SERIAL ORDER: A PARALLEL DISTRIBUTED PROCESSING APPROACH

Michael I. Jordan

May 1986

ICS Report 8604

Institute for Cognitive Science
University of California, San Diego
La Jolla, California 92093

My sincere thanks to Donald Norman and David Rumelhart for their support of many years. I also wish to acknowledge the help of Eileen Conway, Kathy Farrelly, Jonathan Grudin, Bernhard Keller, Michael Mozer, David Navon, and Stanley Parkinson.

This research was sponsored by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract No. N00014-85-K-00450, Contract Authority Identification Number, NR 667-548. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies. Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government. Requests for reprints should be sent to the Institute for Cognitive Science, C-015; University of California, San Diego; La Jolla, CA 92093.
Copyright © 1986 by Michael I. Jordan.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public reslease; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ICS 8604	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Institute for Cognitive Science University of California, San Diego 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Personnel & Training Research Programs Office of Naval Research (Code 1142PT)
6c. ADDRESS (City, State, and ZIP Code) C-015 La Jolla, CA 92093	7b. ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85-K-0450
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 61153N; PROJECT NO. RR04206; TASK NO. RR04206-0A; WORK UNIT ACCESSION NO. NR 667-548

11. TITLE (Include Security Classification)

Serial Order: A Parallel Distributed Processing Approach

12. PERSONAL AUTHOR(S)
Michael I. Jordan

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM June 85 TO Mar 86	May 1986	40

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Serial order; parallel distributed processing; connectionist networks; dual tasks; speech production
05	10		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A theory of serial order is proposed which attempts to deal both with the classical problem of the temporal organization of internally generated action sequences as well as with certain of the parallel aspects of sequential behavior. The theory describes a dynamical system which is embodied as a "parallel distributed processing" or "connectionist" network. The trajectories of this dynamical system come to follow desired paths corresponding to particular action sequences as a result of a learning process during which constraints are imposed on the system. These constraints enforce sequentiality where necessary, and as they are relaxed, performance becomes more parallel. The theory is applied to the problem of coarticulation in speech production and simulation experiments are presented.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Harold Hawkins	22b. TELEPHONE (Include Area Code) (202) 696-4323 22c. OFFICE SYMBOL ONR 1142PT

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 2

Contents

SERIAL ORDER 2
Coarticulation 4
PARALLEL DISTRIBUTED PROCESSING 4
Recurrent and Nonrecurrent Networks 5
Distributed Representations 6
Nonlinearities 6
Learning 7
A THEORY OF SERIAL ORDER 8
Learning and Parallelism 11
Attractor Dynamics 13
Serial Order 14
Examples and Simulations 16
Plan Representations 19
Dynamical Properties of the Networks 20
Learning With Don't-Care Conditions 22
COARTICULATION 23
Simulation Experiments 24
Discussion 27
DUAL-TASK PARALLELISM 29
Simulation Experiments 31
Experiment 1 31
Experiment 2 33
Experiment 3 33
Discussion 34
OTHER ISSUES 36
Rate 36
Errors 36
Hierarchies 36
CONCLUSIONS 37
REFERENCES 37

Serial Order: A Parallel Distributed Processing Approach

MICHAEL I. JORDAN

Even the most cursory examination of human behavior reveals a variety of serially ordered action sequences. Our limb movements, our speech, and even our internal train of thought appear to involve sequences of events that follow one another in time. We are capable of performing an enormous number of sequences, and we can perform the same actions in a variety of different contexts and orderings. Furthermore, most of the sequences that we can perform were learned through experience.

A theory of serial order in behavior should clearly be able to account for these basic data. However, no such general theory has emerged, and an important reason for this is the failure of current formalisms to deal adequately with the parallel aspects of serially ordered behavior. We can tentatively distinguish two forms of parallelism. The first is parallelism that arises when actions in a sequence overlap in their execution. In speech research, such parallelism is referred to as *coarticulation* (Kent & Minifie, 1977; Moll & Daniloff, 1971; Ohman, 1966), and it greatly complicates the traditional description of sequential speech processes. The second form of parallelism occurs when two actions are required to be performed in parallel by the demands of the task or by implicit constraints. Such is the case, for example, in the *dual-task* paradigm, in which actions that have been learned separately must be performed together. This differs from the case of coarticulation, in which actions that are nominally separate in time are allowed to be performed in parallel. It is important to characterize both how such parallelism can arise within a sequential process and how it can be constrained so that unwanted parallel interactions are avoided.

In this paper, I present a theory of serial order which describes how sequences of actions might be learned and performed. In the theory, parallel interactions across time (coarticulation) and parallel interactions across space (dual-task interference) are viewed as two aspects of a common underlying process. Briefly, both are seen as manifestations of a tendency of the output system to generalize learned relationships between internal state vectors and output vectors representing actions. These generalizations are constrained during the learning process so that inappropriate generalizations that cause interference are overridden with practice.

Although the emphasis in this paper is on the production of sequences of actions, it should be noted that the system to be described has a natural interpretation as a *dynamical system*, and a continuous-time perspective is possible. In this case, the sequences of actions define trajectories in a state space. It will be shown that the learning of particular trajectories can generalize to nearby regions of the space and that interesting attractors such as limit-cycles can arise. The approach would therefore seem to have some potential for reconciling problems of serial order with problems relating to the continuous nature of behavior.

The theory is embodied in the form of a parallel distributed processing (Rumelhart & McClelland, 1986) or connectionist (Feldman & Ballard, 1982) network. Such networks are composed of a large number of simple processing units that are connected through weighted links. In various forms, such networks have been used as models of phenomena such as stereopsis (Marr & Poggio, 1976), word recognition (McClelland & Rumelhart, 1981), and reaching (Hinton, 1984). The success of these models has been in large part due to their high degree of parallelism, their ability to bring multiple,

interacting constraints to bear in solving complex problems, and their use of distributed representations. However, none of these properties seem particularly well suited to the problem of serial order, and indeed a criticism of this class of models has been their inability to show interesting sequential behavior, whereas the more traditional symbolic approaches, typically by assuming a sequential processor as a primitive, deal with serial order in a much more straightforward manner. This criticism is challenged in this paper, in the context of a theory of serial order that has been developed to take advantage of the underlying primitives provided by parallel distributed processing.

SERIAL ORDER

Many of the problems encountered in developing a parallel distributed processing approach to the serial order problem were anticipated by Lashley (1951). Lashley pointed out the insufficiency of the *associative chaining* solution to the serial order problem. The associative chaining solution assumes that serial ordering is encoded by directed links between control elements representing the actions to be ordered, and that the performance of a sequence involves following a path through the network of control elements. Lashley argued that this solution fails to allow different orderings of the same actions because there is no mechanism for specifying which link should be followed from an element having more than one outgoing link. He also argued that serial behavior shows anticipatory effects of future actions upon the current action, and that such context effects are not accounted for within the associationist framework.

Lashley's arguments have had an impact on those seeking to understand the role of feedback in a theory of motor behavior, but have been less influential on those interested in the structure of *motor programs.*[1] This is in all likelihood due to the impact on theorists of the development of the digital computer, which made it possible to see how arbitrary sequential programs can be executed. Theories based explicitly on the computer metaphor have invoked the notion of a *buffer* which is loaded with the actions to be performed, and a program counter which steps through the buffer (Shaffer, 1976; Sternberg, Monsell, Knoll, & Wright, 1978). Despite the generality of such a theory, simple buffer theories are known to have several problems, including accounting for error patterns (Kent & Minifie, 1977; MacKay, 1981). It is also true that coarticulation is not well handled by buffer theories. One approach is to assume that buffer positions can interact with each other (Henke, 1966). However, this interaction, which must take place when successive actions are simultaneously present in the buffer, takes time, as does the process of reloading the buffer once a set of related actions have been executed, and implies the presence of delays at certain times in the production of long sequences. Such delays are not observed in fluent sequential behavior (cf. Shaffer & Hardwick, 1970). Another problem is that the interactions between actions should depend on their relative positions in the buffer, not their absolute positions. For example, the interactions between /i/ and /n/ should presumably be the same when saying *print* and *sprint*. This would seem to imply the need for a complex mechanism whereby learned interactions can automatically generalize to all buffer positions. Such issues, which arise due to the explicit spatial representation of order in buffer theories, seem to be better handled within an associationist framework.

Wickelgren (1969) revived the associationist approach by assuming that serial order is indeed encoded by directed links between control elements, but that the control elements are different for different orderings of the same actions. The control element for the action B in the sequence ABC can be represented by the form ${}_{A}B_{C}$ whereas the control element for B in the sequence CBA is represented as ${}_{C}B_{A}$. These control elements are distinct elements in the network, thus there is no problem with representing both the sequences ABC and CBA in the same network. In this account, actions look

[1] That is, control structures which can in principle produce sequences of actions without feedback from the periphery. Lashley also argued against associative chaining because of the relatively long response time of peripheral feedback. Note, however, that the arguments stated above hold for any associative chaining theory, whether or not the links go through the periphery.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 5

different in different contexts not because of parallel execution but because they are produced by different control elements.

Wickelgren's theory provides a solution to the problems posed by Lashley but it has several shortcomings. First, it requires a large number of elements, yet has difficulty with the pronunciation of words, such as *barnyard*, that have repeated subsequences of length two or more (Wickelgren, 1969). Second, effects of context in speech have been shown to extend up to four or five phonemes forward in an utterance (Benguerel & Cowan, 1974). Extension of the theory to account for such effects would require an impossibly large number of control elements. Finally, note that there are only representations for *tokens* in the theory, and no representations for *types*. There is nothing in the theory to tie together the contextual variations of a given action. This means that there is no way to account for the linguistic and phonetic regularities that are observed when similar actions occur in similar contexts (Halwes & Jenkins, 1971).

A different approach is to assume that actions are to some extent produced in parallel (Fowler, 1980; Rumelhart & Norman, 1982). The parallelism allows several control elements to influence behavior at a particular point in time, and therefore provides an account of coarticulatory effects, even though actions are represented in terms of context-free types. Rumelhart and Norman have shown that a model of typing incorporating parallelism can produce overlapping keystrokes much like those observed in transcription typing.

Allowing parallel activation of control elements accounts for context sensitivity; however, there remains the problem of temporal ordering. Rumelhart and Norman achieved temporal ordering by assuming that elements suppress other elements through lateral inhibitory connections if they precede those elements in the sequence. This particular scheme is susceptible to Lashley's critique because all possible inhibitory connections must be present to allow the performance of the same elements in different orders and a mechanism is needed for selecting the particular inhibitory connections used in the performance of a particular sequence. However, there are other ways of achieving the same effect that are not open to Lashley's critique (Grossberg, 1978; Grudin, 1981). Essentially, all of these schemes produce temporal order by inducing a graded activation pattern across the elements in the sequence to be performed, such that elements more distant in the future are activated less than earlier elements. Elements are assumed to influence behavior in proportion to their level of activation. Because the next action in the sequence is the most highly activated, it has the most influence on behavior. Once the activation of an element reaches a threshold, it is inhibited, allowing the performance of other items in the sequence.

A problem with these parallel activation theories is that they have difficulty with sequences in which there are repeated occurrences of actions. In a pure type representation, there is simply no way to represent the repeated action. Rumelhart and Norman used a modified type representation in which they introduced special operators for doublings (e.g., AA) and alternations (e.g., ABA). However, they provided no general mechanism, and, for sequences such as ABCA, invoked a parser to break up the sequence into pieces, thus allowing no parallel influences across the break. This is not a satisfactory solution, in general, because data in speech show that coarticulatory influences can extend across sequences like ABCA (Benguerel & Cowan, 1974). Another possibility is to assume that repeated occurrences of actions are represented by separate control elements (representation by tokens). However, the combined effects of partially activated control elements will cause the first occurrence of a repeated action to move forward in time, whether or not this is actually desirable. Indeed, in a sequence such as ABBB, the B may overwhelm the A and be executed first. These problems are enhanced in featural representations of the kind that are often posited for actions (Grudin, 1983; Perkell, 1980; Rosenbaum, 1980) because the total activation from elements representing the repeated features will be greater than the activation levels for features that only occur once in the sequence, irrespective of the order of the features. Such problems arise because the single quantity of activation is being used to represent two distinct things: the parallel influences of actions and the temporal order of actions.

It is my view that many of these problems disappear when a clear distinction is made between the *state* of the system and the *output* of the system. Explicitly distinguishing between the state and the output means that the system has two activation vectors, which allows both temporal order and parallel

influences to be represented in terms of activation. In the theory developed in this paper, the state and the output are assumed to be represented as patterns of activation on separate sets of processing units. These sets of units are linked by connections defining an *output function* for the system. Serial order is encoded both in the output function and in recurrent connections impinging on units representing the state; there is no attempt to encode order information in direct connections between the output units.

Coarticulation

In this section, I briefly introduce some of the parallel aspects of sequential behavior that have been considered important in the development of the current theory. I will focus on the parallelism between actions in a sequence and will return to the problem of dual-task parallelism in a later section.

Several studies involving the recording of articulator trajectories have shown that speech gestures associated with distinct phonemes can occur in parallel. 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.[2] 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*/. They showed that lip-rounding for the /*y*/ can begin as early as the first /*s*/. This is presumably allowable because the articulation of the consonants does not involve the lips.

These examples suggest that the speech system is able to take advantage of free articulators and use them in anticipating future actions. This results in parallel performance and allows speech to proceed faster and more smoothly than would otherwise be possible. Such parallelism clearly must be constrained by the abilities of the articulators. However, there are other constraints involved as well. 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 are different phonemes than non-nasal vowels, the velum would not be allowed to coarticulate with the vowels. Thus the articulatory control system cannot blindly anticipate articulations, but must be sensitive to phonemic distinctions in the language being spoken by only allowing certain coarticulations.

The situation is more complicated still, if we note that constraints on parallelism may be specific to particular features. For example, in the case of /*strstry*/, only the *rounding* of the /*y*/ can be anticipated. The *voicing* of the /*y*/, which also involves an articulator that is not used by the consonants, cannot be anticipated, because that would change the identities of the consonants (for example, the /*s*/ would become a /*z*/). Again, such knowledge cannot come from consideration of strategies of articulation, but must reflect higher-level phonemic constraints.

