Convention Record Part IV*, 96-104.