Thus, speech presents a difficult distributed 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 show in the remainder of this paper is how this problem can be approached with a theory based on parallel distributed processing networks.

PARALLEL DISTRIBUTED PROCESSING

Before discussing the theory, I will provide a short discussion of those aspects of parallel distributed processing needed for the remainder of this paper. An extensive discussion of this class of models can be found in Rumelhart and McClelland (1986).

A parallel distributed processing network is a network of processing units, connected by weighted, unidirectional links. The state of each processing unit, at each moment of time, can be described by a

[2] This is an example of *forward coarticulation*. It is also possible to see perseveration, which is referred to as *backward coarticulation*.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 7

single real number, its *activation*. Units compute their activations in parallel, according to the following equation:

$$x_j = \phi\left(\sum_{i=1}^{n} w_{ji}\, x_i + \theta_j\right), \tag{1}$$

where x_i is the activation of the ith unit, w_{ji} is the weight from the ith unit to the jth unit, θ_j is a bias associated with the jth unit, and n is the number of units in the network. The quantity inside the parentheses is referred to as the *net input* to a unit. The net input is modified by a squashing function ϕ, which is typically either a logistic function or a thresholding function. For units in the simulated networks described in this paper, ϕ is either the identity function

$$\phi(x) = x,$$

in which case the output of a unit is simply a linear function of its inputs, or the logistic function

$$\phi(x) = \frac{max - min}{1+e^{-x}} + min,$$

where *min* is the minimum value attained by the logistic function, and therefore the minimum value of activation that any unit can have, and *max* is the maximum value. The logistic function is an S-shaped function with asymptotes at *min* and *max*.

It is useful to give a geometric interpretation to the state of an entire network. The activations of the processing units form an n-dimensional vector, which is a point in a state space. Over time, as units update their activations, this point moves, tracing out a trajectory in the state space. The particular trajectory that arises depends in general on the connection pattern of the network, the weights, and the initial state of the network.

Recurrent and Nonrecurrent Networks

An important distinction can be made between networks based on their overall connectivity. If a network has one or more cycles, that is, if it is possible to follow a path from a unit back to itself, then the network is referred to as *recurrent*. A *nonrecurrent* network has no cycles.

Nonrecurrent networks can be thought of as computing an input-output function. If we treat some of the units in the network as *input* units, and other units as *output* units, then we can speak of a functional relationship between the inputs and the outputs. That is, the activations of the input units form a vector **x**, the activations of the output units form a vector **y**, and there is a function relating **x** and **y**.

A simple example of a recurrent network is given in Figure 1. The recurrent connection of the output unit back on itself means that the output of the network depends not only on the input, but also on the state of the network at the previous time step. For example, letting μ be the value of the recurrent weight, and assuming for simplicity that the units are linear (i.e., the function ϕ is the identity function), the activation of the output unit at time t is given by

$$\begin{aligned} x_2(t) &= \mu x_2(t-1) + w_{21} x_1(t) \\ &= \mu^t x_2(0) + \sum_{\tau=0}^{t-1} \mu^\tau w_{21} x_1(t-\tau), \end{aligned}$$

where $x_1(t)$ is assumed to be constant over time. This equation shows that the trajectory of the network exponentially approaches a constant state for μ less than one, and goes to infinity for larger values of μ.

Another simple example of recurrence is a network of three units connected in a ring with positive weights on the links. If the initial state has one unit with positive activation, and the other two units

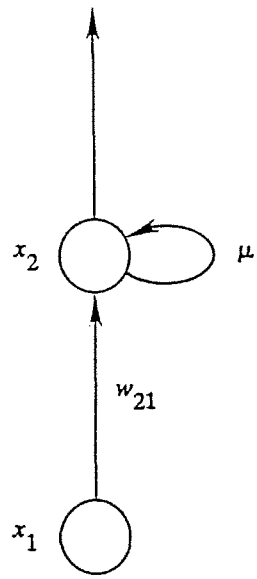

FIGURE 1. An example recurrent network with two units.

with activation zero, then the network will tend to pass the positive activation around the ring. Such behavior is characterized by a *cycle* in the state space.

If the input to a nonrecurrent network is held constant, the trajectory of the network in state space will remain at a single point. Clearly, in order to achieve interesting sequential behavior in the presence of a constant input, there must be recurrent connections in the network.

Distributed Representations

In order to use parallel distributed processing networks to model behavior, decisions must be made as to how patterns of activation in the network are to represent psychological entities. A *local* representational scheme assumes that each unit in the network represents an entity in the theory (Feldman & Ballard, 1982). Most theories of serial order either implicitly or explicitly assume a local representation of actions in which an action is represented by a single unit or control element, and activation of that unit causes the action to be executed.

In a *distributed* representational scheme, entities are represented not by single units, but rather by patterns of activation across a set of units (Hinton & Anderson, 1981). Different patterns of activation across the same units correspond to different entities. By virtue of this overlap in the representation of different objects, distributed representations can provide a natural account of phenomena of generalization and similarity. This is due to the linear term in the activation functions; it is a general property of these networks that similar inputs produce similar outputs. Distributed representations also allow an economy of units: if there are n processing units, a local representational scheme can only represent n objects, whereas a distributed scheme can potentially represent many more if the mechanisms that operate on these representations are able to make the necessary distinctions between patterns.

Nonlinearities

Nonlinear mappings can be implemented in a network by interposing a set of intermediate units having nonlinear activation functions (the function ϕ in Equation 1) between the input units and the output units (Hinton & Sejnowski, 1983; Minsky & Papert, 1969; Poggio, 1975; Rosenblatt, 1962; Rumelhart,

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 9

Hinton, & Williams, 1985). These intermediate units will be referred to as *hidden* units, following the terminology of Hinton and Sejnowski (1983). The role of hidden units can be clarified through a consideration of the geometry involved. Consider a unit with a threshold activation function. That is, let

$$x_i = \begin{cases} 1 & \text{if } \sum_j w_{ij} x_j \geq \theta \\ 0 & \text{otherwise.} \end{cases}$$

Such a unit can be thought of as dividing the space of its input vectors into two regions: one region where its net input is less than the threshold θ and one region where its net input is greater than the threshold. The border separating the regions is a hyperplane, due to the linear way in which the inputs to the unit are combined. The input vectors on the same side of the hyperplane are mapped to the same value. There are some functions, such as the logical functions AND and OR, that can be computed by dividing the space into two regions in this way. Such functions are referred to as *linearly separable*. However, there are a very large number of functions, including the logical function XOR, which cannot be computed in this way. Hidden units must be used to compute such functions. If each hidden unit divides the space into two regions, then a set of hidden units can partition the space into simplices. Each simplex can be mapped to a different value by the output units.

A logistic function can be thought of as a continuous version of a threshold. The use of a logistic function instead of a threshold gives some more flexibility by allowing real-valued outputs. However, the problem of nonlinearly separable functions remains, and hidden units must be used to implement such functions.

Learning

Parallel distributed processing networks learn by changing the weights on the links between units so that the network can achieve certain criterion behaviors. In networks with no hidden units, learning rules have been developed that allow any linearly separable function to be learned (Kohonen, 1977; Rosenblatt, 1962; Widrow & Hoff, 1960). Only recently have algorithms been developed that allow learning to occur in networks with hidden units (Ackley, Hinton, & Sejnowski, 1985; Barto & Anandan, 1985; Rumelhart, Hinton, & Williams, 1986).

The algorithm developed by Rumelhart, Hinton, and Williams (1986) is used in the networks discussed in this paper. This algorithm is an error-correcting scheme in which errors generated at the output units are propagated into the network to allow hidden units to change their weights. Let us assume that the network is a nonrecurrent network and that the task of the network is to produce a particular vector on the output units when a particular vector is present on the input units. The basic idea of the algorithm is the following. It is assumed that there is a teacher which provides the desired activation values of the output units. An error signal is generated at each output unit by comparing the desired output with the actual output. The weights on the links coming in to the output units are then changed by an amount proportional to the error signal. Error signals are then propagated back down these links to the hidden units and error signals for the hidden units are computed by adding the propagated signals. Essentially, the network assigns blame to units that lead to a large error in the output vector. Units with larger blame change their weights more in order to correct the error. Rumelhart, Hinton, and Williams (1986) have shown that this algorithm changes a given weight by an amount proportional to the partial derivative with respect to the weight of the sum of squared error at the output units. Thus, the algorithm is a gradient search in weight space for a set of weights that implements the desired function.

The algorithm can also be applied to recurrent networks. However, in recurrent networks, it is necessary for each unit to keep a history of its activations at prior time steps in order for the error-propagation process to work properly. This seems an excessive requirement for networks in biological systems and prevents a direct application of the algorithm to the problem of temporal performance. By

restricting the form of the recurrent connections in the network, however, it is possible to find biologically plausible architectures that implement the current theory and do not require histories of activations to be stored.

A THEORY OF SERIAL ORDER

Let there be some sequence of *actions* $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_r$, which are to be produced in order in the presence of a *plan* $\mathbf{p}$. Each action is a vector in a parameter or feature 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.

In general, we would like the system to be able to produce many different sequences. Thus, different vectors $\mathbf{p}$ are assumed to be associated with different sequences of actions. A particular sequence is produced when a particular vector $\mathbf{p}$ is presented as input to the system. Note that, in principle, there need be no relationship between the form of plan vectors and the sequences that they evoke. Rather, a plan vector evokes a particular sequence because it was present as input to the system when the sequence was learned. Thus, plans can simply be arbitrary patterns of activation which serve to key particular sequences; they are not scripts for the system to follow.

Actions are produced in a temporal context composed of actions nearby in time. This context entirely determines the desired action, in the sense that knowing the context makes it possible to specify what the current action should be. It is proposed that the system explicitly represents the temporal context of actions in the form of a state vector and chooses the current action by evaluating a function from states to actions. At each moment in time, an action is chosen based on the current state, and the state is then updated to allow the next action to be chosen. Serial order does not arise from direct connections between units representing the actions; rather, it arises from two functions that are evaluated at each time step: a function f which determines the output action $\mathbf{x}_n$ at time n,

$$\mathbf{x}_n = f(\mathbf{s}_n, \mathbf{p}) \tag{2}$$

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

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

where both functions depend on the constant plan vector as well as the current state vector. Following the terminology of automata theory (Booth, 1967), f will be referred to as the *output* function, and g will be referred to as the *next-state* function.[3]

Assumptions are made in the theory about the form of these functions. The output function f is assumed to arise through the learning of associations from state and plan vectors to output vectors. These learned associations are assumed to generalize so that similar states and plans tend to lead to similar outputs. The major requirement for the next-state function g is that it have a continuity property: State vectors at nearby points in time are assumed to be similar. This requirement makes sense if the state is thought of as representing the temporal context of actions; intuitively, it seems appropriate that the temporal context should evolve continuously in time. Note that if the continuity property holds, then the generalizations made by the output function are such as to spread actions in time, and that as learning proceeds there is a tendency towards the increasing parallel execution of nearby actions. This process will be discussed below in detail, where it will also be shown how the generalizations leading to parallelism can be constrained.

[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.

The basic network architecture is shown in Figure 2. The entities of the theory — plans, states, and outputs — are all assumed to be represented as distributed patterns of activation on three separate pools of processing units. The plan units and the state units together serve as the input units for a network which implements the output function f 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 full network is shown in Figure 3, and the connection scheme is summarized in Table 1.[4]

In the proposed network, there is no explicit representation of temporal order and no explicit representation of action sequences. This is due to the fact that there is only one set of output units for the network, so that at any point in time, only one output vector is present. Output vectors must arise as a dynamic process, rather than being prepared in advance in a static buffer and then serially executed. 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. The proposed network essentially implements the output function which relates these patterns as a network associative memory in which many associations are stored in the same set of weights. The learning and generalization abilities demonstrated for such networks (Hinton & Anderson, 1981; Rumelhart & McClelland, 1986) are just those that are needed for the output function in the current theory.

Although it is possible that the next-state function as well as the output function arises through learning, this is not necessary for the system as a whole to be able to learn to produce sequences. Furthermore, given that the next-state function is set up in such a way that the continuity property holds, little is lost in the current framework if the recurrent connections implementing the next-state function are taken as fixed and if only the output function is learned. This is the approach taken in the remainder of this paper. One choice of values for the fixed recurrent connections is based on the conception of the state as a temporal context. Consider the case of a sequence with a repeated subsequence or a pair of

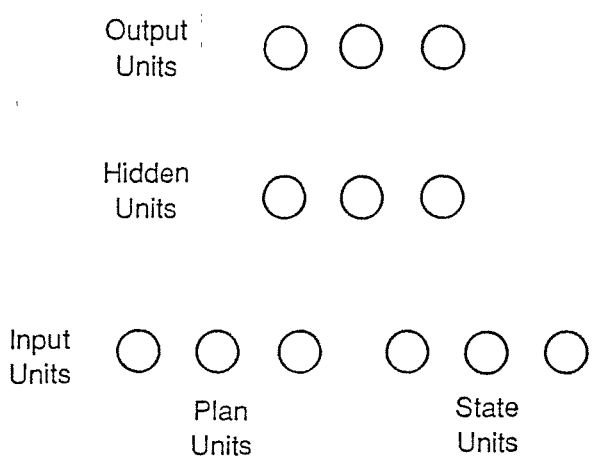

FIGURE 2. The processing units in the network. The plan and state units together constitute the input units for the network.

[4] In a later section, I will also suggest that the plan units should be interconnected with symmetric connections.

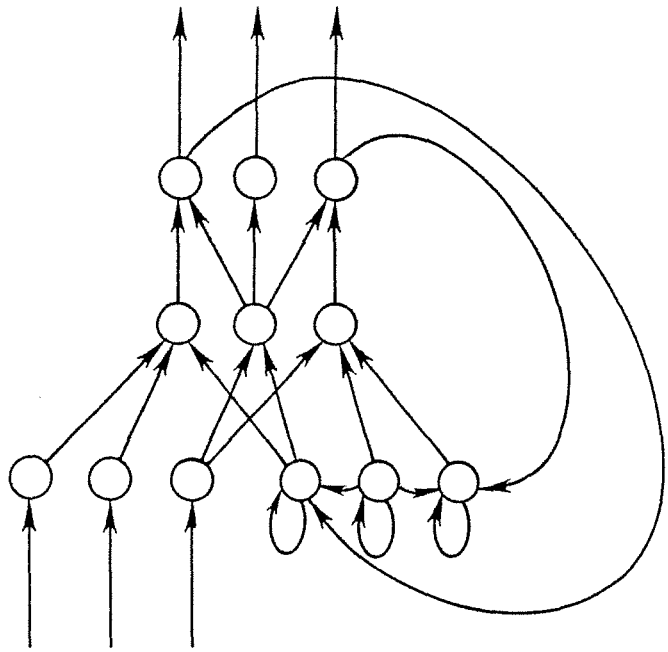

FIGURE 3. Basic network connection scheme (not all connections are shown).

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. This suggests defining 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 is achieved if

TABLE 1

SUMMARY OF CONNECTION PATTERNS IN THE NETWORK

	Plan Units	State Units	Hidden Units	Output Units
Plan Units			×	×
State Units		×	×	×
Hidden Units				×
Output Units		×		

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.[5] In this case, the state at time n is given by

$$\mathbf{s}_n = \mu\, \mathbf{s}_{n-1} + \mathbf{x}_{n-1}. \tag{2}$$

$$= \sum_{\tau=1}^{n-1} \mu^{\tau-1}\, \mathbf{x}_{n-\tau}.$$

The similarity between states 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). It is worth mentioning that the recurrent connections from the output units to the state units are not necessary for the operation of the network, given that there are connections between the state units. Consider, for example, two state units with antisymmetric connections. That is, let the weight on the link from one state unit to another be the negative of the weight on the inverse link. Furthermore, let these units have logistic activation functions with asymptotes at -1 and 1. These units act as an oscillator, and the trajectory of activations of the units follows a circle in the plane. Finally, if the units also have connections onto themselves, then the trajectories will be continuous, as required of the next-state function. Note that with this representation, state trajectories do not differ between sequences. This does not cause problems in learning to perform different sequences, however, because the plan vector serves to distinguish between sequences. Both this representation of the state and the exponential trace representation have been used in simulations of the network; however, only the results from the exponential trace representation will be reported in this paper.

Although I have presented the theory using discrete-time state equations, the continuous case does not present substantial changes. In the continuous case, the state equations become a system of differential equations defining a continous time dynamical system. The network architecture is the same as before, as are the assumptions made of the functions f and g.[6] Associations are learned between state and plan vectors to desired output vectors at particular epochs, essentially constituting points in the space of activations of the output units through which trajectories must pass. As to be discussed in the next section, I will modify the form of desired output vectors so that they specify regions in the output space, rather than points. Transitions between these regions tend to be smooth due to the underlying continuity of the state.

Learning and Parallelism

Learning is assumed to occur throughout the sequence being learned (rather than only at the end of the sequence). In the network, learning is realized as an error-correcting process in which parameters of the network are incrementally adjusted based on the difference between the actual output of the network and a desired output. Essentially, the next-state function provides a time-varying state vector, and associations are learned from this state vector and the plan vector to desired output vectors.

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, 1962; 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.

[5] 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.

[6] For the continuous equations to go through, it is imperative that each state unit have a connection to itself. This requirement is met by both of the state representations discussed above.

Constraints may specify a range of values that 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 that are defined among output units; for example, the sum of the activations of a set of units might be required to take on 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 changing system parameters towards a configuration in which the constraint is met. An example of this process is shown in Figure 4, 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 cases where more than one constraint is imposed on an output unit, the error is just the sum of the errors from the separate constraints.

Consider first the case in which desired output vectors specify values for only a single output unit. Suppose that a network with three output units is learning the sequence

$$\begin{bmatrix} .9 \\ * \\ * \end{bmatrix}, \begin{bmatrix} * \\ .9 \\ * \end{bmatrix}, \begin{bmatrix} * \\ * \\ .9 \end{bmatrix}.$$

The network is essentially being instructed to activate its output units in a particular order, and this case can be thought of as involving local representations for actions. At each time step, errors are propagated from only a single output unit, so that activation of that unit becomes associated to the current

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

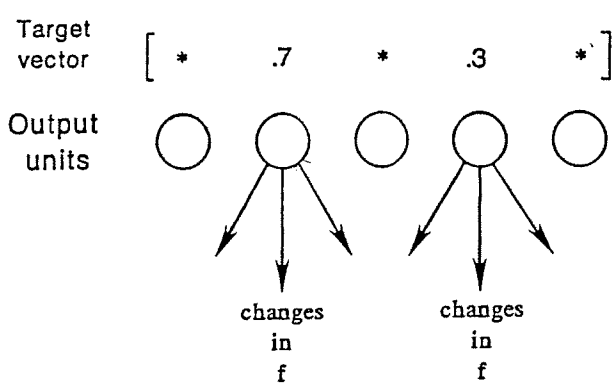

FIGURE 4. The network learns by propagating errors from output units where constraints are imposed on the activations.

state.[7] Associations are learned from s_1 to activation of the first output unit, from s_2 to activation of the second output unit, and from s_3 to activation of the third output unit. These associations also generalize so that, for example, s_1 tends to produce partial activations of the the second and third output units. This occurs because s_1 is similar to s_2 and s_3, by the assumption of continuity of the next-state function, and similar inputs produce similar outputs in these networks. After learning, the network will likely produce a sequence such as

$$\begin{bmatrix} .9 \\ .7 \\ .5 \end{bmatrix}, \begin{bmatrix} .7 \\ .9 \\ .7 \end{bmatrix}, \begin{bmatrix} .5 \\ .7 \\ .9 \end{bmatrix},$$

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

$$\begin{bmatrix} .9 \\ 0 \\ * \end{bmatrix}, \begin{bmatrix} * \\ .9 \\ * \end{bmatrix}, \begin{bmatrix} * \\ * \\ .9 \end{bmatrix}.$$

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

$$\begin{bmatrix} .9 \\ 0 \\ .5 \end{bmatrix}, \begin{bmatrix} .7 \\ .9 \\ .7 \end{bmatrix}, \begin{bmatrix} .5 \\ .6 \\ .9 \end{bmatrix},$$

where the added constraint is now met. In this example, the network must block the generalization that is made from from s_2 to 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 nonoverlapping sets of output units. Note that there is no need to invoke a special process to program in the parallelism; it arises from the ability of the system to generalize and is a manifestation of the normal functioning of the system. Indeed, 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. These observations are summarized in Figure 5, 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

[7] I am ignoring the plan here to simplify the exposition.

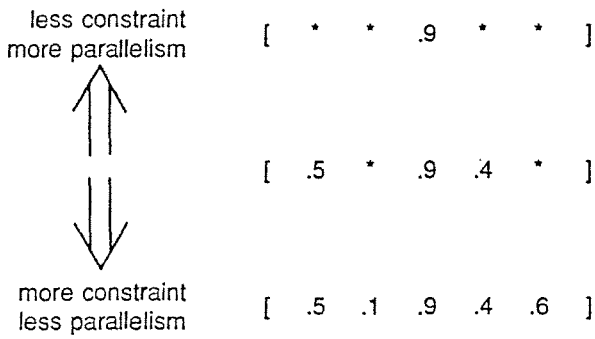

FIGURE 5. Relationships between constraint and parallelism.

sequence. However, a stronger property appears to hold: The learned sequences become attractors for nearby regions of the state space and perturbed trajectories return to the learned trajectories. For example, if the network learns a sequence that corresponds to a cycle in the state space, then the cycle becomes a *limit cycle* for the system. This behavior will be demonstrated in the next section. To have limit cycles, a dynamical system must be nonlinear (Hirsch & Smale, 1974), which further demonstrates the need for a nonlinear output function in the network.

More globally, a network that has learned to produce several different cyclical sequences may have several regions of the state space that 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 that 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.

Serial Order

Before turning to a discussion of simulations of the network, it is worth considering how the current theory fares with respect to some of the general requirements of a theory of serial order. It should be clear that the theory can account for the production of abstract sequences, given that the state changes over time, and given that an appropriate output function can be constructed. Different orderings of the same actions can be achieved, both because the state trajectories may differ between the sequences and because the output function depends on the plan, and the plan can distinguish the different orderings. The theory has no problem with repeated actions; the existence of repeated actions simply indicates that the output function is not one-to-one, but that two or more state, plan pairs can map to the same output vector. Finally, sequences such as ABAC, which cause problems for an associative chaining theory because of the transitions to distinct actions after a repeated action, are possible because the state after the first A is not the same as the state after the second A.

The theory is able in principle to account for a variety of regularities that occur within and between sequences. This is due to the fact that outputs and states are represented as types—that is, there is only one set of output units and one set of state units. The same weights underlie the activation of actions, in whatever position in the sequence, and in whatever sequence. Thus, particular weights can underlie the regularities observed for similar actions in similar contexts. This mechanism can potentially extend to higher level regularities, such as the phonological regularities of speech production. For example, the fact that voiceless stops lose their aspiration following an initial /s/ (e.g., /spIn/ is pronounced

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 17

[sbIn]) could be encoded by inhibitory connections from state units encoding the recent occurrence of a voiceless fricative to output units controlling the degree of aspiration. The ability of distributed networks to capture linguistic regularities in this way has been discussed by Rumelhart and McClelland (1986) in the domain of verb morphology, and Sejnowski and Rosenberg (1986) in the domain of grapheme to phoneme translation.

One of the more important tests of a theory of serial order is that it account for interactions both forward and backward in time. In the current theory, time is represented implicitly by the configuration of the state vector. Interactions in time are due to the similarity of the state vector at nearby points in time. There is no time arrow associated with this similarity, thus, forward and backward interactions are equally possible.

Limitations on the structure of the functions f and g will lead to some sequences being more difficult to learn and perform than others. For example, the temporal context cannot extend indefinitely far in time; thus, the repetition of lengthy subsequences that make transitions to different actions can be difficult to learn and perform. Also, similarity between action transitions in different plans can cause interference, as can similarity between plan representations. The interference can lead to errors and to the learning of one sequence causing negative transfer on another sequence. Interference can also have a positive side, of course, in the form of positive transfer.

An important issue not directly addressed by the theory concerns the information necessary for the learning process. The theory assumes that learning occurs during the course of a sequence, rather than only at the end of the sequence. This implies the existence of another sequential process, which can be referred to as a *teacher*, that makes available the constraint vectors needed for learning. In other words, the theory posits that learning takes place by deriving one sequential process from another. This would seem to lead to an infinite regress. However, this need not be the case. Constraint vectors may arise through analysis or planning based on sensory information from long loops through the environment. The theory provides a way for the system to eventually liberate itself from the environment and to produce sequences without the need for waiting for and analyzing sensory information. Another possibility is that the teacher process may depend on simpler serial ordering mechanisms than are needed for the general theory. For example, consider learning to pronounce a word that one encounters for the first time. If the word is written, we may scan the word slowly, relying on the properties of foveal vision to activate perceptual representations of the letters in order. These representations are already temporally ordered, and can be converted directly into a sequence of phonemic representations to be learned. Similar considerations apply when the words are presented auditorally, in which case the decay of representations in a low-level auditory buffer can be used to induce temporal order. These arguments suggest that mechanisms that are more properly part of a theory of attention or of short-term memory are relevant to specifying how the basic sequential processes needed for learning arise. It should also be noted that there are several potential gains to be realized in passing from one sequential process to another. For example, the teacher process can produce non-context-sensitive actions, and the learning process will integrate the actions, producing the sequence in a more rapid and fluent manner. Also, the teacher process may decide on the actions through a computationally intense planning process. The learning process produces the actions without the need for the planning. In this sense, the learning system can be likened to a compiler. Finally, there also remains the problem of specifying how the components of the constraint vectors themselves arise through learning. One approach involves the learning of an inverse kinematic or dynamic mapping, which can then be used to generate motoric error vectors from spatial errors (Rumelhart, personal communication, 1985). Another approach is to suppose that constraint vectors are not explicitly present during learning, but that the learning process acts as if it were using the constraints. For example, the learning process may be using a simple evaluation of performance (cf. Barto, Sutton, & Anderson, 1983), and the construction of the evaluation signal may obey the constraints. In the current paper, I have assumed that these problems, involving learning of how to produce basic actions, can be usefully separated from problems that involve learning to produce sequences of actions.

Examples and Simulations

The operation of the network can be elucidated with the example of a network designed to perform the sequence AAAB, where A is the vector $\begin{bmatrix}1\\1\end{bmatrix}$ and B is the vector $\begin{bmatrix}0\\0\end{bmatrix}$. Consider the network shown in Figure 6 with two output units, two state units, and one hidden unit. In this example, there is no need for a plan because only one sequence is being learned by the network. The weights are shown near the links between the units, and the biases are shown in the circles representing the units. Assume that the hidden unit and the output units are binary threshold units that have an output of 1 if their net input is positive, and 0 if their net input is negative.[8] The state units are assumed to be linear. Consider now the operation of the network when it starts with the state units set to 0. The hidden unit has a negative bias, so its output is 0. Thus the net input to the output units is 0. However, these units have a positive bias, thus, they both have an output of 1, and the action A is produced by the network. At succeeding time steps, the same action A is produced until the activation on the state units is large enough to turn on the hidden unit. As shown in Table 2, this occurs after three occurrences of A. At this point, the hidden unit inhibits the output units and the action B is produced.

The more complex network in Figure 7 can produce the two sequences AB and AAAB. This network is essentially the same as before, but with an added hidden unit and two plan units. If the plan is $p = \begin{bmatrix}0\\1\end{bmatrix}$, then the second hidden unit is strongly inhibited and the network functions exactly as in the

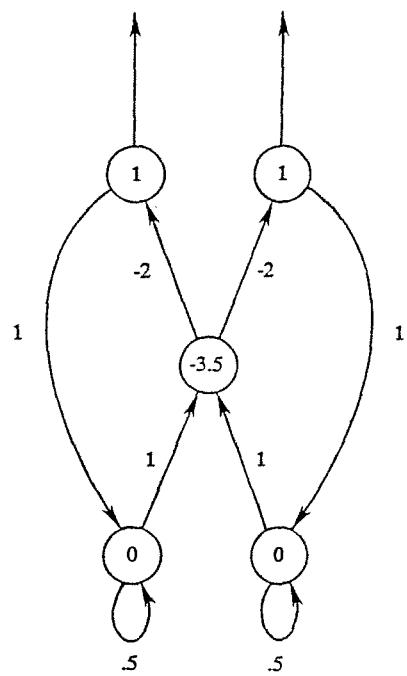

FIGURE 6. An example network with five units. The biases are shown in the circles representing the units, and the weights are shown near the links between units.

[8] Threshold units are used in the example for simplicity of exposition; all simulations to be described later, however, used the logistic function both for hidden units and for output units.

TABLE 2

ACTIVATION OF STATE UNITS

Time step	Activations		Sum
0	0	0	0
1	1	1	2
2	1.5	1.5	3
3	1.75	1.75	3.5

previous example, producing the sequence AAAB. If, on the other hand, the plan is $\mathbf{p} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, then the first hidden unit is inhibited, and the second hidden unit comes into play. This hidden unit has a smaller bias, and turns on after one occurrence of A. Thus, the network performs the sequence AB.[9]

These examples give some indication of how nonlinear hidden units allow networks to perform repeated actions and to use plan vectors to distinguish different sequences of actions. In general, however, the weights in the network arise not through design but through learning. A variety of simulation experiments have been carried out in which the network learned to perform sequences by changing the

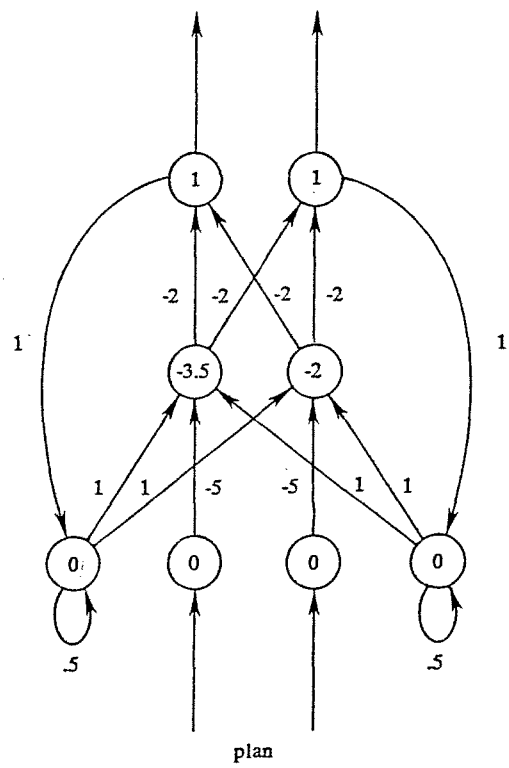

FIGURE 7. A more complex network with two hidden units and two plan units.

[9] This network is somewhat unrealistic in that it continues to perform the action A, even when the plan units are turned off. Note also that if the plan (0, 1) is left on the plan units, the network will cycle, but will not repeat AAAB. This issue is discussed below.

weights in the path from the state units to the output units. One such experiment used all possible permutations of the actions $A = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, $B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, and $C = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$. There were four hidden units and six plan units in the network, with two state units and two output units. The hidden units and the output units had a logistic activation function with $min = -1$ and $max = 1$, while the state units were linear. There were recurrent connections from the output units to the state units and from state units to themselves. The recurrent weight μ on these latter connections was set equal to 0.4. The plans were randomly chosen six-dimensional vectors, one for each permutation.

In the simulation, a learning trial involved the following sequence of events. All units were first initialized to zero, then a particular plan was chosen as the input to the plan units. At successive time steps, the output of the network was compared to the next action in the permutation being learned, and errors were propagated back into the network using the algorithm of Rumelhart, Hinton, and Williams (1986). This procedure was followed for each permutation, with the units being reinitialized between permutations. After 334 such learning trials, the network was able to correctly perform each permutation when presented with the appropriate plan vector.[10]

During learning, there is a choice as to the vector which is fed back to the state units at each time step. It is possible to feed back the actual output vector, or to feed back the desired output vector. When this second approach was used, the number of learning trials needed to learn the permutations decreased to 145. The learning is slower in the first case because the states change over the course of learning, and the mappings that are learned early on therefore need to be adjusted as learning proceeds. In further simulations, the second method was used exclusively. When the network performance was tested, of course, the actual output vectors were fed back.

In a second experiment, a similar network with one plan unit and two hidden units was taught to be an up-down counter. The network was required to produce the sequence

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

when the plan unit had an activation of -1, and to produce the same sequence in the opposite order when the plan unit had an activation of 1. When μ was set to 0.4, the network needed 78 trials to learn these two sequences.

Note that the sequences in the up-down counter cycle back to the action they started from. In the case in which μ is 0, the network is automatically able to repetitively cycle through the actions after only learning one cycle because in this case the network simply associates the previous action to the current action. On the other hand, when μ is greater than 0, the network will not be in exactly the same state after the last action as after the first action, because there will be some residual activation on the state units. If the network is allowed to cycle, the residual will build up and cause an error to occur. For example, a network with a value of $\mu = 0.5$ was taught the sequence AAABA, where A and B are as defined in the permutation experiment. When the network was allowed to cycle, it produced the sequence AAABAAABABAAAB, in which there is an error in the third cycle. For this network to be able to cycle correctly, it was necessary to teach it how to connect two cycles by teaching it the sequence AAABAAABA.

Table 3 shows the mappings from the state units to the output units that the network must learn in the case of the up-down counter when the weight μ is equal to zero. As can be seen in the table, the first output unit must compute the parity function. Parity is a nonlinear function which is difficult for these networks to compute (Minsky & Papert, 1969; Rumelhart, Hinton, & Williams, 1986). This example suggests that even for simple sequences, the output function f should be expected to be nonlinear.

---

[10] Correct performance was defined in terms of the sum of squared error over the output units, summed across all actions in all sequences. This total error was required to be less than .05.

TABLE 3

COUNTER TRANSITION FUNCTION

Plan	State units		Output units	
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

Certain sequences are more difficult than others for the network to learn. The number of trials needed to learn various sequences of the actions A, B, and C (as defined above) are shown in Table 4. In all cases, the network was started with small random weights and learned only one sequence. As can be seen, an important and obvious determinant of difficulty is sequence length. Also, repeated actions cause difficulty when different actions can follow the repeated action in the same sequence. Otherwise, repeated actions speed learning somewhat (compare AAB and AABA). The reader might wish to compare the intuitive difficulty of these sequences by defining them as repetitive tapping sequences.

Plan Representations

The major role of the plan vectors is to distinguish between the different sequences that the network can perform. Thus, as in the example of the permutations, the plans can be arbitrary vectors that merely serve to key a particular sequential process. However, there is much to be gained if plans have some structure that relates them to the sequences they produce. For example, if similar plan representations are used for similar sequences, then there will be generalization or positive transfer from one sequence to another. Also, in a noisy system, the inadvertent choice of a plan that is nearly the correct plan should be expected to lead to a simple error, such as a transposition, rather than to an entirely different sequence.

A comparison between plan representation schemes was made for the permutation-learning network described above. Three different ways of using the six plan units were compared. The first representation was an arbitrary representation. Ten different replications were run with different random choices for the plan vectors. In the second representation, the plan units were partitioned into three slots of two units each. Each slot directly represented one of the three actions in the sequence being learned. For example, the plan for the sequence ABC was (–1, 1, 1, 1, 1, –1), with A in the first slot, B in the

TABLE 4

LEARNING OF VARIOUS SEQUENCES

Sequence	Trials
AA	16
AB	29
ABC	34
AABB	45
AAB	47
AABA	55
ABAC	61
ACABAA	134

second slot, and C in the third slot. The third scheme represented the transitions in the sequence being learned. There are six possible transitions: AB, AC, BA, BC, CA, and CB. One plan unit was used for each transition. If that transition appeared anywhere in the sequence being learned, then the unit had a value of 1; otherwise it had a value of −1. For example, the plan for the sequence ABC was (1, −1, −1, 1, −1, −1).

The number of learning trials needed to learn all the permutations to criterion was 135 trials in the arbitrary representation, 129 trials in the slot representation, and 98 trials in the transition representation. The value given for the arbitrary representation is the average over the ten replications. Four of the arbitrary choices were better than the slot representation; none, however, were better than the transition representation. These results suggest that transitions are a good way to represent similarity between sequences in these networks, given the particular state representation that is being used in these simulations. It is also clear that the slot representation captures no underlying similarity in the sequences, due to the fact that the underlying mechanism does not take absolute position of actions into account. However, with larger action sets, such that the sequences being learned do not always involve the same actions, the slot representation would capture some of the similarity between sequences that involve the same actions. In this case, it would be expected to fare better than an arbitrary plan representation.

Dynamical Properties of the Networks

When a network learns to perform a sequence, it essentially learns to follow a trajectory through a state space. The state space consists of the ensemble of possible vectors of activation of the output units. An important fact about the learned trajectories is that they tend to influence points nearby in the state space. Indeed, the learned trajectories tend to be *attractors*.

Consider, for example, a network taught to perform the cyclic sequence

$$\begin{bmatrix} .25 \\ .25 \end{bmatrix}, \begin{bmatrix} .75 \\ .25 \end{bmatrix}, \begin{bmatrix} .75 \\ .75 \end{bmatrix}, \begin{bmatrix} .25 \\ .75 \end{bmatrix}, \begin{bmatrix} .25 \\ .25 \end{bmatrix}.$$

The trajectory of the network is on the four corners of a square in the first quadrant of the plane. The trajectory will repeatedly move around this square if the initial vector of activations of the output units is one of the corners of the square.[11] It is also possible to set the initial activations of the output units to other values, thereby starting the network at points in the space other than the four corners of the square. Figure 8 shows the results of a simulation experiment in which the network was started at the point (.4, .4). As can be seen, the trajectory spirals outward and begins to approximate the square more and more closely. When the network is started at a point outside of the square, the trajectory is found to spiral inward towards the square. A sample trajectory starting from the point (.05, .05) is shown in Figure 9.

The two figures taken together indicate that the the square is a *limit cycle* for the network. All trajectories eventually reach the square in the limit.[12] Note that trajectories starting inside the square approach the limit cycle less rapidly than do trajectories starting outside the square. At a point inside the square, the trajectory is subject to influences associated with all four corners, and these influences are in conflicting directions and therefore tend to cancel one another. At a point outside the square, however, only a pair of adjacent corners tend to influence the trajectory, and adjacent influences do not conflict in this example.

[11] I am assuming that the weight μ in the network is 0, or that the network has been explicitly taught to cycle.

[12] Technically, there is a single point in the interior of the square which is a point of unstable equilibrium.

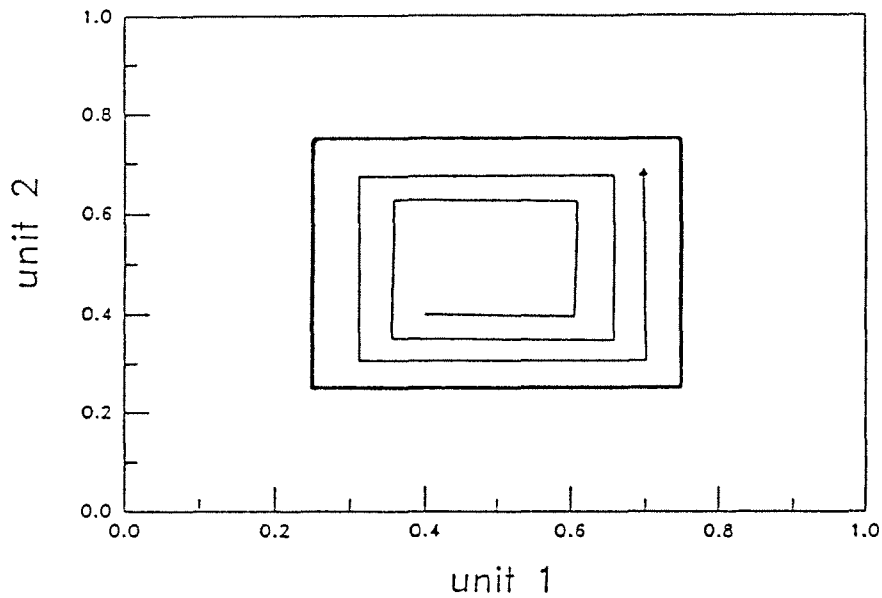

FIGURE 8. The activations of the two output units plotted with time as a parameter. The square is the trajectory that the network learned, and the spiral trajectory is the path that the network followed when started at the point (.4, .4).

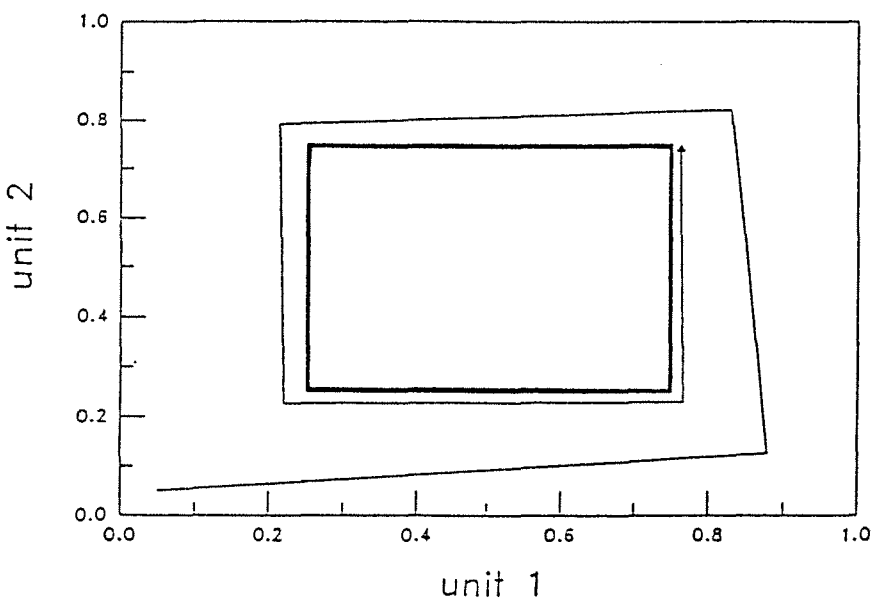

FIGURE 9. The activations of the two output units plotted with time as a parameter. The square is the trajectory that the network learned, and the spiral trajectory is the path that the network followed when started at the point (.05, .05).

Further experiments were conducted with an up-down counter, in which there are two cycles that are learned, one for each of two plan vectors. When the plan unit was set to either of the plans corresponding to the learned sequences, it was possible to observe the limit cycle behavior as before. However, when the plan unit was set to an intermediate value, the network cycled indeterminately in a complex manner depending on the particular value of the plan unit. Thus, for this network, the learned

sequences are not limit cycles. This is because there is no force pushing the plan unit towards one of the known plan vectors. However, such a force can be easily arranged: the plan units can be linked by symmetric connections in such a way that the known plans become point attractors for the subnetwork consisting only of the plan units (Hopfield, 1982). For example, in the case of the up-down counter, a single positive recurrent weight is sufficient to drive the plan unit towards −1 or 1, which are the plans corresponding to the learned sequences. When this is done, the entire network can be started in an arbitrary configuration, and the activation trajectories will eventually approach one of the two learned sequences. In this network, the learned sequences *are* limit cycles. The particular cycle that is approached in the limit depends on the initial values of the units; each of the limit cycles has a basin of attraction. In general, such networks will have multiple basins of attraction, one for each of the learned sequences.

An attractor dynamics of the kind exhibited by the networks described above has several nice properties. The system tends to be noise-resistant, because perturbed trajectories return to the attractor trajectory. The system is also relatively insensitive to initial conditions. Finally, the learning of a particular trajectory automatically generalizes to nearby trajectories, which is what is desired in many situations. The relevance of these properties to motor control has been recognized by several authors (Kelso, Saltzman, & Tuller, in press; Saltzman & Kelso, in press). I wish to suggest that such dynamics may also characterize the higher-level dynamical system which is responsible for serial ordering.

Learning With Don't-Care Conditions

The learning of sequences with don't-care conditions will be treated in more detail in the sections on coarticulation and dual-task performance. In this section, I discuss briefly the limiting case in which actions have no output units in common, so that every action has don't-care conditions for units used by other actions. In this case, due to the exclusive use of units by particular actions, the output units are perhaps better thought of as representations for whole actions, rather than as action features. That is, this case essentially involves local representations for actions.

A simulation was carried out using the sequence

$$\begin{bmatrix} .9 \\ * \\ * \\ * \end{bmatrix}, \begin{bmatrix} * \\ .9 \\ * \\ * \end{bmatrix}, \begin{bmatrix} * \\ * \\ .9 \\ * \end{bmatrix}, \begin{bmatrix} * \\ * \\ * \\ .9 \end{bmatrix},$$

in which a star is used to designate don't-care conditions. A network with two hidden units and a weight $\mu = 0.2$ learned to produce this sequence. Note that at each time step, errors are propagated back from only one output unit.

Table 5 shows the sequence that the network produced once learning was complete. As can be seen,

TABLE 5

SIMULATED NETWORK PERFORMANCE

Output unit	Activations			
1	.90	.81	.79	.77
2	.81	.90	.82	.79
3	.80	.82	.90	.83
4	.79	.81	.83	.90

Note. The four columns are the activations of the output units at four successive time steps.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 25

the values of .9 are correctly produced by the units at the appropriate times. Also, the don't-care conditions have been filled in by the network. The filled-in values are all above .75, thus the performance is highly parallel. When the weights in the network were inspected, it was found that these large activations were mainly due to the biases of the output units, which had grown fairly large. This explains the fact that there is little dropoff in the partial activations of the units over time: The partial activations are not determined by the time-varying state but rather by the constant biases. In a second simulation, the network was required to produce values of 0 on the output units for the two time steps immediately consecutive to the sequence being learned. This manipulation constrained the biases and insured that the values of .9 were achieved mainly by the weights in the path from the state units to the output units. The performance of the network after learning was complete is shown in Table 6. The table shows that the don't-care values that were filled in were smaller. The parallelism is more restricted in this example, and there is a dropoff in the partial activations over time. Also, at any given time step, there is a graded pattern of activation such that the current action is most highly active, and future actions are successively less active. Finally, notice that there is an asymmetry to the filling-in process: There is more anticipation of future actions than there is perseveration of past actions. This asymmetry has been observed in many other simulations and arises from the use of the exponential average representation of the state.

COARTICULATION

The theory presented in this paper involves a dynamical system which is constrained through a learning process to follow particular trajectories in a state space. The learning process relies on lists of constraints on the output units of the network.

In the case of speech, these constraint lists 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, phonemes constrain the dynamic process that produces utterances by changing parameters of the process until the constraints are met. During the learning process, parallel interactions between nearby phonemes can arise as long as they do not violate constraints.

In this section, I present some simple simulations to demonstrate some predictions that the theory makes. It should be emphasized that I am not proposing a realistic model of speech production in this section. The physical level of the speech articulators is itself a complex dynamical system with inertias, stiffnesses, and other dynamical parameters. The output of the network is best thought of as influencing articulator trajectories indirectly, by setting parameters or providing boundary conditions for lower level processes which have their own inherent dynamics. Saltzman and Kelso (in press) have recently presented a mathematical framework in which it is possible to model such lower level dynamics. Their approach may eventually provide a reasonable set of parameters in terms of which constraints on the output units of the network can be defined. For present purposes, however, I have simply used a subset of traditional speech features to provide constraints for the output units. Furthermore, only the simplest

TABLE 6

SIMULATED NETWORK PERFORMANCE

Output unit	Activations			
1	.90	.62	.35	.17
2	.59	.90	.70	.36
3	.37	.79	.90	.53
4	.26	.66	.81	.90

Note. The four columns are the activations of the output units at four successive time steps.

form of constraints were employed — value constraints and don't-care conditions. The value constraints were adapted from a list of real-valued features proposed by Ladefoged (1982). Choices for don't-care conditions were based on known allophonic variations when possible (for example, the rounding for the French */s/* was taken to be a don't-care condition, because it is possible to have a rounded or an unrounded */s/*). These simplifications make it unrealistic to attempt to model the details of the time course of coarticulation. However, certain more global aspects of coarticulation can still be discussed and the general operation of the network further elucidated.

The problem of serial ordering in speech is typically treated in discrete terms, and the relationship between discrete higher level processes and continuous lower level articulatory processes has provoked much debate in the literature on speech production (Fowler, 1980; Hammarberg, 1982; Perkell, 1980). In the current theory, however, such issues are not particularly problematic, because the entire system can be thought of as operating in continuous time. It is consistent with the current theory to assume that the defining state equations are simply a discrete version of a continuous dynamical system. In the continuous case, learning involves imposing constraints intermittently on the system at various points in time. In geometric terms, constraints appear as regions through which continuous network trajectories must pass, with trajectories between regions unconstrained.[13] To approximate the continuous system in the simulation, I have inserted several time steps between steps at which constraints are imposed. During these intermediate time steps, the network is free running (these intermediate steps can be thought of as having don't-care conditions on all of the output units). By conducting the simulation in this manner, it is possible to demonstrate the differences between the current approach and an assimilatory model in which different allophones are produced at each time step and interactions must begin and end at allophonic boundaries (cf. Fowler, 1980).

Simulation Experiments

Representations for the phonemes were adapted from Ladefoged (1982). Eight features were selected that provided adequate discriminations between the particular phonemes used in the simulations. The feature values were all between 0.1 and 0.9.

The network used in the simulations had 8 output units, 10 hidden units, 6 plan units, and 8 state units. The state units had recurrent connections onto themselves with weights of $\mu = 0.5$.

The procedure used in the simulation was essentially that of the preceding section, with the following modification. During learning trials, constraint vectors were presented to the network every fourth time step. Learning occurred only on these time steps. During the intermediate three time steps, the units were updated normally with no learning occurring.

In the first experiment, the network was taught to perform the utterance *sinistre structure*. The phoneme representations which were used are shown in Table 7, for the embedded sequence */istrstry/* only. The learning process involved repeated trials in which the phonemes in the sequence were used as constraint vectors for the network. The plan was a particular constant vector whose composition is irrelevant here because the network learned only this one sequence. The results for the embedded sequence */istrstry/* are shown in Figure 10, which displays the output trajectories actually produced by the network once the sequence was learned to criterion. The network has learned to produce the specified values, as can be seen by comparing the values produced at every fourth time step with the values in the table. The network has also produced values for the don't-care conditions and for unconstrained parts of the trajectories. In particular, the value of .9 for the rounding feature of the rounded vowel */y/* is being anticipated as early as the third time step. In a control experiment, the sequence *sinistre stricture*, in which the same consonant sequence is followed by the unrounded vowel */i/*, was taught to the network. As shown in Figure 11, there is no rounding during the entire utterance. These results parallel the results obtained by Benguerel and Cowan (1974).

[13] A game of croquet, with large, perhaps overlapping, wickets provides a picturesque analogy.

TABLE 7

THE PHONEMES OF */istrstry/*

Feature	*i*	*s*	*t*	*r*	*s*	*t*	*r*	*y*
voice	8	1	1	*	1	1	*	8
place	7	9	9	2	9	9	2	7
sonorant	8	2	1	5	2	1	5	8
sibilant	1	9	2	4	9	2	4	1
nasal	*	*	1	*	*	1	*	*
height	9	9	9	9	9	9	9	9
back	1	*	*	2	*	*	2	1
round	1	*	*	*	*	*	*	9

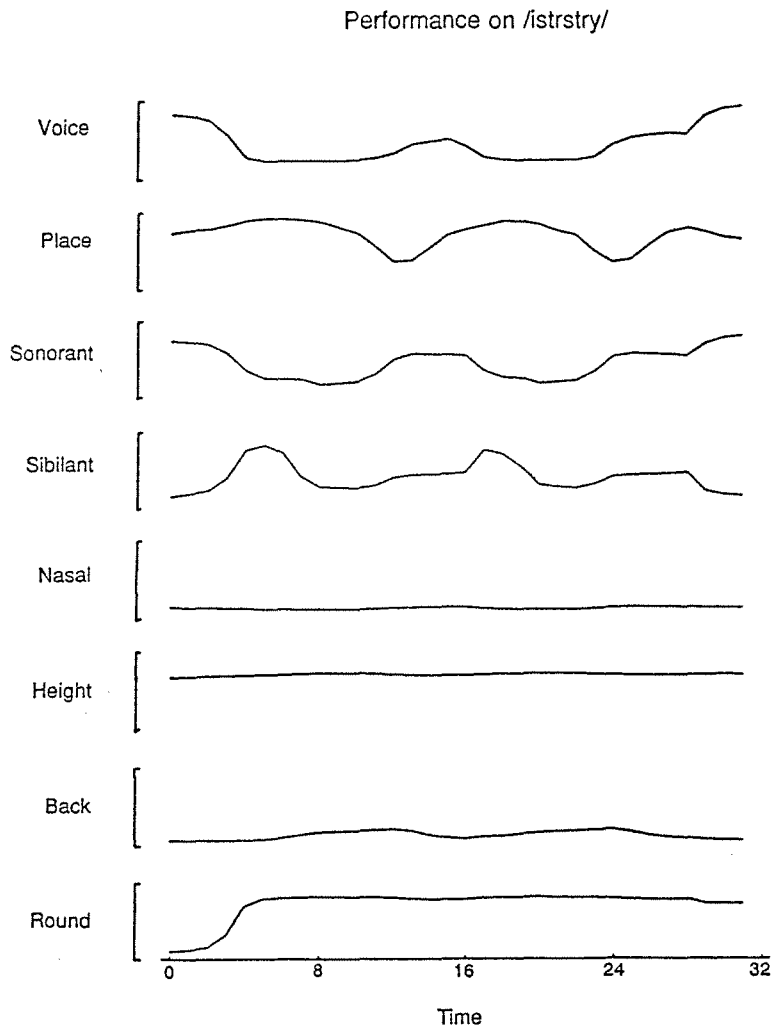

FIGURE 10. Output trajectories for the sequence */istrstry/*.

In a third experiment, the network learned the word *freon*, where the feature of interest is the nasal feature associated with the terminal */n/*. In the phoneme vectors, the */f/* was specified as 0.1 for the nasal feature, the */n/* was specified as 0.9, and the intervening three phonemes had don't-care values for

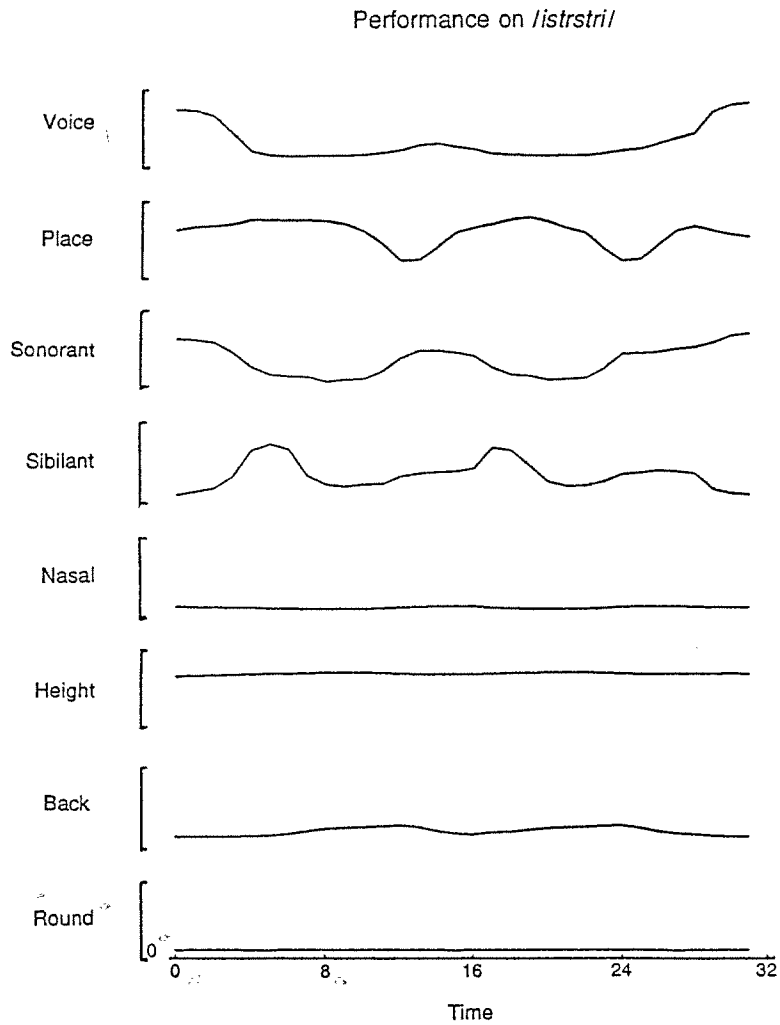

FIGURE 11. Output trajectories for the sequence */istrstri/*.

the nasal feature. Thus, this experiment is analogous to the previous experiment, with the interest in the anticipation of the nasal feature rather than the rounding feature. The results are shown in Figure 12, in terms of the activation of the nasal feature at every fourth time step. As in the data of Moll and Daniloff (1971), there is substantial anticipation of the nasal value of the */n/* before and during the two vowels. Note that there is a steeper dropoff in the amount of anticipation in this sequence than in the sequence */istrstry/*. An investigation of the weights learned during these sequences revealed that the extensive coarticulation in the latter sequence arises from the repetition of phonemes. The rounding of */y/* is produced in a temporal context in which */str/* was the preceding subsequence. A very similar context occurs after the first */r/*, thus, there is necessarily coarticulation into the first repetition of */str/*. These considerations suggest that there should be more coarticulation over strings that have homogeneous phonemic structure than over strings with heterogeneous phonemes.

Another interesting aspect of the way in which coarticulation occurs can be seen by considering the voicing feature in Figure 10. This feature is unspecified for the phoneme */r/*,[14] but is specified as a 0.1 for the directly adjacent features */t/* and */s/*. Nevertheless, the first */r/* receives a small amount of voicing, which comes from the positive value of voicing for the nearby, but not adjacent, phonemes */i/*

[14] The */r/* in French can be voiced or unvoiced depending on the context; compare *rouge* and *lettre*.

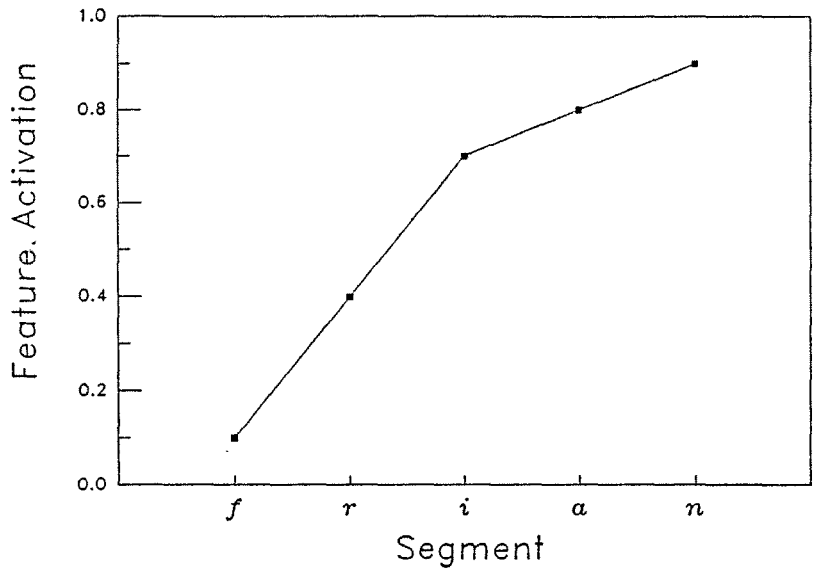

FIGURE 12. Activation of the nasal feature at every fourth time step during performance of the word *freon*.

and /y/. This result emphasizes the underlying mechanism of activation of the output units: Units are activated to the extent that the current state is similar to the state in which they were learned. This means that units with don't-care conditions take on values that are, in general, a compromise involving the values of several nearby phonemes, and not simply the nearest specified value. Typically, however, the nearest phoneme will have the most influence.

These considerations suggest that the amount of forward coarticulation should be expected to depend not only on the preceding phonemes, but also on the following phoneme. If the phoneme following /y/ is unrounded, for example, then there should be less anticipation of the rounding of the /y/ than when the following phoneme is rounded or unspecified on the rounding feature (as in the example of *structure*). This prediction is borne out in simulation. The French pseudowords *virtuo*, *virtui*, and *virtud*, in which the rounded phoneme /y/ is followed by the rounded phoneme /o/, the unrounded phoneme /i/, or the "don't-care" phoneme /d/, were taught to the network. The results are shown in Figure 13, in terms of the activation of the rounding feature at successive points in time. The figure shows that forward coarticulation in the network clearly depends on the following context.

Discussion

In their review on coarticulation, Kent and Minifie (1977) distinguish between submovements in an articulatory sequence that have "immediate successional impact," that is, those that "must follow one another in a prescribed sequence," and submovements without immediate successional impact, that are "accommodated within the sequential pattern defined by the locally critical articulatory transitions." The model presented in this section obeys this distinction, where constraints specify the locally critical articulatory transitions. The model also provides a mechanism for the process of "accommodation," by which features without immediate successional impact can be integrated into the articulatory program.

If the distinction made by Kent and Minifie is correct, the question arises as to how the system knows which transitions have immediate successional impact and which do not. The model presented here assumes that this knowledge is encoded in the definitions of the phonemes. Note that these

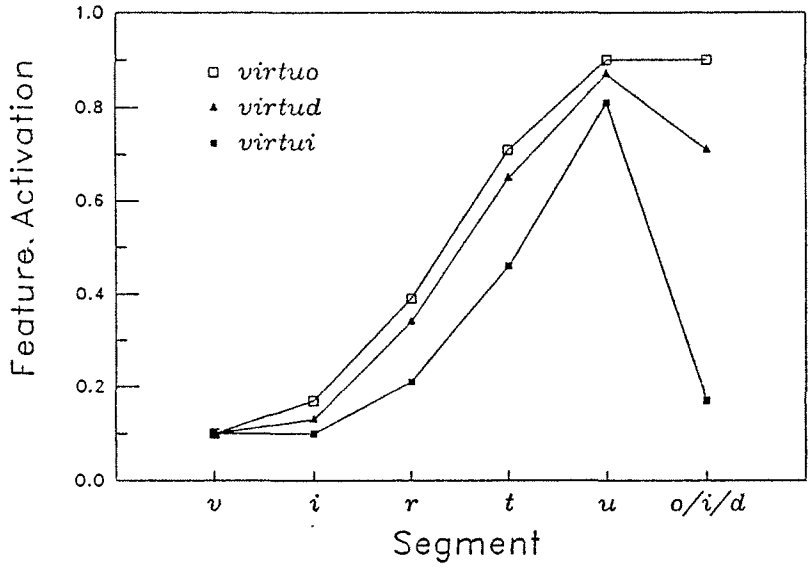

FIGURE 13. Activation of the rounding feature at every fourth time step during performance of three French pseudowords.

definitions are themselves independent of context: They specify in what ways a phoneme can be altered by its context, without specifying values for particular contexts.

It is worthwhile to compare the current model to a feature-spreading model such as that proposed by Henke (1966). Henke's model is essentially a buffer model, in which positions in the buffer are loaded with the phonemes to be produced. Phonemes are lists of trinary features, each of which can have the value +, –, or 0. When a buffer position is to be executed, features having value 0 are filled in by an operator that serially inspects "future" buffer positions until a plus or a minus is found. Once all features are filled in, the allophonic variation thus created can be executed. Although this model is similar to the current model in the sense that both rely on context-independent representations of phonemes that specify dimensions along which the phonemes can be altered, there are important differences. From a conceptual point of view, the underlying mechanisms that determine output values are quite different and have different empirical consequences. In the current model, parallel performance arises automatically, without the need for a special process to program in the parallelism. This occurs because the current state is similar to the state in which nearby phonemes were learned, and similar states tend to produce similar activations of the output units. There is therefore no implication that features can spread indefinitely in time, which is true of a strict interpretation of Henke's model (cf. Gelfer, Harris, & Hilt, 1981). Rather, the spread of a feature in time diminishes due to the dropoff in similarity of the state. For similar reasons, there is no implication that feature vectors change discretely in time. As the state evolves continuously in time, the components of the output vector also evolve continuously in time, with no necessary coherence between anticipated or perseverated features and adjacent segments (cf. Fowler, 1980). Indeed, there is really no notion of a segment in the output of the network. Also, whereas Henke's model is an assimilatory model of coarticulation, the current model is best thought of as a model of parallelism in speech production. As shown in the simulations, the parallel model predicts nonadjacent interactions: For example, the amount of forward coarticulation of a feature in a phoneme depends on what follows the phoneme. Although an assimilatory model could be constructed to mimic this behavior (if need be), it would seem better accounted for within the parallel approach. However, I know of no empirical evidence relevant to deciding this issue. Finally, it should be noted that in the current model, utterances are not explicitly represented (i.e., in a buffer) before

being produced. Rather, the process is truly dynamic; utterances are implicit in the weights of the network, and become explicit only as the network evolves in time.

The simulations presented above relied only on the simplest constraints on the output units. However, there is much to be gained by considering more complex constraints such as inequality constraints, range constraints, or constraints between units. Certain effects of context, such as the dentalization of the */d/* in *width*, are often treated as phonological in origin, rather than resulting from coarticulation. In the current model, however, the */d/* could be represented as having a range constraint on the place of constriction feature (i.e., a constraint that the place be between a pair of values). The actual value chosen for the place feature will be dependent on the neighboring context through a constraint satisfaction process during learning, rather than dynamically at the time of production. Similarly, constraints between units can determine which gesture is chosen out of several possibilities. For example, if the sum of the activations of three output units must be a particular value, then it is possible to trade off the activations among the units if particular units are further constrained by neighboring context.

There are two possible versions of a parallel model of coarticulation. The first assumes that parallelism is feature-specific, that is, that particular features of a phoneme can be anticipated or perseverated. This approach is consistent with the distinction of Kent and Minifie (1977) discussed above, and is the approach that I have emphasized. However, it is also possible to assume that all of the components of a phoneme must be activated together. This is the approach favored by Fowler (1980), who claims that coarticulation results from the coproduction of "canonical forms." In the current framework, such phoneme-specific parallelism occurs when phonemes specify constraints on nonoverlapping sets of output units. In the limiting case, each phoneme can constrain a unique output unit, in which case the partial activations of output units lead to the partial production of entire phonemes rather than specific features. It is still possible to represent phonemes by features, but this must be done at a lower level in the system, below the level at which parallelism arises.

However, it would appear that feature-specific parallelism is necessary. For example, in the production of a sequence of vowels followed by an */n/*, it would seem important that only the velar movement associated with the nasal be anticipated, and not the alveolar tongue position. There is some evidence for this kind of phenomenon in the data of Kent, Carney, and Severeid (1974). In recordings of the articulatory movements during the utterance *contract*, they found that the movement towards the alveolar tongue position for the */n/* began 120 milliseconds after the onset of velar lowering for the */n/*. More detailed investigation of this issue, particularly EMG studies, would seem highly relevant to a better understanding of coarticulation and the representation of speech.

To summarize, the current proposal is that coarticulation results from the similarity structure of the state at nearby points in time. The dropoff in similarity of the state defines the zone in which the features of a phoneme can possibly be present in the output. Within this zone, the pattern of coarticulation that is obtained depends on the constraints that are imposed by other nearby phonemes.

DUAL-TASK PARALLELISM

In previous sections, I have concentrated on the interactions that arise between actions in a sequence. It has been shown how a network that learns to produce sequences comes to merge actions so that they are produced at least partly in parallel. I now discuss interactions between actions that are to be performed simultaneously. In this case, parallelism is not simply allowed by the task demands, it is required.

In a simple sense, such parallelism is already exhibited by the networks previously discussed. The actions produced by the network are vector-valued and can be thought of as composed of subactions that are performed in parallel. However, these subactions have been treated only as parts of a whole, and have not been thought of as being separate actions which can be performed alone. I now wish to focus on the "dual-task" problem: How is it possible for actions that have been learned separately to be performed simultaneously? What I intend to show is that the interference that arises in such a situation

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 32

is due to the same mechanism as the coarticulatory "interference" that arises between actions in a sequence. Furthermore, the learning process which eliminates dual-task interference is of the same form as that required to block coarticulatory interactions when neighboring actions are not allowed to overlap.

Figure 14 shows a simplified network model of interaction between tasks. In the figure, both the input units and the output units can be partitioned into nonoverlapping sets of units, over which states and outputs corresponding to the two tasks can arise simultaneously and independently.[15] The tasks interact because the mappings from states to outputs pass through overlapping sets of hidden units. The hidden units therefore form a channel in which task interference can arise. Such interference is not inevitable because it is perfectly possible for vectors of activation to coexist on a set of units without creating interference. This will be the case if the weights emanating from the channel are organized in such a way as to "filter" the relevant vectors from the overall activation pattern. However, interference is highly likely when the tasks have been learned separately, and special learning regimes are needed to allow simultaneous performance.

Consider first the learning of a single task in isolation. Suppose that this involves the learning of an association between a vector $\mathbf{s}_1$ on the $state_1$ units and a vector $\mathbf{x}_1$ on the $output_1$ units. The learning should be restricted to the path between the $state_1$ units and the $output_1$ units. Given the way the network learning rule works, the entire network can be thought of as learning an association between the state vector $<\mathbf{s}_1, 0>$ and the output vector $<\mathbf{x}_1, *>$, where 0 stands for zero activation on all of the $state_2$ units and the star stands for a vector of don't-care conditions on the $output_2$ units. This notation emphasizes the fact that the desired output vector specifies only those values that are relevant to the association being learned (it is also likely that there are don't-care conditions within the $\mathbf{x}_1$ vector, so that coarticulation is possible). Learning of only the relevant values simplifies the learning process;

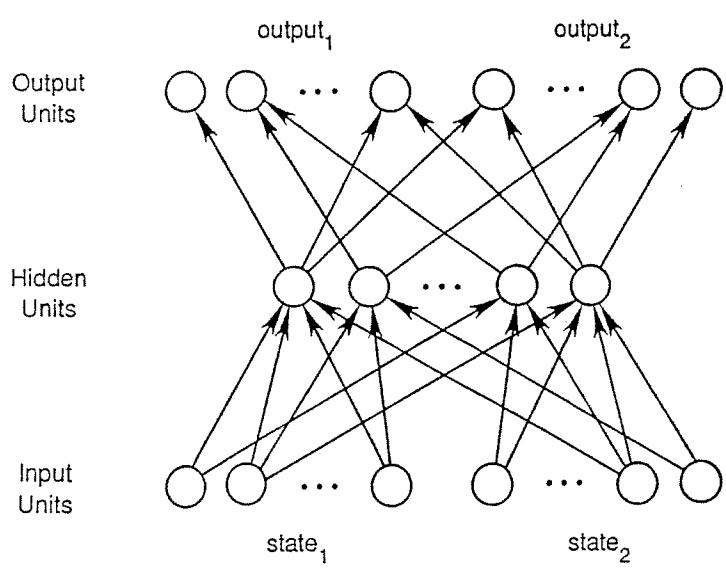

FIGURE 14. A network model of task interference. The output units are partitioned into two sets of units corresponding to the two tasks, as are the input units.

[15] In what follows, I have simplified the notation by omitting reference to the plans.

however, it also means that the activation of the $state_1$ units will produce spurious activation of the $output_2$ units, due to the existence of the shared channel. This spurious activation or *crosstalk* will pose no problem for single task performance, because these output units can be inhibited as a group. However, there will be interference when tasks involving both sets of units are performed simultaneously, because the crosstalk from one task will conflict with the activations produced by the other task (and vice versa).

There are two ways in which learning can suppress this interference and allow dual-task performance to improve. First, the crosstalk can be *eliminated*, by relearning the tasks in such a way as to take each other into account. Second, the crosstalk can be *incorporated* into the output activations. This involves a different sort of relearning in which the two tasks are redefined as a single combined task. In the combined task, the state vectors and the output vectors are simply the juxtaposition of the corresponding vectors in the tasks as defined separately. As will be shown, this second method has the disadvantage of potential negative transfer to single task performance.

Simulation Experiments

A number of simulation experiments have been carried out to investigate some of the relationships between crosstalk, task similarity, and learning.

The network used in the simulation is essentially that shown in Figure 14. There were 6 input units, 3 hidden units, and 6 output units, with all possible connections between the input units and the hidden units, and between the hidden units and the output units. The units themselves were as described in Equation 1, with linear output functions. The assumption of linearity is made simply for purposes of clarity of presentation, and the results obtained apply qualitatively to the nonlinear case as well.

The following notation will be used to describe the simulations. There are assumed to be two tasks, each involving a single association. The state vectors associated with the tasks are denoted $\mathbf{s}_1$ and $\mathbf{s}_2$, respectively. They are three-dimensional vectors representing the activation patterns on nonoverlapping sets of units—$\mathbf{s}_1$ is a pattern on the first three input units, whereas $\mathbf{s}_2$ is a pattern on the remaining three units. The single-task situation involves the presentation of either $\mathbf{s}_1$ or $\mathbf{s}_2$, with zeroes on the remaining input units. Thus, the full input vector is either $<\mathbf{s}_1, 0>$ or $<0, \mathbf{s}_2>$. In the dual-task situation, the pattern $<\mathbf{s}_1, \mathbf{s}_2>$ is presented to the input units. The desired output vectors are also three-dimensional and are denoted by $\mathbf{x}_1$ and $\mathbf{x}_2$. The first task is performed correctly when $\mathbf{x}_1$ appears on the first three output units, with undefined values on the remaining output units. For the second task, the vector $\mathbf{x}_2$ should arise on the last three output units, with the first three units undefined. In dual-task performance, the desired output vector is $<\mathbf{x}_1, \mathbf{x}_2>$. I will use the star notation introduced earlier to refer to don't-care conditions. Thus, the pattern $<*, \mathbf{x}_2>$ is a six-dimensional vector with the first three values unconstrained. Finally, the patterns of activation on the hidden units are denoted $\mathbf{v}_1$ and $\mathbf{v}_2$. Unlike the other vectors, these vectors arise on the same set of units. In the dual-task condition, due to the linearity assumption, the pattern $\mathbf{v}_1 + \mathbf{v}_2$ will appear on the hidden units. It is the relationship between $\mathbf{v}_1$ and $\mathbf{v}_2$, as well as the mappings from these vectors to the output vectors, that determine the interference between processes.

The learning rule is the same as that used in previous networks. During each learning trial, the actual output of the network is compared to the desired output, and the weights are changed to reduce the discrepancy. This occurs only for outputs with specified values; no learning takes place for don't-care conditions. In all of the simulations, the network was started with zero initial weights.

Experiment 1

As discussed above, dual-task interference is assumed to arise from crosstalk through a shared channel. The first experiment shows how to quantify this crosstalk and how it can be eliminated through learning.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 34

Particular random values were chosen for the vectors s_1 and x_1, and the network learned the mapping from s_1 to x_1 until a criterion performance was achieved.[16] The full patterns used on each learning trial were $<s_1, 0>$ on the input units and $<x_1, *>$ on the output units. With these patterns, the properties of the learning rule lead to only those weights in the path from s_1 to x_1 being changed.

Once the first task had been learned, two different versions of the second task were created. The two versions were used in separate replications of the experiment, and were created by the following procedure. First, a particular vector v_2 was chosen such that its inner product with v_1 was a particular value (0.5 in the first version of the task and 0.7 in the second version).[17] Then, by inverting the matrix from the s_2 units to the hidden units, a particular s_2 vector was found that produced v_2. Finally, the desired output vector x_2 was defined to be the output produced when $<0, s_2>$ was presented to the network, effectively eliminating the need for learning of the second task. This manner of defining the second task was simply a convenient way of insuring a particular value of the inner product between v_1 and v_2 at the outset.

At this point, the network could correctly perform the two tasks in single-task conditions. However, no provisions had been made for simultaneous performance, and it was expected that there would be interference in this condition.

Dual-task trials were performed by putting $<s_1, s_2>$ on the input units and observing the output produced on the x_1 units. The output error was calculated as the sum of the squared error between x_1 and the actual output. This error constitutes a measure of crosstalk because x_1 was correctly produced in the single-task condition. The data are shown as the left most data points in Figure 15. As can be seen, the crosstalk is larger when the inner product of v_2 with v_1 is larger. This is the case in which there is more similarity in the representations of the simultaneous processes in the shared channel.

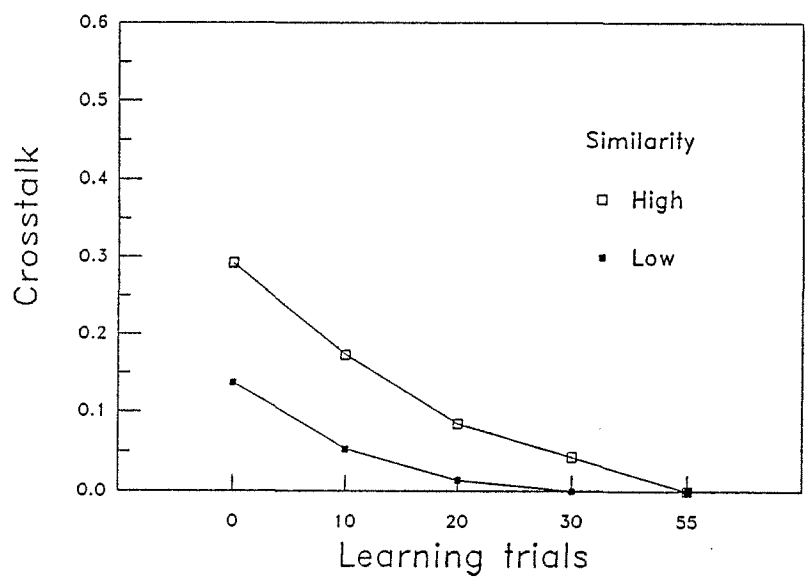

FIGURE 15. Crosstalk as a function of amount of learning for high and low similarity tasks.

[16] The criterion was that the squared error, summed over the output units, was less than .01.

[17] The vector v_1 is that vector present on the hidden units when $<s_1, 0>$ is put on the input units. It is determined entirely by the learning process. In both versions, the vector v_2 was chosen such that $||v_2|| = ||v_1||$.

Next, a learning process was initiated whereby the crosstalk could be eliminated and dual-task performance improved. Ten learning trials were run in which the mapping $<0, \mathbf{s}_2>$ to $<0, \mathbf{x}_2>$ was learned. Note that the values of the first three output units, which would normally be don't-care features, are now specified as 0. The $\mathbf{s}_1$ to $\mathbf{x}_1$ mapping was then relearned to criterion, and crosstalk in the dual-task condition was remeasured. As shown in the figure, the crosstalk was lowered and dual-task performance was therefore improved. The learning process was then repeated, leading to even less crosstalk. Eventually, after a sufficient number of learning trials, the dual-task interference was reduced to zero. This held for both levels of initial similarity. At this point, the tasks could be performed correctly in both the single-task condition and the dual-task condition.

Experiment 2

Experiment 1 demonstrated the influence of task similarity on crosstalk, where task similarity is defined by the similarity in the representations of two tasks in a shared channel. In Experiment 1, the two tasks were the only tasks that the network learned to perform. In a more realistic situation, however, the network will have learned many other tasks, and this prior learning may have some effect on the crosstalk observed between tasks. In terms of a communication metaphor, it might be expected that the shared channel has some inherent capacity, and as this capacity is approached, interference between processes will grow.

Experiment 2 was performed to investigate the notion of interference due to channel capacity. The experiment involved a replication of Experiment 1, with the following change. Six pairs of vectors were selected randomly and the network learned to associate these vector pairs. Three of the pairs involved the first three input and output units and three of the pairs involved the remaining input and output units.[18] The procedure of Experiment 1 was then followed, using the same vectors $\mathbf{s}_1$ and $\mathbf{x}_1$. The vectors $\mathbf{s}_2$ and $\mathbf{x}_2$ were again defined such that at the beginning of the dual-task trials, the similarity between the tasks was 0.5 and 0.7, and both mappings were being performed correctly in the single-task condition.

The results in the dual-task condition are shown in Figure 16. The results are similar to those shown in Figure 14; however, the crosstalk is initially much larger than in Figure 14, and it takes more trials to reduce the crosstalk to zero. There is once again an effect of similarity, with more similar tasks producing more crosstalk.

The larger crosstalk in this experiment is due to proactive influences being evoked through the shared channel. It is not difficult to see why this must happen. Consider the weights between the hidden units and the first three output units. These weights implement the mapping from $\mathbf{v}_1$ to $\mathbf{x}_1$ and thus allow the first task to be performed correctly. However, the same weights also implement other mappings involving the prelearned vector pairs. When $\mathbf{v}_1 + \mathbf{v}_2$ is present on the hidden units in the dual-task condition, the $\mathbf{v}_2$ vector evokes some of these other mappings, to the extent that these other mappings involve similar representations on the hidden units. This leads to crosstalk over and above the direct influence of $\mathbf{v}_2$ on the $\mathbf{v}_1$ to $\mathbf{x}_1$ mapping that was observed in Experiment 1.

Experiment 3

The third experiment investigated how learning can occur in the dual-task condition through the incorporation rather than the elimination of crosstalk. In the first two experiments, the dual-task learning procedure respected the integrity of the separate tasks. That is, learning trials involved learning one task or the other, with the added stipulation that a task not produce crosstalk on units used by the other task. This learning procedure would be expected to improve both dual-task and single-task

[18] The procedure was repeated ten times, with different random selections of the vector pairs. The results are averages over the repetitions.

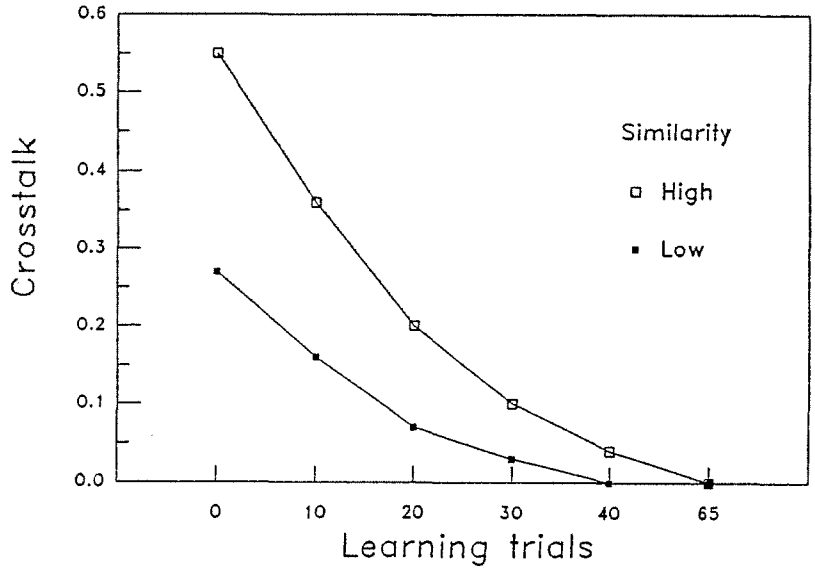

FIGURE 16. Crosstalk as a function of amount of learning for high and low similarity tasks, in a network with previous learning.

performance. Another learning procedure is possible in which the tasks are integrated into a single combined task. In the combined task, the input and output patterns are the juxtapositions of the patterns in the tasks as defined separately. Thus, dual-task learning trials involve learning to associate the combined input vector $<s_1, s_2>$ with the combined output vector $<x_1, x_2>$.

As in Experiment 1, the mapping from s_1 to x_1 was first learned to criterion, and the vectors s_2 and x_2 were defined so that the network was able to correctly perform both tasks in single-task conditions. At this point, error in the dual-task condition was exactly as in Experiment 1. These data are shown as the left most data points in Figure 17. The second learning procedure was then used to learn the combined task. As can be seen in the figure, the error in dual-task performance eventually goes to zero. Transfer to single-task performance was then investigated by presenting the network with the vector $<s_1, 0>$. The error, which was zero before the dual-task learning trials, was found to have increased to 0.18. Relearning the task separately led once again to error in dual-task performance; but it was possible, through alternation of the two learning procedures, to eventually find a set of weights where both single-task and dual-task performances were correct.

Discussion

The simulation experiments show how phenomena similar to those observed in behavioral experiments on dual-task performances can arise from crosstalk in a parallel distributed processing network. Interference due to crosstalk was shown to be a function of the similarity of the tasks in terms of their representations in a shared channel. Interference also arises from limited channel capacity when the same channel is used to implement several mappings. Both sources of interference have a graded deleterious effect upon performance; the network thus obeys the principle of graceful degradation (Norman & Bobrow, 1975).

Interference can be lessened and eventually eliminated through learning. The learning procedure essentially finds representations that interfere minimally so that tasks can be performed in parallel. This process has implications for the distinction between automatic and serial processes (Shiffrin & Schneider, 1977). It suggests that the automaticity of a task is always relative to a particular set of

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 37

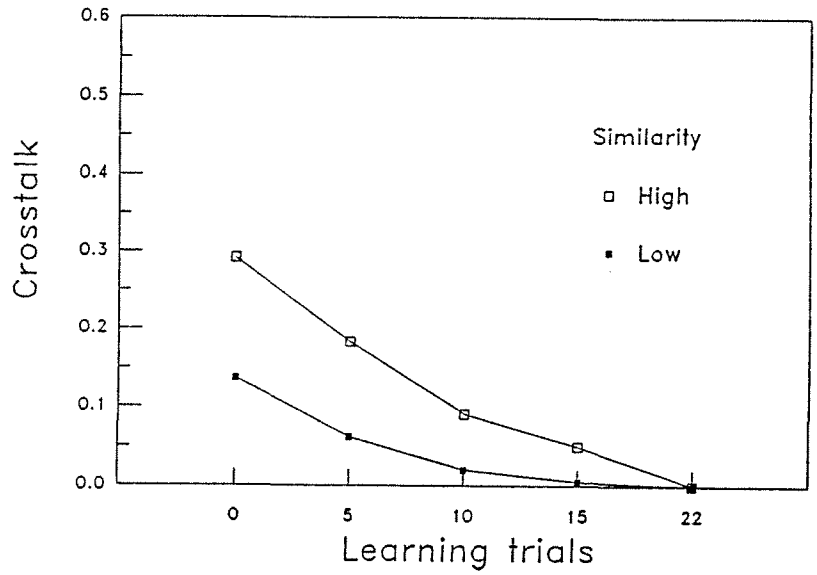

FIGURE 17. Amount of crosstalk as a function of learning in the integrated learning condition.

tasks with which it no longer interferes. There may be other tasks that show interference with an "automatic" task. This is a very different conception of automaticity from the view that says that a task is automatic when it no longer needs limited resources.

A final point should be made about learning. If crosstalk can be eliminated through learning, the question arises as to why it is not eliminated *ab initio*. That is, during the original learning process, it would seem possible to specify values for all don't-care conditions so that no crosstalk would be produced once learning was complete. However, there are several reasons why this approach would not be possible. First, it would destroy previously established connections within domains (learning with don't-care conditions on units protects associations previously made to those units). It would also destroy potentially useful connections that had previously been established between domains, such as those underlying the relationships between speech and hand gestures. Also, in general, the number of units that have don't-care conditions would be expected to be very much larger than the number of units with constrained values (for example, in speech no values would be specified for limbs, hands, etc.). Even if it were possible to specify values for all of these units, this would impose a large number of further constraints upon the weights being learned and slow learning considerably. Finally, as argued in previous sections, don't-care conditions are needed for interactions between actions in a sequence. Such coarticulatory interactions can be thought of as a useful form of crosstalk.

In general, both coarticulatory parallelism and dual-task parallelism would be expected to be present in complex sequential behavior. They essentially involve interactions across time and across space, respectively (where by "space," I mean different motor subsystems).[19] Coarticulatory parallelism is possible because the generalizations leading to actions spreading in time have been allowed. Dual-task parallelism is possible because generalizations leading to interference (actions spreading in "space") have been suppressed.

[19] Perhaps a better terminology would be the linguist's "syntagmatic" and "paradigmatic" dimensions.

OTHER ISSUES

The present paper has concentrated on only certain aspects of the serial order problem, namely, those involving temporal ordering, learning, and parallelism. However, there are many other phenomena that are relevant to the serial order problem and in this section I briefly consider some of these other phenomena. The discussion here should be taken only as indicating directions in which further research should proceed.

Rate

There are several possible approaches to making the overall rate of performance speed up or slow down in the networks considered here. One approach would be to add tonic excitation or inhibition to the entire network. However, this would likely have the effect of expanding or contracting behavioral sequences without regard to their content, which is not what is typically observed (Gentner, 1985). Another approach, similar to that used by Rumelhart and Norman (1982), is to have mutually inhibitory connections with variable gain between the output units. In the current approach, these connections would not be used to encode serial order, but rather would allow a kind of competition in which stronger outputs would suppress weaker outputs. Larger values of inhibition would tend to suppress weak coarticulatory interactions and slow down the overall rate of performance.

Errors

A discussion of the kinds of errors that can be generated by a sequential network modified to have a stochastic activation rule is beyond the scope of this paper. However, the basic principle is fairly clear: The network embodies the assumption that similar inputs tend to lead to similar outputs, so that cases in which discriminations must be made between similar plan and state vector pairs yield the most potential for error. This means that capture errors, substitution errors, and omission errors would be likely (cf. Grudin, 1983; Norman, 1981). Also, increasing parallelism between neighboring actions can be expected to increase the probability of errors (cf. Rumelhart & Norman, 1982).

Dell (Dell, 1984; Dell & Reich, 1980) has shown how a variety of error patterns, including transpositions and spoonerisms, can be accounted for in a connectionist model. The output of his model is a single static vector encoding the string of phonemes to be produced, thus the problem of temporal performance is not directly addressed. However, this approach could be treated as a specification of how the *plan* is set up, in the terminology of the current theory. This approach of assuming that errors can be made in setting up the plan is able to account for the fact that coarticulatory interactions tend to be appropriate to the sequence actually produced, not the intended sequence (Harris, 1984).

Hierarchies

Many researchers have suggested that motor sequences are organized into hierarchical structures (Albus, 1981; Estes, 1972; Greene, 1972; MacKay, 1982; Miller, Galanter, & Pribram, 1960; Povel & Collard, 1983; Rosenbaum, Kenny, & Derr, 1983). The concept of a hierarchy is not inconsistent with the current approach — it is possible to construct layered systems where each layer is a sequential network of the kind discussed in this paper. The output units of higher layers would constitute plan units for lower layers, and it would be necessary to have backward connections from the state units in a layer to the state units in the preceding layer. Some of the technical considerations involved in this construction are discussed in Jordan (1985).

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2009
Page 39

CONCLUSIONS

The current theory provides an alternative to the traditional motor program approach to the serial order problem. The traditional approach, based on the von Neumann stored program, assumes that motor actions are instructions that are assembled into a structure which is then scanned by a sequential processor. The parallelism and interactiveness of real behavior prove burdensome to such an approach, and typically, extra mechanisms must be invoked. In the current approach, on the other hand, parallelism is a primitive, arising directly from the continuity of the mappings defining the system. Strictly sequential performance is simply the limiting, most highly constrained case.

The concept of state is central to the current theory. Time is represented implicitly by the configuration of the state vector, and it is the assumption of a continuously varying state that relates nearby moments in time and provides a natural way for behavior to be parallel and interactive locally in time while still broadly sequential. The similarity structure of the underlying state should provide, according to the current approach, a theoretical point of convergence for many kinds of behavioral data. The pattern of coarticulation depends on this similarity structure, errors are more likely when discriminations must be made between similar states, dual-task interference is a function of similarity, and learning is faster when similar actions are associated to similar states. Thus, if the theory is to prove useful, elucidation of the similarity structure of the states underlying sequential behavior becomes an overriding theoretical and empirical concern.

REFERENCES

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

Albus, J. A. (1981). *Brains, behavior, and robotics*. Peterborough, NH: BYTE Books.

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

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*, 835-846.

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.

Dell, G. S. (1984). Representation of serial order in speech: Evidence from the repeated phoneme effect in speech errors. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *10*, 222-233.

Dell, G. S., & Reich, P. A. (1980). Toward a unified model of slips of the tongue. In V. A. Fromkin (Ed.), *Errors in linguistic performance: Slips of the tongue, ear, pen, and hand* (pp. 273-286). New York: Academic Press.

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

Estes, W. K. (1972). An associative basis for coding and organization in memory. In A. W. Melton & E. Martin (Eds.), *Coding processes in human memory* (pp. 161-190). Washington, DC: Winston.

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.

Gelfer, C. E., Harris, K. S., & Hilt, G. (1981). *Temporal constraints on anticipatory coarticulation* (Status Report on Speech Research SR-77/78). Haskins Laboratories.

Gentner, D. R. (1985). *Skilled motor performance at variable rates: A composite view of motor control* (CHIP Report 124). La Jolla: University of California, San Diego, Center for Human Information Processing.

Greene, P. H. (1972). Problems of organization in motor systems. In R. Rosen & F. M. Snell (Eds.), *Progress in theoretical biology* (Vol. 2, pp. 303-338). New York: Academic.

Grossberg, S. (1978). A theory of human memory: Self-organization and performance of sensory-motor codes, maps, and plans. *Progress in Theoretical Biology, 5*, 233-302.

Grudin, J. G. (1981). *The organization of serial order in typing.* Unpublished doctoral dissertation, University of California, San Diego.

Grudin, J. T. (1983). Error patterns in skilled and novice transcription typing. In William E. Cooper (Ed.), *Cognitive aspects of skilled typewriting* (pp. 95-120). New York: Springer-Verlag.

Halwes, T., & Jenkins, J. J. (1971). Problem of serial order in behavior is not resolved by context-sensitive associative memory models. *Psychological Review, 78*, 122-129.

Hammarberg, R. (1982). On redefining coarticulation. *Journal of Phonetics, 10*, 123-137.

Harris, K. S. (1984). Coarticulation as a component in articulatory description. In R. G. Daniloff (Ed.), *Articulatory assessment and treatment issues* (pp. 190-227). San Diego: College-Hill Press.

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

Hinton, G. E. (1984). Parallel computations for controlling an arm. *Journal of Motor Behavior, 16*, 171-197.

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

Hinton, G. E., & Sejnowski, T. J. (1983). Optimal perceptual inference. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 448-453.

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. (1985). *The learning of representations for sequential performance.* Unpublished

doctoral dissertation, University of California, San Diego.

Kelso, J. A. Scott, Saltzman, E. L., & Tuller, B. (in press). The dynamical perspective on speech production: Data and theory. *Journal of Phonetics.*

Kent, R. D., Carney, P. J., & Severeid, L. R. (1974). Velar movement and timing: evaluation of a model for binary control. *Journal of Speech and Hearing Research, 17*, 470-488.

Kent, R. D., & Minifie, F. D. (1977). Coarticulation in recent speech production models. *Journal of Phonetics, 5*, 115-133.

Kohonen, T. (1977). *Associative memory: A system theoretical approach.* New York: Springer.

Ladefoged, P. (1982). *A course in phonetics* (2nd ed.). New York: Harcourt, Brace, Jovanovich.

Lashley, K. S. (1951). The problem of serial order in behavior. In L. A. Jeffress (Ed.), *Cerebral mechanisms in behavior* (pp. 112-136). New York: Wiley.

MacKay, D. G. (1981). *A general theory of serial ordering in behavior.* Paper presented at the Twenty-Second Annual Meeting of The Psychonomic Society.

MacKay, D. G. (1982). The problems of flexibility, fluency, and speed-accuracy tradeoff in skilled behavior. *Psychological Review, 89*, 483-506.

Marr, D., & Poggio, T. (1976). Cooperative computation of stereo disparity. *Science, 194*, 283-287.

McClelland, J. L., & Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: Part 1. An account of basic findings. *Psychological Review, 88*, 375-407.

Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the structure of behavior.* New York: Holt, Rinehart, & Winston.

Minsky, M. L., & Papert, S. (1969). *Perceptrons.* Cambridge, MA: MIT Press.

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). Categorization of action slips. *Psychological Review, 88*, 1-15.

Norman, D. A., & Bobrow, D. G. (1975). On data-limited and resource-limited processes. *Cognitive Psychology, 7*, 44-64.

Ohman, S. E. G. (1966). Coarticulation in VCV utterances: Spectrographic measurements. *Journal of the Acoustical Society of America, 39*, 151-168.

Perkell, J. S. (1980). Phonetic features and the physiology of speech production. In B. Butterworth (Ed.), *Language production: Vol. 1. Speech and talk* (pp. 337-372). London: Academic Press.

Poggio, T. (1975). On optimal nonlinear associative recall. *Biological Cybernetics, 19*, 201-209.

Povel, D.-J., & Collard, R. (1983). Structural factors in finger tapping. *Acta Psychologica, 52*, 107-124.

Rosenbaum, D. A. (1980). Human movement initiation: Specification of arm, direction, and extent. *Journal of Experimental Psychology: General, 109*, 444-474.

Rosenbaum, D. A., Kenny, S. B., & Derr, M. A. (1983). Hierarchical control of rapid movement sequences. *Journal of Experimental Psychology: Human Perception and Performance, 9*, 86-102.

Rosenblatt, F. (1962). *Principles of neurodynamics.* Washington, DC: Spartan.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep. 8506). La Jolla: University of California, San Diego, Institute for Cognitive Science.

Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tenses of English verbs. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 2: Psychological and biological models.* Cambridge, MA: MIT Press/Bradford Books.

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

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

Saltzman, E. L., & Kelso, J. A. Scott (in press). Skilled actions: A task dynamic approach. *Psychological Review.*

Sejnowski, T. J., & Rosenberg, C. R. (1986). *NETtalk: A parallel network that learns to read aloud* (Technical Report 86/01). Department of Electrical Engineering and Computer Science, Johns Hopkins University.

Shaffer, L. H. (1976). Intention and performance. *Psychological Review, 83*, 375-393.

Shaffer, L. H., & Hardwick, J. (1970). The basis of transcription skill. *Journal of Experimental Psychology, 84*, 424-440.

Shiffrin, R. M., & Schneider, W. (1977). Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. *Psychological Review, 84*, 127-190.

Sternberg, S., Monsell, S., Knoll, R. L., & Wright, C. E. (1978). The latency and duration of rapid movement sequences: Comparisons of speech and typewriting. In G. E. Stelmach (Ed.), *Information processing in motor control and learning* (pp. 117-152). New York: Academic Press.

Wickelgren, W. A. (1969). Context-sensitive coding, associative memory, and serial order in (speech) behavior. *Psychological Review, 76*, 1-15.

Widrow, G., & Hoff, M. E. (1960). Adaptive switching circuits. *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4*, 96-104.